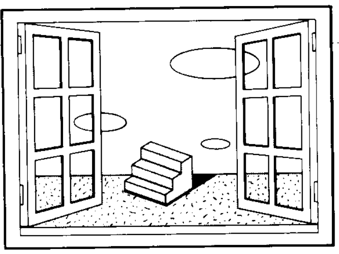

A Construction Set for Multimedia Applications

Matthew E. Hodges, Russell M. Sasnett, and **Mark S. Ackerman**
Project Athena

The Athena Muse system combines four representation schemes to simplify the construction of multimedia educational software that lets teachers and students explore subjects from many views.

"Content vanishes without form. We can have words without a world but no world without words or other symbols," wrote Nelson Goodman in his book *Ways of World Making*[1] In some respects, as Goodman points out, world making is a symbolic affair. Especially in the case of computer-based realities, worlds exist only as descriptions. World makers are confined to the expressive limits of their symbol systems, and so the range of possible worlds is limited by the available means for defining them.

With $100 million in funding primarily from Digital Equipment Corp. and IBM, the Massachusetts Institute of Technology established Project Athena as an eight-year research program to explore innovative uses of computing in the MIT curriculum. One focus is to create an experimental construction set for building multimedia learning environments. The system, called Athena Muse, was designed to support MIT faculty in developing worlds — fictional places for students to experience remote or intangible phenomena, to traverse conventional structures of knowledge, and to practice the skills of research.

The goal of Muse is to reduce the time and skill needed to construct such learning environments. The basic approach is to diversify the tools of symbolic representation available to the computer-based world maker. Rather than restrict the world's author to just one method, as most authoring systems do, our aim is to offer several complementary approaches to better match the patterns of human expression and representation.

To create the hardware platform for Muse, the project's Visual Computing Group modified a standard Athena workstation to support 256-color graphics as well as full-motion digitized video. The group, which focuses on the uses of still- and full-motion imagery in educational software, supports MIT faculty members

0704-7459/89/0001/0037/$01.00 © 1989 IEEE

Target applications

The following projects provided the principal design targets for the Muse Application Construction Set.

• Athena Language Learning Project. This project had two applications, Direction Paris (Getting Around Paris) in French and No Recuerdo (I Don't Remember) in Spanish. Direction Paris has two parts. "À la Rencontre de Philippe" ("A Meeting with Philippe") is a story about a young writer who must solve his romantic problems or find a new apartment in Paris. *Dans le Quartier St. Gervais* (*In the St. Gervais Neighborhood*) is a video documentary of the real neighborhood where this fictional story takes place. *No Recuerdo* was filmed in Bogota, Colombia, and features a maze-like series of plots and simulations: Students help Gonzalo, a Colombian scientist, recover his memory and the secret substance he created before the onset of an amnesia plague.

• Neuroanatomy Learning Environment. A flexible learning environment for the exploration of the anatomy of the human brain, the project was designed for medical and neuroscience students. The major components of the project include an illustrated glossary, a 3D rotation of the brain, a slide collection, and documentary footage of a brain dissection.

• Boston Architecture Collection. Using MIT's Boston Architecture Collection, this videodisc and database provide resources for the study of city planning, architecture, and historical development for Boston from 1620 to today.

• New Orleans: A City in Transition. With more than five hours of documentary film, this project records the changes in New Orleans architecture caused by the 1984 World's Fair there. The materials are used for the study of urban planning, social policy, and documentary-film technique.

• Navigation Learning Environment. This Digital Equipment Corp. videodisc project was developed to explore techniques and tool requirements for producing highly interactive computer-based learning materials.

in the development of both audiovisual content and software for multimedia applications.

The Visual Workstation uses either an IBM RT PC or DEC MicroVAX II workstation as a base with a Parallax Graphics board added as the display subsystem. The Parallax board digitizes a standard NTSC video signal in real time. The video images are presented on the display through the X Window System, where they can be managed as any other X Windows application. Normally, the video images are drawn from an optical videodisc player, but they can also be received from the campus cable television network.

We developed Muse under a strongly application-driven approach. In general, people design tools to solve production problems. The eventual utility and scope of such tools depend on the clarity and magnitude of the problems that shape their design. For the Muse prototype, we were favored with a broad set of problems. Our initial set of target applications included two foreign-language simulations, a neuroanatomy database, a collection of images of Boston's architecture, documentary-film footage on the architectural planning for the New Orlean's World Fair, and a simulation of coastal navigation and piloting in Maine.

All these projects were designed by professional educators across a spectrum of disciplines (the box above briefly describes these projects). They represent all the major approaches to using computers in education: simulation, tutorial, reference, and educational tools. Each project presented its own unique requirements, and each brings its own variations to several common themes.

From the developer's standpoint, Muse provides some of the basic capabilities of a user-interface management system for defining user-interface components and linking their behavior to application structures. The Muse architecture provides a data model and a tool set for creating structured multimedia documents and for linking these together in complex networks. It also can perform limited forms of dynamic system modeling and discrete simulation.

The work described in this article represents the initial efforts to build from the target applications a coherent set of tools capable of addressing the problems they raise. These capabilities rest on four distinct representational approaches: directed graphs, multidimensional spatial frameworks, declarative constraints, and a procedural language. We integrate these approaches so a developer can use all or any of them, defining each part of an application with the most suitable approach.

Directed graphs

In educational computing, the distinction between electronic documents and application software is blurring. Electronic documents are no longer static constructions sharing the limitations of their paper-based counterparts. With the advent of multimedia workstations, electronic documents are becoming richly choreographed presentations, often with complex user interaction. Interactive information processing and display have become part of the expressive medium, and a document must now be viewed more as a set of dynamic resources than as a simple analog of the printed page.

A multimedia authoring environment must provide resources to handle conventional documents but also provide facilities to create interactive programs. One representation system that straddles both areas is the directed graph, so we chose it as a starting point for the Muse architecture.

Example structures. In Muse, materials are grouped into information packages, which combine text, video, and graphics. When a package is activated, the display elements that make up its contents are presented on the screen. These packages can be linked together in directed-graph structures, as shown in the following two examples.

• Hypermedia. The directed graph plays a fundamental role in the hypermedia paradigm, where the aim is to manage extensive cross-references among a set of documents. Normally, each document becomes a node in the graph, with cross-references signified by the arcs or edges connecting them. Research at Brown University's Iris project[2,3] and other hypermedia systems[4] have shown the utility of this approach in designing educational materials.

The Neuroanatomy Learning Environment project at Athena uses this structure. It has 1,400 descriptive text documents and an archive of still and motion images of the brain. The documents are cross-referenced to one another and also to image regions where anatomical structures appear. This network of nodes is built on a concept structure that supports the linking of high-level abstractions rather than the simple linking of data objects.

To implement hypermedia applications, the Muse packages are linked in a network. Cross-references are fired by activation signals sent from one package to another. The display of information is controlled by the activation and deactivation of packages, so a whole chain of packages can be displayed as the cross-references are triggered.

• State-transition networks. In a different branch of educational computing — one related more to simulation than document management — the directed graph again emerges as a fundamental organizational structure. In the state-transition network, nodes of the directed graph represent a system's finite states and arcs indicate branching paths from one state to another.

The Language Learning Project uses two forms of this model. The first is a branching movie, which is a simulated interaction with a cast of fictional characters. Each scene occupies a node in the network. The order of scenes is based on input from the viewer, which is generally gathered at the end of each scene. The film's entire structure can be represented as a tree, where viewer choices determine which path through the tree is followed and, ultimately, which conclusion is reached.

The second example is a fixed-route surrogate travel system. In a French-language application, Direction Paris, students are asked to visit and explore apartments in Paris as part of the interactive story. The apartments are represented as sequences of photographs that step through the various rooms. In this case, each sequence of photographs becomes a node in the network. The branch points are intersections in the apartment where the student can choose to turn left or right, continue straight ahead, or turn around. Here, the graph does not follow a tree structure but is a network of nodes (see Figure 1).

Limitations. While using the directed-graph model extensively, the foreign-language projects also indicate some limitations that this approach has as a representational scheme. These projects' design calls for dynamic subtitling for the

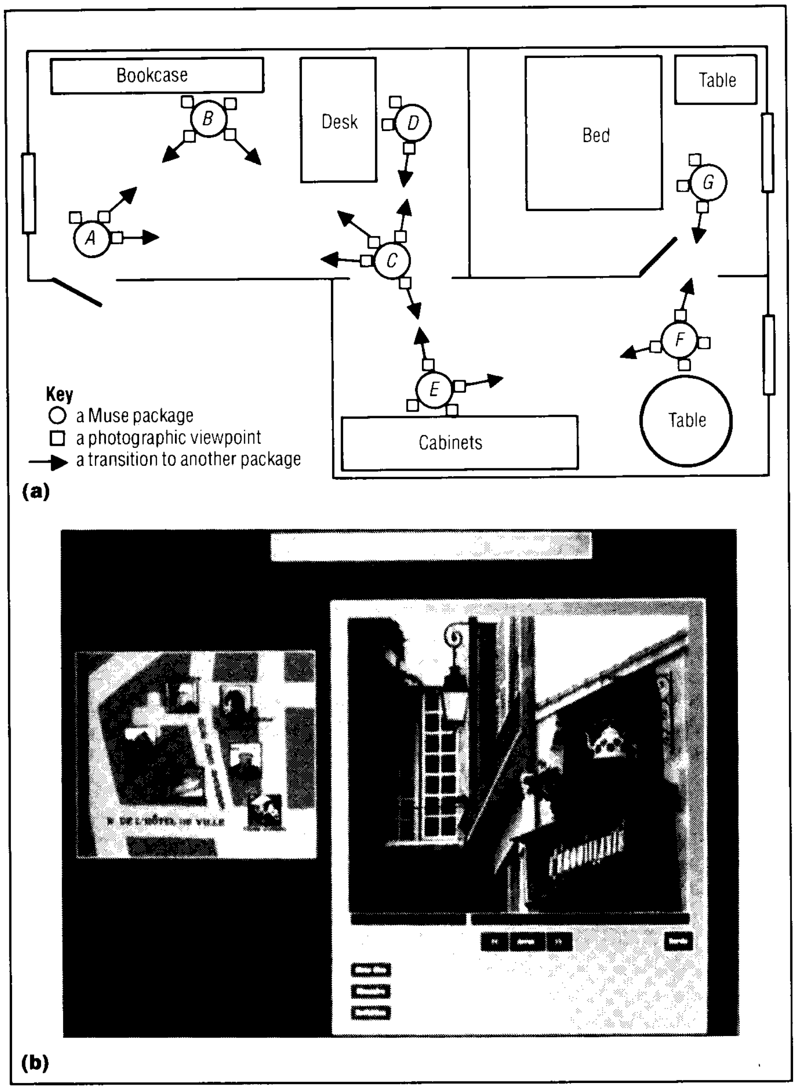

Figure 1. (a) This floor plan of an apartment in the Direction Paris application shows how Muse packages arranged in a directed graph can construct an interactive surrogate travel environment. Each package contains a series of photographs that show the passage from one point to another. The current position of the dimension determines which photograph is displayed at any time. **(b)** A Direction Paris photograph.

full-motion video. Subtitle texts are displayed on the screen phrase for phrase with the native speakers. Also, intermittent cultural flags appear when the speaker uses an idiomatic expression or references something of cultural interest.

Students control the video: They can stop at any point and back up an arbitrary amount to repeat the presentation. The problem, of course, is that with the subtitles and cultural flags, this means the text must be rewound to stay matched with the video.

We could have achieved this with a directed-graph model, but it would have been an awkward implementation. Instead, we used a second — and very different — system of representation, one that uses a spatial framework to organize information.

Multidimensional information

Information like that of the foreign-language subtitles is a matter of timing. In such cases, we use a conventional time line with onset and offset times for each information unit. The time line implies a structure fundamentally different from the spaceless web of nodes that make up a directed graph. In this case, time functions as a dimension, with information or-

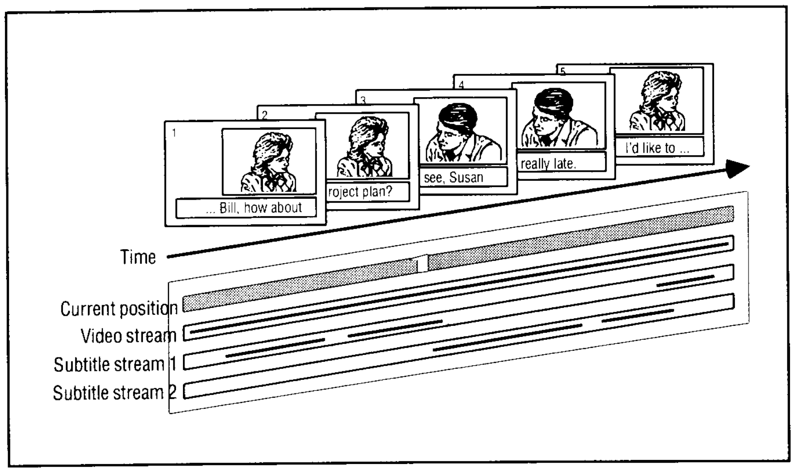

Figure 2. Internal structure of a package showing the synchronization of subtitles and full-motion video. The time dimension controls the choreographed display of text and video.

ganized by position. Muse lets spaces of arbitrary dimensionality exist within each package. Dimensions need not represent real time or physical space; they can represent any changing value in a dynamic system.

First dimension. The subtitles for foreign-language films described earlier are a good example of how to use dimensionality (in this case, one dimension). The easiest way to couple subtitles with video is to create links between frame numbers on the videodisc and text fragments in the subtitle stream. With this arrangement, the video stream becomes the representative of time in the display and the subtitles look directly to the video for timing information.

Instead of using frame numbers directly, we created a layer of abstraction between the two channels. We used an independent dimension of time and attached both the video and text to this abstraction separately. This way, no two components of the display are directly tied to one another, which makes it easy to add and remove channels of information without disrupting the whole presentation.

The dimensions themselves are bounded integer ranges, with maximum and minimum values. Each dimension maintains a current position in its range. Set-position signals adjust the current positions. User-interface controls, such as on-screen command buttons and scroll bars, can generate these signals, as can the system clock as it drives time forward and thus determines when the text and video appear and disappear. You can attach any number of channels to the dimensions, so a dimension of time can handle multiple streams of subtitles, cultural flags, and so on, in a complex display choreography (as Figure 2 shows).

This provides a general mechanism for annotating video material. With the spatial framework, it becomes possible to cut and paste video without losing the annotations. This capability has utility in several Project Athena target applications.

For example, the New Orleans: A City in Transition project has a database of three hours of full-motion documentary footage stored on videodisc. These materials were produced in 1983-84 and document the planning and development that took place for the New Orleans World's Fair. The video is being used in both urban planning and filmmaking classes. Students are asked to draw illustrations from this archive and to prepare small documentary pieces representing different interests and perspectives. It is important for the users to be able to attach annotations and data references to specific points in the video.

Second dimension. It's a small step from one to two dimensions, where x and y dimensions define a Cartesian space. Again, the dimensions function as an abstract spatial framework to organize and manipulate display data.

Such a space is commonly used to manipulate images on the screen—zooming and panning, for example. Once the image is mapped to the dimension space, these effects are achieved by manipulating the dimensions. Any change in the current position of x or y affects the image display. In this respect, the dimensions provide a coherent system for handling different kinds of display information.

The Muse prototype lets the x and y dimensions be defined as virtual coordinates, so the display window becomes a viewport into a larger space. Zooming and panning over an image become a matter of moving the viewport within this virtual space and changing the scope of its display. All the display materials, including still video images, are scaled and repositioned appropriately.

The neuroanatomy project uses this technique to view cross-sectional photographs of the brain. The student can zoom in on any region of an image. Because Muse defines the image in a virtual space, it can resolve the coordinates of anatomical features no matter where the features appear on the screen or at what scale they are displayed. This lets different areas be tagged as endpoints to a cross-reference while still letting the student manipulate the image.

You can apply the same technique to view more than one image. Consider the Boston Architecture Collection project, which has an archive of 30,000 images of Boston architecture from MIT's Rotch Visual Collection. These images are used for comparative analysis of architectural style; the teacher or student typically wants to select images and organize them into presentations.

A query to the Boston database will return a list of images, which can then be displayed and manipulated in a 2D virtual space. You can control these images the same way as in the neuroanatomy example, so you can view any subset of images at any scale. Muse allows multiple simultaneous views of this space, so you can create more than one viewport, each showing different regions at different scales. You can add text and graphic annotations and interface-control mechanisms such as scroll bars. As you work, you can organize these virtual light tables into the pages of a personal notebook.

***N*th dimension.** Muse does not limit the number of dimensions that can be created. Dimensions can represent not only phyical space and time but any changing parameter in a dynamic system. Dynamic systems are often modeled as a state space, which is a set of coordinates or dimensions that define the range of movement along each degree of freedom (dimension) in the system. A dynamic function or process governs how the position of the system changes and moves through its state space.

You can use dynamic systems of this kind to implement some forms of simulation. Because simulation is a classic method of using computers in education, it is a highly important focus of the Muse system.

In Muse, the dimensions are well-integrated into the indexing and cross-referencing machanisms. Taking advantage of this, a teacher can easily make cross-references directly into the state space of a dynamic system. This provides a generic method for querying the simulation state, changing the simulation state, and setting trigger conditions in the simulation that can change state in other nodes of the directed graph.

The Navigation Learning Environment project is an example of a time-driven simulation implemented with Muse. This project simulates a boat moving through two square miles of coastal waters in Maine, using a library of about 10,000 images to represent the environment. The entire simulation is implemented as a single interactive document and uses seven dimensions. Two dimensions represent the boat's *x,y* position. Another two dimensions represent the boat's heading and speed (the systems uses these to compute the boat's position). A fifth dimension tracks the user's viewing angle, since you can look in any direction from the boat regardless of its heading. The sixth and seventh dimensions manage a simulated compass that can be positioned anywhere on the image.

The effect is that the user can steer through a virtual world, using command buttons and scroll bars to control the boat and a mouse to take compass bearings directly from the images. The simulation becomes active whenever the document is opened, and the user can choose whether to take control or simply watch. A system timer updates the boat's position every 0.1 seconds.

To implement such a simulation, a fairly high level of orchestration is needed among the various display elements. The directed graph and spatial framework provide a foundation for organizing and controlling the display elements, but you also need some facility to coordinate the behavior of these elements. In many cases, you can define the desired relationships as simple functions and thus implement them as declarative constraints.

Declarative constraints

Constraints are functional bindings within a system that define relationships among different components and automatically propagate change among them.[5] In Muse, constraints are limited to bidirectional equality relations: You can apply such bindings to attributes of the dimensions and display elements.

An example constraint is the relation-

It is usually important to have a visible representation of the dimension, either to control it or to see what state it is in. You can use a scroll bar for both.

ship between a scroll bar and a dimension. It is usually important to have a visible representation of the dimension, either to control it or to see what state it is in. You can use a scroll bar for both operations: If you use the mouse to reposition the slide bar indicator, the change will be propagated to the dimension. Conversely, any changes in the dimension that occur from some other input or timing event will be propagated back to the scroll bar, as Figure 3 shows.

A similar case is a dynamic graphic object that modifies its appearance based on a constraint binding to a dimension. In the navigation project, we created a simulated hand-bearing compass by using constraints. The compass moves to the location of a mouse click in a water-level view of the environment and prints a value between 0 and 359. The system can compute the bearing because it knows the camera

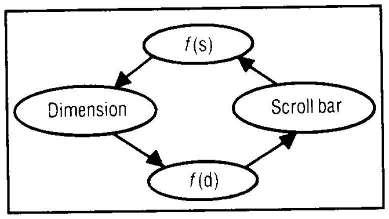

Figure 3. The appearance of scroll bar representing time is a function of the time dimension's current position. Conversely, you can control the current value of time by dragging the scroll bar.

orientation for the panoramic views.

This example uses three constraints, one each for the *x* and *y* locations on the screen and one for the text label that indicates compass direction. When a mouse click occurs, the system uses the mouse's *x* and *y* position to adjust the compasss's *x* and *y* directions in the simulation document. The changes in these dimensions are then propagated via the constraint to reposition the graphic object. The number printed on the screen is constrained to the compass *x* dimension, which ranges from 0 to 359.

Constraints are a common feature in Muse applications. They range from simple one-way bindings on a graphical object to complex interactions among a group of dimensions used to implement a simulation. They are declarative in that they generally consist of simple mathematical expressions that include references to dimensions and display attributes.

As an editing environment for such constraints, we have experimented with a modified spreadsheet calculator. Our aim is to embed small spreadsheets in the Muse environment as editing utilities for defining and managing constraints. A cell can directly reference a dimension in any package as its input value. Values can be picked up from one or more dimensions, processed through a series of mathematical functions on the spreadsheet, and output to other dimensions or display objects. In a simple case, the spreadsheet can act as a patch panel or routing switch, directing the propagation of data values to different components of the system. In a more complex case, the spreadsheet can maintain the constraint relations that form the core of a simulation.

The declarative constraints in Muse maintain reversible linear functions among the system components. In many cases, however, constraints are not sufficient to define all of the actions and be-

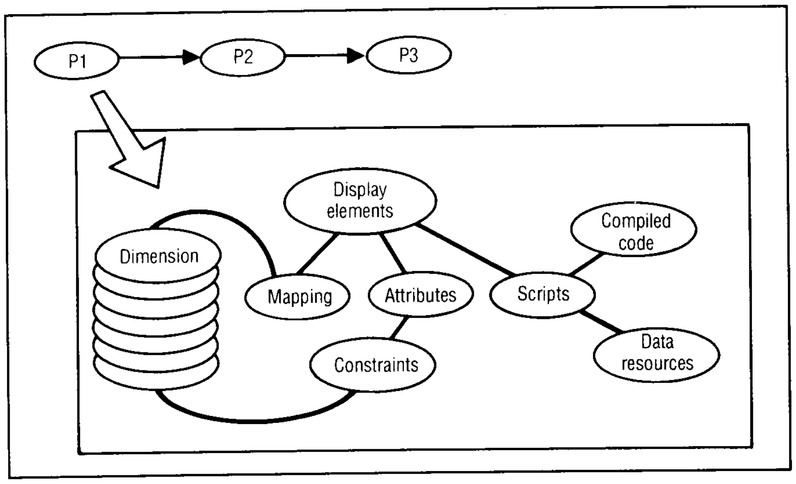

Figure 4. The Muse architecture. P1, P2, and P3 are packages in a directed graph. Each package contains bounded dimensions and display elements. You assign display elements to a position through mapping. You can link attributes of the display elements through unidirectional or bidirectional constraints. Display elements can also have interpreted scripts for specialized user actions. These scripts have access to external data and compiled procedural code.

haviors needed in an application. Procedural descriptions fill this need.

Procedural descriptions

In Muse, one design principle is to minimize the use of procedural code, since data specification tends to be quicker to create and to require less-specialized skill. However, any purely data-driven system is bound to fall short because there will be some application requirements, such as database queries and user interactions, that cannot be accommodated effectively with a strictly data-driven paradigm.

Thus, we included procedural descriptions as one of the four major resources provided to developers. You can incorporate two kinds of procedural materials in Muse applications:

• In the simplest case, single actions can be defined as attributes of Muse packages and their display elements. Often, a single action is sufficient, as in one package sending an activation signal to another. This forestalls the need to use a more formal procedural description.

• In more complex cases, you use an interpreted language called Event Script. Event Script was developed as the procedural component of Muse. It lets you attach special-purpose behaviors to individual display elements. Event Script is a simplified object-oriented language that is used primarily for creating user-interface event handlers. It is not meant to be a general-purpose programming environment.

Event Script follows an uncomplicated syntax, similar to Apple's Hypertalk language, using untyped variables and event-oriented construction.[6] In addition, Event Script permits transparent access to any C variables or functions in the program space. Thus, any object's methods can gain access to general system libraries or resources, providing an extension mechanism to Muse.

Delegation strategy. Event Script uses a variant of the delegation strategy for sharing object behavior.[7] In Event Script, you can specify an object to use the contents of one or more existing prototype objects. If a message handler is not found in the recipient object's methods, a search is begun in the recipient's prototypes (and in their prototypes) until the method is found or the chain is exhausted. If found, the method is then borrowed and executed in the scope of the original object. Event Script addresses the problem of preserving local context through an inheritance chain by built-in functions that return the name of the client and delegate objects, which can be used inside any method. (The client is the original recipient of a message; the delegate is the owner of the borrowed method.)

An important feature of delegation is that any existing object can be duplicated and given a new name. The duplicated object has its own local state variables but shares the method dictionary and prototype list of the original object. This permits great flexibility in defining new objects dynamically. Developers can duplicate and modify prototype objects for specialized applications and their interfaces.

Applications. The primary use of Event Script so far has been as a programmer's interface to X Windows. Objects can be bound to windows; thereafter any X Windows events occurring in that window are translated into messages to the Event Script object. Normally, an Event Script object is associated with a display element in a Muse package. When the display element's window is activated, the object is registered to handle all the window's events. Often the desired behavior is shared by many display elements, as in the case of window exposure events. This situation is well-suited to the method-borrowing form of delegation we have implemented.

Editing in Muse

The human interface begins with the data model. If the data model does not fulfill the purposes of the system as a whole, any efforts on the more superficial aspects of the interface will be largely wasted. Given a sound model to work from, the next step is to design efficient editors and make the development process easier.

One Muse design goal has been to write the editors that manipulate the data model in Muse itself. This work is continuing as a primary focus in the second version of Muse.

Muse uses a direct-manipulation style of editing. One example is a Muse video editor, which we have implemented as a test case. The editor is modeled after a two-buffer text editor, where two buffers of video are displayed on the screen side by side. The two buffers are defined as Muse packages, with time as the principal dimension. Cutting and pasting video segments is a matter of selecting a region of time in one buffer and then cutting all data resources referencing that part of the time dimension. You can then paste these cut materials into the other video stream, just as a text editor would paste text into another buffer. We implemented the cut and paste functions as compiled C functions, but they are controlled through the Event Script section of Muse.

To be useful, the four representations — directed graphs, dimensions, declarative constraints, and procedural descriptions — must be combined into a coherent framework. Muse integrates them into one data model, as Figure 4 shows. We have tried to achieve a system where the strengths and weaknesses of the various approaches complement one another and allow for free interaction so that the combination offers much greater potential than four separate systems would.

The prototype version of Muse has shown promise in speeding up the process of world making that is the basis for teachers and students to explore their subjects and encourage them to investigate many points of view. Muse has also shown promise in reducing the need for specialized programming skills. ❖

Acknowledgment

We thank the MIT faculty members who have carried out the projects for which we developed Athena Muse: Gloriana Davenport, Gilberte Furstenberg, Douglas Morganstern, Janet Murray, Merrill Smith, and Steven Wertheim.

References

1. N. Goodman, *Ways of World Making*, Hackett Publishing, Indianapolis, Ind., 1978.
2. N. Yankelovitch, N. Meyrowitz, and A. van Dam, "Reading and Writing the Electronic Book," *Computer*, Oct. 1985, pp. 15-30.
3. N. Yankelovitch et al., "Intermedia: The Concept and the Construction of a Seamless Information Environment," *Computer*, Jan. 1988, pp. 81-96.
4. J. Conklin, "Hypertext: An Introduction and Survey," *Computer*, Sept. 1987, pp. 17-41.
5. A. Borning, "The Programming Language Aspects of Thing Lab," *ACM Trans. Programming Languages and Systems*, Oct. 1981, pp. 353-367.
6. D. Shafer, *Hypertalk Programming*, Hayden Books, Indianapolis, Ind., 1988.
7. H. Lieberman, "Using Prototypical Objects to Implement Shared Behavior in Object-Oriented Systems," *Proc. OOPSLA 86 [Object-Oriented Programming Systems, Languages, and Applications Conf.]*, ACM, New York, 1986, pp. 214-223.

Matthew E. Hodges is a visiting scientist at the Massachusetts Institute of Technology's Project Athena from Digital Equipment Corp. He has worked at DEC since 1982 in educational research and development. At Athena, he conducts research in educational technology and is manager of software development for the Visual Computing Group.

Hodges received a BA in art from the Pennsylvania State University, a masters in education from Harvard University, and is a doctoral candidate at the Harvard Graduate School of Education.

Russell M. Sasnett is a member of the technical staff at GTE Laboratories and a visiting engineer with the Visual Computing Group at Project Athena. His research interests include rapid prototyping and the design of graphic interfaces to video materials. He is an award-winning documentary filmmaker.

Sasnett received a BA in film and computer science from Dartmouth College and an MS in visual studies from the Massachusetts Institute of Technology.

Mark S. Ackerman is a doctoral candidate in information technology at the Massachusetts Institute of Technology and a research associate at Project Athena's Visual Computing Group. His research interests include information retrieval, reference systems, and human-computer interaction.

Ackerman received a BA in history from the University of Chicago and an MS in computer and information science from Ohio State University.

Address questions about this article to the authors at Project Athena, Bldg. E40-300, 1 Amherst St., Cambridge, MA 02139; ARPAnet hodges@athena.mit.edu.

C++ TRAINING

Developing in C? Why Not C++?

Object-Oriented Programming is clearly the wave of the future and C++ provides an easy path to that future. The use of C++ can capitalize on your current investment in C programs, allow you to continue to benefit from the efficiencies of C, and lay the foundation for new levels of quality and productivity. If you are currently developing software in C, you should be considering migrating to C++. Many large development organizations have already made a full scale commitment to C++.

The **Institute For Zero Defect Software** has extensive experience in training and consulting in C++ based development projects at several major corporations. Our course

C++ Programming for C Programmers

is available for on-site delivery. This three-day course will cover both the C++ language and its effective application. Realistic C++ programs are used for laboratory exercises. The experience gained from these exercises is directly applicable to development projects. Some of the topics covered in this course are:

Introduction to Object-Oriented Programming

■

Data abstraction and C++ classes

■

Class hierarchies, inheritance and virtual functions

■

For more information contact:

Institute For Zero Defect Software
200 Runnymede Parkway
New Providence, N. J. 07974
(201) 668-4508

Reader Service Number 8