DOCUMENT RESUME

ED 459 815 IR 058 351

AUTHOR Yaron, David; Milton, D. Jeff; Freeland, Rebecca
TITLE Linked Active Content: A Service for Digital Libraries for Education.
SPONS AGENCY National Science Foundation, Arlington, VA.
PUB DATE 2001-06-00
NOTE 9p.; In: Proceedings of the ACM/IEEE-CS Joint Conference on Digital Libraries (1st, Roanoke, Virginia, June 24-28, 2001). For entire proceedings, see IR 058 348. Funded as part of the National Science Mathematics Engineering and Technology Education Digital Library (NSDL) program. Figures may not reproduce well.
AVAILABLE FROM Association for Computing Machinery, 1515 Broadway, New York NY 10036. Tel: 1-800-342-6626 (U.S. & Canada); Tel: +1-212-626-0500 (Global); e-mail: acmhelp@acm.org. For full text: http://www1.acm.org/pubs/contents/proceedings/dl/379437/.
PUB TYPE Reports - Descriptive (141) -- Speeches/Meeting Papers (150)
EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Active Learning; *Authoring Aids (Programming); Computer Assisted Instruction; Computer Software Development; Cooperation; Curriculum Development; Education; *Electronic Libraries; Information Services; Learning Activities; *Programming

ABSTRACT

A service is described to help enable digital libraries for education, such as the NSDL (National Science Mathematics Engineering and Technology Education Digital Library), to serve as collaboration spaces for the creation, modification and use of active learning experiences. The goal is to redefine the line between those activities that fall within the domain of computer programming and those that fall within the domain of content authoring. The current location of this line, as defined by Web technologies, is such that far too much of the design and development process is in the domain of software creation. This paper explores the definition and use of "linked active content," which builds on the hypertext paradigm by extending it to support active content. This concept has community development advantages, since it provides an authoring paradigm that supports contributions from a more diverse audience, including especially those who have substantial classroom and pedagogical expertise but lack programming expertise. It also promotes the extraction of content from software so that collections may be better organized and more easily re-purposed to meet the needs of a diverse audience of educators and students. (Contains 20 references.) (Author/AEF)

Reproductions supplied by EDRS are the best that can be made from the original document.

ED 459 815

Linked Active Content: A Service for Digital Libraries for Education

David Yaron
Department of Chemistry
Carnegie Mellon University
Pittsburgh, PA 15213
412-268-1351
yaron@chem.cmu.edu

D. Jeff Milton
Department of Chemistry
Carnegie Mellon University
Pittsburgh, PA 15213
412-268-1065
milton@chem.cmu.edu

Rebecca Freeland
Department of Chemistry
Carnegie Mellon University
Pittsburgh, PA 15213
412-268-7981
rf51@andrew.cmu.edu

ABSTRACT

A service is described to help enable digital libraries for education, such as the NSDL, to serve as collaboration spaces for the creation, modification and use of active learning experiences. The goal is to redefine the line between those activities that fall within the domain of computer programming and those that fall within the domain of content authoring. The current location of this line, as defined by web technologies, is such that far too much of the design and development process is in the domain of software creation. This paper explores the definition and use of "linked active content", which builds on the hypertext paradigm by extending it to support active content. This concept has community development advantages, since it provides an authoring paradigm that supports contributions from a more diverse audience, including especially those who have substantial classroom and pedagogical expertise but lack programming expertise. It also promotes the extraction of content from software so that collections may be better organized and more easily repurposed to meet the needs of a diverse audience of educators and students.

Categories and Subject Descriptors

K.3.1. [**Computers in Education**]: Computer Uses in Education, *computer assisted instruction, computer managed instruction.*

General Terms

Human Factors, Experimentation

Keywords

Education, Active learning, Web authoring

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
JCDL'01, June 24-28, 2001, Roanoke, Virginia, USA.
Copyright 2001 ACM 1-58113-345-6/01/0006...$5.00.

1. INTRODUCTION

A large body of educational research has shown the benefits of active learning, whereby students do not sit passively while they are told information, but rather participate actively in the learning process [13]. Digital libraries for education, such as the NSDL, can help catalyze the shift towards more active, exploratory, inquiry-based learning activities. The digital library's potential lies not only in the ability to bring active learning experiences to an unprecedented and large audience of students, but also in the potential to serve as a collaboration space to catalyze the creation, modification and assessment of student activities. First, a grass-roots development approach, which remains in intimate touch with the needs of teachers and students, may reduce concerns that can prevent adoption of useful teaching innovations. Second, and perhaps more difficult to achieve, if the library is to be populated with a large amount of high-quality material, it will be important to engage a large community of developers and early-adopters, with a wide range of interests, expertise and approaches.

The goal of our research is to provide a technical infrastructure that will help digital libraries, such as the NSDL, overcome some of the main challenges associated with collaborative creation of engaging learning activities. From a community building perspective, creation of these activities requires both technical and pedagogical expertise, a combination of expertise that few members of the NSDL community can be expected to possess. From a collections perspective, the level of interactivity required for engaging activities typically leads to monolithic chunks of software that are difficult to subdivide into components that promote adaptation and reuse. Our premise is that these challenges are intimately coupled, since they both relate to where one draws the line between *software creation* and *content authoring*. The current location of this line, as defined by available web technologies, is such that far too much of the design and development process is in the domain of software creation. By pushing this line to allow for more powerful and flexible content authoring, we can better utilize the expertise of those members of the community with curriculum development and classroom experience. Redefining the line between programmer and author also promotes the extraction of content from software in a manner that leads to better organized collections that may be repurposed to meet the needs of a diverse audience of educators and students.

PERMISSION TO REPRODUCE AND DISSEMINATE THIS MATERIAL HAS BEEN GRANTED BY

D. Cotton

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

BEST COPY AVAILABLE

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)
This document has been reproduced as received from the person or organization originating it.
Minor changes have been made to improve reproduction quality.
Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

[R058351

2. RELATED WORK

Over the last 40 years, there have been thousands of individual efforts to create science and engineering teaching and learning materials in digital formats, ranging from case study depositories to instructional software. Some have been quite effective, such as the Physics Academic Software (PAS) library [4]. It is now widely recognized, however, that developing configurable learning objects that will be reused is essential to vitalizing online instruction. The efforts of groups including the Instructional Management Systems group started by EDUCAUSE [7], the IEEE Learning Technology Standards Committee [11], the ARIADNE project [3], the Advanced Distributed Learning Initiative [5], and others to create standards for learning objects is partly motivated by the reality that reuse is rare. Most of the standards currently being developed revolve around textual content (metadata descriptions for content, standards for questions and answers, and curriculum structure standards), and not around software development and reuse.

It has long been hoped that instructional software developers would contribute to libraries of instructional software that would give less technically oriented educators access to the benefits of software simulations and interactive exercises. There are a number of significant new efforts at creating architectures for developing and sharing educational software components [2, 6, 8, 9, 10, 12], most of them based on the Java language. The ESCOT [9] project is a testbed that seeks to encourage development and re-use of learning objects, with a current emphasis on middle-school mathematics. The NEEDS [14] project to create a digital library for engineering education includes educational software on their site. The National Science Foundation Computer Courseware Repository [15] promises inclusion of instructional software but posts little to date

Current tools to invite participation of instructors with little programming expertise include general authoring environments, such as Macromedia Director and Authorware or Macromedia's Web-based CourseBuilder. While these are powerful and useful tools, they do not have a smooth integration path for coupling to repositories of Java objects, and so are not sufficient to meet the goals addressed here. They also present users with a steep learning curve. Another approach to the construction of tools for this audience starts with general component assembly models such as the JavaBeans model and simplifies and/or provides support to make this approach accessible to a broader audience. Development of such tools is an active research area, for instance at the ESCOT project. As discussed further below, our approach is related to this one but, rather than starting with component assembly and reducing its complexity, we start with Web content creation and extend its abilities. This leads to tools that complement those of other projects, and invite a much broader authoring audience.

3. LINKED ACTIVE CONTENT

One approach to redefining the line between educational software creation and content authoring is to view the creation of curricular materials as primarily *programming* and simplify this activity so that it is accessible to more instructors. Our approach starts from a very different perspective, that of curriculum creation as *Web authoring*. We believe that this approach has advantages in learnability, organizing and reusing content, and supporting domain-specific software components. Most instructors are already familiar with the hypertext link and image map functionalities of Web authoring. Grounded in this model, we extend it to include links to and between simulations and other active learning objects such as tutorials or animations. This builds on the familiar paradigm of hypertext linking: it is intuitively simple to go from creating a hypertext link to sending a message to a learning object. While simple, this linking ability significantly increases the ability of a non-programmer to create interesting active learning experiences.

In addition to the community building advantages discussed above, linked-active content also has advantages for the structure of a digital library for education. Consider the current manner in which active content is supported on the web. Currently, the limitations of HTML are overcome with plug-in technologies, such as Flash or Java, that essentially provide embedded browsers for content stored in a format that has little or no relation to the overall HTML in which the plug-in is embedded. Without good communication between the plug-in content and the overall HTML, even text and image portions of the software, which could be handled with HTML, must instead be embedded in the plug-in. This embedding leads to large chunks of content that are difficult to adapt and reuse. Although this active content often consists of sub-objects with relationships, similar to a hypertext web site, the limits of the plug-in technology prevent this substructure from being made explicit in a manner that can lead to better organized digital library collections.

To catalyze the creation of good active learning experiences, the infrastructure should not only support the creation of domain-specific software components, but actively promote coupling of new components to existing ones. For many of the activities that drive the web, such as product advertising and online sales, the functionality provided by HTML and plug-ins is sufficient. However, the creation of active learning experiences involves a mix of cross-domain components, such as those that handle text and images, along with domain-specific components, such as that in scientific simulations.

Consider a Virtual Lab for chemical education that provides a flexible simulation in which students may perform a large variety of experimental procedures in a manner that mimics that of a real laboratory [18]. While this flexibility supports a wide variety of approaches to chemical education, many approaches call for providing students with guidance so that they interact with the simulation in a meaningful way [19]. This guidance is typically *text* and *image* based: what to do at a certain stage, explanations of concepts, questions to be answered, etc. While much of the functionality needed to provide this guidance is domain independent, educational software presents a rather unique need for both general and domain-specific components. A flexible means to present text and images, such as that in HTML, can be made much more powerful by allowing the text and images to link to simulations. In the Virtual lab example, it should be possible to provide text links and buttons that cause chemical solutions to appear in the virtual lab, or that check the current state of the lab to see if the student has achieved a certain goal. This design promotes reuse of components, since each domain and each software project do not need to reinvent the general tools, but need only provide domain-specific software components.

4. PROGRAMMER AND AUTHOR ROLES

Producing a shift in the line between software creation and content authoring requires not just a change in technical infrastructure, but also a change in how programmers and educators view their roles. Although a programmer must create learning objects, a major goal of our work is to promote a change in the mind-set of the programmer, away from creating finished pieces of educational software and towards creation of learning objects that serve as viewers and manipulators of content.

The issues we are addressing may then be viewed as a specific instance of the more general objective of creating useful components for educational software. Our project is a collaboration with Andries van Dam and Anne Spalter of Brown University, who are considering the creation of components for use by programmers. Our emphasis here is on the creation of components for use by instructors. These instructor components are to be created by programmers, ideally using the software components developed by our Brown collaborators.

5. LIBRARY SERVICES

The goal of our research is to explore the requirements and benefits of allowing links between active content. Active content is that content contained in scientific simulations, tutorials, and other materials that, due to their interactive nature, go beyond the abilities of HTML and so are constructed using JAVA or a web plug-in technology. We will refer to the software components that present this content as learning objects. By "linked active content", we mean information passing between learning objects through conduits that are created during the authoring process rather than being hard-coded into the objects themselves. Such links can pass a message to a learning object, or query an object for data. (More complex means of transferring information between objects are discussed in Section 8.4.)

A principal motivation of this approach is to give an intuitive yet powerful ability to curriculum authors and instructors. It is not a big leap to go from creating a hypertext link to sending a message to a learning object, especially if the link creation is done through dialog boxes as described below. For example, in authoring curriculum around a combustion engine simulation, the author could insert the text "click here to start the engine", with the word "here" being a link that sends the "start" message to the engine. The extended link is similar to a method call in an object oriented programming language, and has the logical structure *target:action:data*, stored via XML.

The output of a completed authoring process is an XML file describing the initial arrangement and state of the learning objects, and the links between these objects that will drive the simulation and react to the student's interactions with it. A viewer, implemented in Java, is used to create the learning environment specified by this file. Note that the XML file is not a script file, but rather a set of linked objects with an initial configuration, analogous to a web site containing a set of linked documents and an initial entry page. This outcome reflects the shift from a simplified-programming to extended-web-authoring paradigm. Unlike current component assembly tools, where errors are typically reported by a compiler in a language that is often obtuse even to experienced programmers, the only types of errors that can arise here are broken links, and even these can be avoided via automated link-generation facilities.

The viewer is a browser that supports dynamic loading of Java objects, and that supports our extending linking structure between these objects. The responsibilities of the viewer are intentionally minimized to make the environment maximally extensible. The functionality resides in the learning objects (implemented as Java classes), and in the links between these objects, not in the viewer itself. We provide implementations of core objects, discussed below, that bring extended linking to web functionalities such as hypertext and imagemaps. However, these learning objects can be swapped out for other implementations. The viewer handles only the extended linking of these objects. Even the placement of objects on the screen is done using a replaceable learning object, analogous to the layout managers of Java. Our initial implementation of the layout manager mimics HTML frames in order to capitalize on potential instructor familiarity.

The environment consists of:

- **Core learning objects** such as a hypertext viewer and image-map viewer that build on their web counterparts by supporting extended linking. These learning objects consist of cross-domain objects for viewing text, images, setting up navigation structures, assessment and grading tools, etc.
- An **authoring environment**, that provides a simple graphical means for arranging the objects and creating links, and that helps the curriculum author organize the potentially large number of text snippets and image maps that make up a complete tutorial or exercise. The organizational aspects are handled by a file-explorer interface that presents the author with a hierarchical list of the corresponding text and image maps. This environment gives the author access to the library of learning objects that can be used to construct simulations. As objects are added to the library, the range of simulations that can be created grows.
- **A viewer**, implemented in Java that creates the learning environment described in the XML file output by the authoring environment.

We are also developing domain-specific learning objects for chemical education, such as a virtual chemistry lab and other simulations. All of the software components use the Java Beans API to expose the methods that will accept messages via the linking mechanism.

6. EXAMPLES

A major goal of our research is to explore the authoring flexibility that may result from allowing linking between active content. Just as hypertext brings more power to text authoring than may have been expected given its relative simplicity, linking of active content leads to considerably more power and flexibility in the construction of student activities than we anticipated at the start of this project. We will attempt to illustrate this power through the following two examples.

6.1 Mission to Mars

In creating a student activity that utilizes a scientific simulation, the role of the programmer is to develop the simulation while that of the curriculum author is to guide student interaction with the simulation. Under current web technologies, a programmer may provide the simulation as a Java applet, which can then be placed

on a web page along with text and image maps that guide the student interaction. The text, especially if placed in a frame below the applet, can take full advantage of the hypertext abilities of HTML to present the material in an appropriate order and even provide nonlinear paths through the material. However, this text remains disconnected from the simulation.

One use of the extended linking mechanism we are developing is to allow the text to pass messages to the simulation.

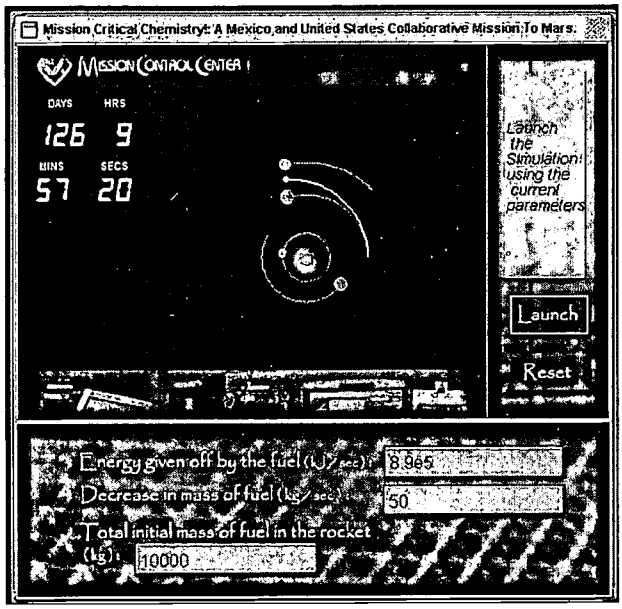

Figure 1. The Mars Simulation exercise with simulation frame (top left), control frame (top right) and parameter frame (bottom).

Figure 1 shows a simulation of a rocket trajectory from Earth to Mars, built using our authoring system. A programmer created the trajectory simulation as a Java component with methods that set the fuel characteristics and other simulation parameters. The author may then use the core learning objects describe in Section 5 to guide student interaction with the simulation. Here, the author has created three frames: the *simulation_frame*, which contains the trajectory simulator object, and a *control_frame* and *parameter_frame*, each of which contains an image-map viewer.

The image-map viewer is a core learning object that builds on the paradigm of the standard web image map. A region (ellipse or polygon) of an image can be made into a "hot spot" that responds when the pointer passes over it, or is clicked/double-clicked. The authoring environment allows the author to create some simple but useful responses to activating a hot spot. For instance, in the *control_frame* of Figure 1, the pointer has activated a hot spot over "Launch" that brings up "bubble help" describing the action of this menu item. (Various filters are provided to allow the look of the bubble help to be customized.) Hot spots may also pop up additional images or a menu of links. In this case, clicking on the Launch hot spot sends a launch message to the trajectory simulator via the extended link: *trajectorySimulator:launch.*

The image map also supports editable hot-spots, which serve as entry boxes for user input. For instance, in the *parameter_frame* of Figure 1, the student may enter and edit text. This text is then passed as data to the method associated with the hot spot. For instance, the upper most text box sends the message *trajectorySimulator:setHeat:8.9e5*. In this manner, the author is allowed to create simple control panels to the simulation. The construction of the control panel is quite different from the approach of JAVA Bean assembly environments. Here, the author first creates an image of the control panel using a tool such as Photoshop. The author then uses our imagemap editor tool to make the relevant portions of the image hotspots that link to the simulation.

In our current implementation, the author must type in the link using appropriate syntax. However, we are currently developing dialog-driven link generation. In this manner, when the author chooses a hot spot for a link, the authoring environment presents a dialog box showing possible targets for the link, such as a list of frames into which a new learning object might appear in response to the student clicking on the hot spot. For example, the author could select *simulation_frame* as the location where the object would appear. This would then prompt for the desired learning object from the library. Once a learning object is selected, the author is prompted with a list of instructions that can be sent to the chosen learning object. For example, a petri dish simulation for growing bacteria could have methods *setGrowthRate, startGrowth,* and *stopGrowth.* Finally, if the chosen method requires data, such in *setGrowthRate,* the curriculum author would be prompted for the relevant data. Thus, the author stipulates how a simulation will behave in response to student actions by doing little more than making hyperlink connections. Note also that more than one link can be attached to a single hotspot or link.

6.2 Pathogen

While conceptually simple, the functionality described above is quite powerful. At the simplest level, providing one frame containing text next to a frame containing a simulation allows authoring of a tutorial that guides a student through the simulation. It is also possible to create the illusion of moving through a virtual world by placing links on imagemaps that load other imagemaps, as is common in computer games such as King's Quest or Myst. Clicking on a door loads an image in which the door is open, and clicking again loads an image of the next room. Such images can be easily obtained with a digital camera. In addition, components may be designed that fit into the environment and provide the author with considerably more power, as illustrated by our Pathogen exercise.

The Pathogen exercise addresses topics typically covered in the first few weeks of an introductory college or high school chemistry course. These topics are put in the context of drug discovery, and in Figure 2, students lead a team of researchers on an island in search of plants with medicinal activity. (They will later bring these plants back to a pharmaceutical laboratory and help determine the active ingredients.)

BEST COPY AVAILABLE

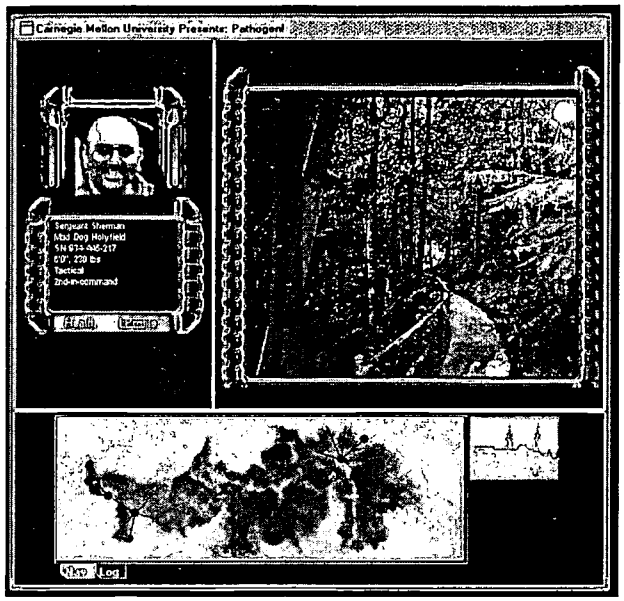

Figure 2: The Pathogen student exercise.

Occasionally, a team member becomes infected and must be given a appropriate drug to be cured. In determining the type and amount of drug, the student must solve a problem involving chemical concepts. If the drug is not appropriate, the team member is airlifted off the island and to a local hospital. The number of initial team members then sets the number of problems the student may get wrong before needing to get a new team and start again.

To support the navigation required by this application, a programmer created a map navigation component, which is loaded into the lower frame of Figure 2. The component shows a map of the island and the current location of team members, and allows the student to move these members around the island. When a team member enters a new location on the map, an image of that location is loaded into the upper-right *workspace* frame of Figure 2. This is an example of a component passing a message to another component. In this case, the map component passes a *loadImage* message to the image-map component in the *workspace* frame. The programmer also created an authoring tool, shown in Figure 3, to allow authors to start with any image and place nodes at various locations to construct a map.

This simple yet powerful navigation component is reused to allow the student to navigate through the pharmaceutical lab in the second part of this exercise. We are currently extending this component to support Quicktime VR images [17]. One can envision other types of navigation components that utilize, for instance, 3-D graphics.

Since the image in the *workspace* frame is displayed through the image-map viewer discussed above, it supports all of the extensions discussed there. For instance, clicking on a hot spot on the image can load a protein viewer showing the molecular structure of a protein relevant to the current exercise.

The upper left panel in Figure 2 contains a profile viewer component that displays information about the currently active team member. This component also has an associated authoring tool that allows an instructor to add his or her own team members.

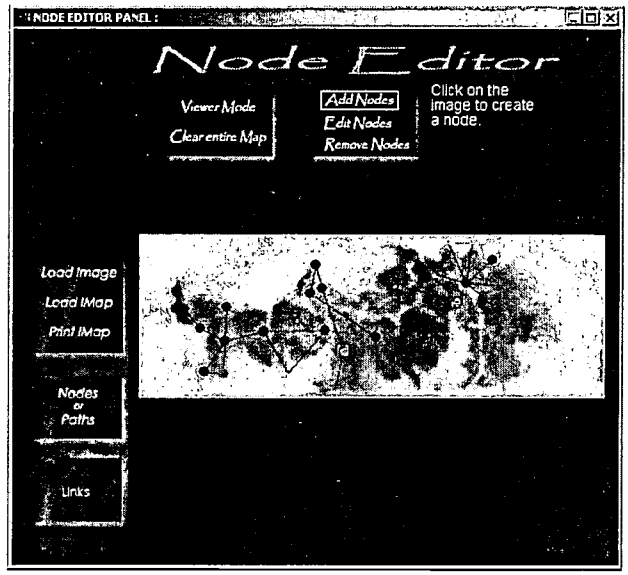

Figure 3. Authoring tool for creating and editing new maps.

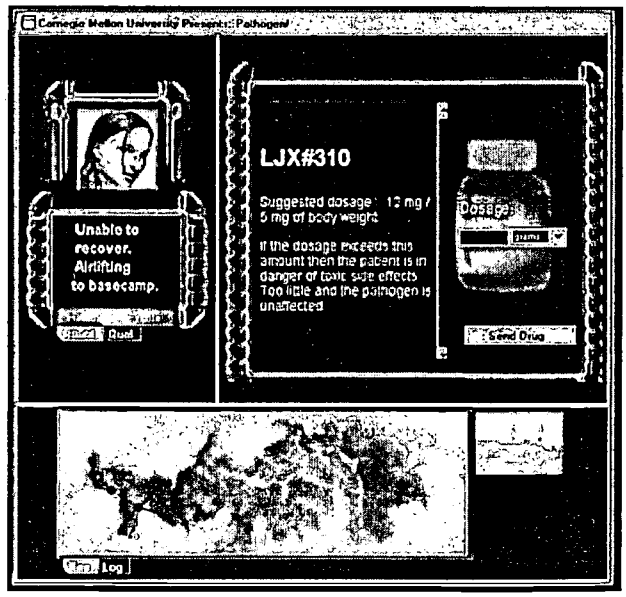

Figure 4. The Pathogen student exercise displaying the written problem (upper right).

BEST COPY AVAILABLE

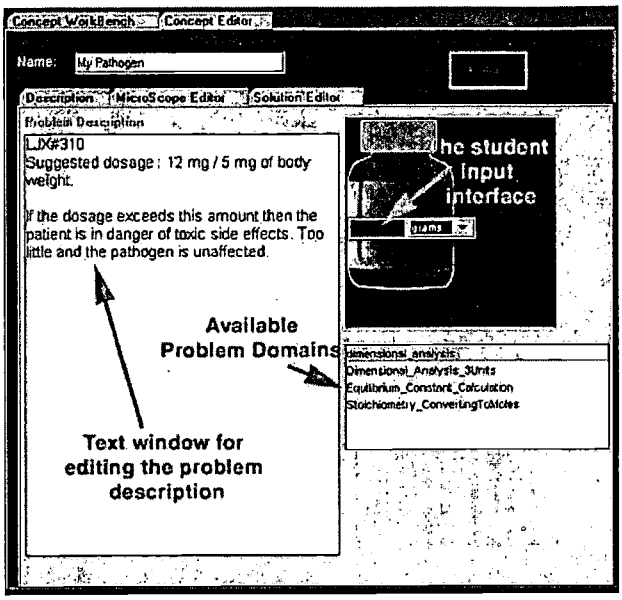

Figure 5. The Pathogen instructor authoring tool showing the creation (or editing) of a specific drug problem.

When a team member becomes infected with a pathogen, a drug-bottle component appears in the *workspace* frame, as shown in Figure 4. The instructions on the drug bottle pose a chemical problem to be solved by the students. Again, an authoring tool, shown in Figure 5, is provided to allow instructors to add their own problems to the application. Note that this problem queries the profile viewer for information such as the body weight of the team member, which may be of relevance to the appropriate cure. This communication between viewers illustrates the use of extended linking to do simple queries, in addition to the message passing discussed above.

Note that in adding a powerful component such as those created for Pathogen, the programmer creates three items: a viewer, a format for the content displayed and/or manipulated by this viewer, and a authoring tool to allow non-programmers to create or modify this content.

Note also that the approach provides multiple entry points for instructors. For instance, an instructor may first simply use the drug authoring tool of Figure 5 to add their own chemistry exercises to Pathogen. Or they may use the profile authoring tool to replace the team members with students in their class. As they become more familiar with the approach, they may attempt more complex modifications and eventually assemble their own exercises.

7. ADVANTAGES

7.1 Instructor Versus Student Interfaces

The above examples illustrate a fundamental shift in the development of interactive software for education, whereby the programmer's primary focus is on designing an "instructor interface" so that the instructor can design a "student interface." Consider the current situation, where a simulation of a combustion engine would typically be constructed as a complete, stand-alone application. The programmer would need to attach a student interface to the simulation, and in so doing, would make curriculum decisions that severely limit potential reuse. If the application were meant to illustrate a college-level concept such as thermodynamic cycles, the resulting user interface would exclude use of the application in high schools to illustrate the ideal gas law. A potential alternative would be to create a simulation with a very flexible interface, but this could easily lead to a complex application that confuses students. With the above assembly environment, the programmer is able to design an "instructor interface," with the goal of providing sufficient flexibility that the curriculum author can design useful "student interfaces." This separation of responsibilities between the programmer and curriculum developer should lead to much more effective collaboration and reuse than is currently possible.

7.2 Browsing through Active Content

In creating a student exercise or tutorial, the curriculum author is essentially creating a means for a student to browse through active content. This browsing ability is supported by the ability to create links that load learning objects into frames. For instance, in creating an exercise about hemoglobin, an author can create two frames, one of which initially contains text describing the role of hemoglobin in the body and the other containing a protein viewer object displaying the 3D molecular structure of hemoglobin. In response to student clicks on text links or images, the instructor can load the virtual lab containing solutions in which hemoglobin has bound up various numbers of oxygen molecules, or an animation showing how hemoglobin sequentially binds up to 4 oxygen atoms.

The curriculum author is thereby allowed to focus on content, independent of the particular software viewer (text viewer, molecular viewer, virtual lab, QuickTime player) needed to display this content. The content is also well extracted from the viewer, in a web with relationships and interconnections that reflect the curriculum content, rather than integrated with the technology needed to display this content.

8. FUTURE DIRECTIONS

8.1 More Flexible Control Panels

The Mars example of Figure 1 showed the use of an editable hot spot to serve as a control to a simulation. This control may be extended to allow sliders, dials etc. to be attached to certain variables of a simulation. This is an area where graphical assembly of components is known to work well. For instance, users with little programming expertise are able to construct simple yet powerful front panels in LabView [16] that control real instruments. Also, attaching text boxes, sliders, etc. to JavaBeans is one area where even simple Bean assembly tools work well. We may adopt this paradigm by allowing users to drag controls on top of an image map to construct a control panel for a simulation. Attaching these controls to a simulation uses the extended linking mechanism; for instance, the author could arrange for a slider to act as a throttle for the engine by first setting the minimum and maximum value, and then linking it to the engine, choosing the "*engine:setspeed:value*" link to the engine object. Such controls are easily built into the hyperlink-like scheme for message passing, since they need to send the message only when altered. This model mimics the construction capabilities of a commercial JavaBeans beanbox assembly environment but delivers it in a non-programming, education-specific service that opens

construction to authors. Simultaneously, it provides a service to which educational programmers can contribute their *industry standard* Java classes as learning objects. Of course, those programmers will need to take into consideration design specifications that emerge from the research being done on object design both by other groups and as a result of our own research.

8.2 Branching

Another issue for continued research is the means by which the environment responds to student input. In the Pathogen application discussed above, the student's response to the chemical problem posed on the drug bottle is checked through a special-purpose mechanism provided by the programmer. We are currently working on branching objects that can provide authors with a simple means to add conditional behavior, or if-then statements, to the environment. These form the basis for individualizing responses and feedback to the choices that a student makes in navigating the learning environment. We view complex branching behaviors as the domain of the programmer, and therefore they are meant to be programmed into learning objects. For instance, a simulation such as our Virtual Lab [20] provides the student with varied choices and feedback on the choices they make and actions they take.

One way to give authors branching capabilities is by allowing branching controls to be inserted into image maps, or control panels. The author could, for example, use a multiple choice panel component to allow a student to follow different links depending on his or her answer. Alternately, an integer response control and real-number response control would accept a value from the student, and follow various links depending on the value. Allowing the input to such controls to come from simulations (via extended linking) leads to a flexible simulation environment. For instance, the author can create text in a text-viewer that asks the student to adjust the air/fuel ratio of the combustion engine to achieve a certain power level. They can then add the text "click here when done", and link the word here to a branching control. The branching control is of the real-number type, linked to *engine:get_power*, and configured to link to *text_frame:load:power_correct.htm* if the value is in the correct range and to *text_frame:load:power_incorrect.htm* if the value is out of the correct range. This model becomes more powerful since multiple links can be attached to the same hypertext or image-map region.

8.3 Assessment Components

Student performance assessment is an essential functionality. Our environment allows assessment/grading components to be easily integrated into exercises with the simulations. The functionality will include the ability for students to login to the assignment, and then the collection of milestone data. The author can create links to the assessment objects as the simulation is created. For instance, in the above pathogen exercise, the link to *text_frame:load:power_incorrect.htm* could be coupled with the link *grading_object:milestone:("power question",incorrect)*, and similarly for the correct response. (This illustrates the importance of multiple links from a single point.) The assessment component would then keep track of the number of attempts before the correct response was obtained. Grading may be viewed as a special case of this assessment model, where the grade is based on achieving certain milestones. We are currently developing methods to store milestone data in a network database, for which we will provide a simple implementation using an open source database such as MySQL. Extensions could include learning objects that interface to standard course management systems such as those offered by Blackboard and WebCT or with assessment tools developed for distance learning [1].

8.4 Interobject Communication

Another possible extension is to add an additional mode of inter-object communication to the viewer. For instance, objects may be allowed to publish or monitor messages on a bus, as in the InfoBus[1] standard. While this would allow programmers to provide authors with more powerful control panel objects than that described above, it is important to assess the effects of this added complexity on the author. In particular, will issues of synchronization lead to new types of errors, beyond the broken links of the current design, and potentially frustrate the curriculum author?

9. DIGITAL LIBRARY SERVICES

We stress that our approach is to start from a simple design and add in only those functionalities that retain a high level of ease-of-use. This assembly environment is not meant to provide a single all-purpose solution to component assembly. Rather it is meant to complement assembly environments currently available that are based on the simplified-programming model such as ESCOT. Design of assembly environments involves decisions of flexibility vs. ease-of-use. The emphasis in this tool is on ease-of-use via an extended-web-authoring approach that should be intuitive to instructors, and on the inclusion of features that help organize the potentially large number of text and images that make up a learning environment. To the extent that this environment does not support a more complex mode of component assembly, one of the tools based on the simplified-programming model may be used to assemble components into an object for inclusion in this environment. Thus the two classes of tools actually complement one another. Our tool fills an important need, since sole reliance on more complex tools would exclude potentially valuable contributors.

Our research then consists of the development of two services in support of digital libraries for education. First is providing an authoring environment that illustrates the "learning object" approach to creating on-line learning activities. This set of core learning objects such as a hypertext viewer and image-map viewer extend their web counterparts by supporting links to active content. Second is exploring the extent to which the piecing together of learning objects can be moved from the domain of the programmer into the domain of the curriculum developer. We anticipate that this shift will lead to both a larger community creating, adapting and using materials for the digital library and a better organized collection.

10. ACKNOWLEDGMENTS

This work was funded by the National Science Foundation as part of the NSDL (National Science Mathematics Engineering and Technology Education Digital Library) program.

[1] InfoBus is a standard for data communication between JavaBeans components.

11. REFERENCES

[1] Project ADEPT - http://www.users.csbsju.edu/~tcreed/adept/index.html

[2] AGENTSHEETS - http://www.agentsheets.com

[3] ARIADNE - Collaborative browsing project on digital libraries. http://www.comp.lancs.ac.uk/computing/research/cseg/projects/ariadne/

[4] American Physics Society - Physics Academic Software. http://webassign.net/pasnew/

[5] Advanced Distributed Learning Intiative network., DOD http://www.adlnet.org/

[6] BELVEDERE http://www.ics.hawaii.edu

[7] EDUCAUSE - http://www.educause.edu/

[8] EOE – http://www.eoe.org

[9] ESCOT - http://www.sri.com/policy/ctl/html/escot.html

[10] ESLATE - http://e-slate.cti.gr/

[11] IEEE Learning Technology Standards Committee (LTSC) http://www.manta.ieee.org/groups/ltsc/

[12] JAVASKETCH - http://www.keypress.com/sketchpad/java_gsp

[13] Meyers, B.J. & Jones, T.B. Promoting Active Learning: Strategies for the College Classroom. Jossey-Bass, San Francisco, 1993.

[14] NEEDS - http://www.needs.org

[15] NSFCCR - http://www.education.siggraph.org/nsfcscr/nsfcscr.home.html

[16] National Instruments Corp., LabVIEW – http://www.ni.com, 11500 N Mopac Expwy, Austin TX, 78759.

[17] Quicktime VR – Apple Computer Inc., http://www.apple.com/quicktime

[18] Smith, J. M. What Steve Jobs Did Right, The Educational Potential of the Ideas Behind NeXTSTEP. Educom Review, 1994.

[19] Squires, D. Educational Software for Constructivist Learning Environments: Subversive Use and Volatile Design. Educational Technology, May-June 1999, 48-54.

[20] Virtual Laboratory, The IrYdium Project, Carnegie Mellon University, Chemistry Dept.

U.S. Department of Education
Office of Educatonal Research and Improvement (OERI)
National Library of Education (NLE)
Educational Resources Information Center (ERIC)

REPRODUCTION RELEASE
(Specific Document)

NOTICE

REPRODUCTION BASIS

☒ This document is covered by a signed "Reproduction Release (Blanket) form (on file within the ERIC system), encompassing all or classes of documents from its source organization and, therefore, does not require a "Specific Document" Release form.

☐ This document is Federally-funded, or carries its own permission to reproduce, or is otherwise in the public domain and, therefore, may be reproduced by ERIC without a signed Reproduction Release form (either "Specific Document" or "Blanket").