Scalable Integration of Educational Software: Exploring The Promise of Component Architectures[1]

Jeremy Roschelle, Jim Kaput and Walter Stroup
University of Massachusetts, Dartmouth

Abstract

Technology-rich learning environments can serve the pressing need for core curriculum reform in science and mathematics by enabling more diverse students to learn more complex concepts at a younger age. Unfortunately, today's technology research and development efforts result not in an richly integrated environment, but rather with a fragmentary collection of incompatible software application islands. In this article we ask: how can the best innovations in technology-rich learning integrate and scale up to the level of major curricular reforms?

A potential solution is component software architecture, which provides open standards that enable plug and play composition of software tools produced by many different projects and vendors. We describe an exploratory effort in which four research groups produced software components for the mathematics of motion. The resulting prototypes support (a) integration of the separately produced tools into the same windows, files, and interfaces, (b) dynamic linking across multiple representations and (c) drag and drop composition of activities without requiring programming. We also summarize an extended Internet discussion which raised critical issues regarding the future of component software architecture in education, and look ahead to the prospect of scalable integration of components beyond the desktop computer.

Introduction

Technology has the potential to enable core curriculum reform in science and mathematics education (SME) by democratizing access to powerful ideas. Powerful software for designing, experimenting,

---

[1] [1] This material is based upon work supported by the National Science Foundation under Grant No. 9619102. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

modeling, simulating, visualizing and communicating can allow students to learn ideas at a younger age and with deeper understanding (Kaput, 1992). Despite the current sense of progress and hope, we are concerned that our community's proposed solutions may not be sufficiently scalable to achieve the systemic level of implementation needed for deep sustained reform in SME.

The paradigmatic research and development project in SME examines short-term conceptual development. Typically, a project investigates a single concept or small cluster of concepts with a uniform age group over the course of days to weeks. The software involved normally consists of a single program, usually developed over months to a year by a handful of programmers. We do not question the value of this paradigm — such design and teaching experiments produce extremely important results — but instead ask how focused, localized, distributed innovations in technology-rich learning might integrate and scale up to the level of major curricular reforms?

A Sample Challenge: The Mathematics of Change & Variation

Some problems cannot be solved by one software program at a time. The long term mission of our SimCalc Project, for instance, is to support the reform of the core mathematics curriculum to democratize access to the MCV. The history of mathematics and science (Bochner, 1966) acknowledges the ubiquity and importance of MCV: rate of change, accumulation, approximation, continuity, and limit (and related concepts) are both necessary and illuminating concepts across a wide range of contexts and processes. And understanding change and variation in their many forms and representations is critical to informing personal as well as political choices in a democratic society. Alas, MCV is neglected, delayed, or denied for all students but the relatively elite few who reach a formal calculus course. And until very recently, even the elite developed more manipulation skill than understanding (Tucker, 1990).

Since learning MCV in all its richness and depth requires years, not weeks or months, a longitudinal approach is required. Thus SimCalc's approach is to work with collaborating partners towards the creation of a elementary through university level curricular strand that revisits and deepens students' understanding of MCV over time while simultaneously contextualizing and organizing important topics such as multiplication, rate, ratio, signed numbers, area, variable, and so on.

Technology will clearly play a major role in making such a strand possible, enabling a diverse population of mainstream students to profit from it. Previous research highlights a range of techniques that lead children to construct MCV concepts at a younger age and with deeper understanding than ordinarily achieved. For example, micro-computer based laboratories (MBL, Mokris & Tinker, 1987; Thornton, 1992) can accelerate students' ability to analyze real motions in mathematical terms. Video analysis tools (e.g. CamMotion, Boyd & Rubin, 1996) bridge between real world experiences and more formal representations of change. Simulations that reify idealized mathematical objects such as vectors can support the growth of intuitive concepts of vector addition (Roschelle & Teasley, 1995), allowing sixth grade students to achieve mastery that surpasses their twelfth grade peers who are enrolled in a standard physics course (White, 1993). [Another approach is offered by physical devices controlled by computers, wherein a student can create, say, a velocity function that controls the motion of an electronically driven "toy car" ([reference? or url?]) In a play on MBL, these are termed "LBM" devices, abbreviating "Line Becomes Motion."] The MathWorlds software that SimCalc is developing [url?] complements these kinds of tools by providing dynamic, editable graphs of piecewise linear functions (Roschelle & Kaput, 1996b).

In the MCV curriculum that we are working to prototype and study, each of the above tools could play an important role in a coordinated whole. For example, video and MBL data collection are useful in different situations: video works in 2 dimensions, whereas MBL more gracefully handles a single, long axis, such as a child walking across a classroom. Simulation and modeling tools can present more idealized versions of these motions in a form that is easier to explore systematically. Whereas data collection tools have superior connection to experienced phenomena, simulations can enhance reflective experimentation. And indeed other tools could prove provocative alongside these: Logo programs, dynamic geometry toolkits, presentation and collaboration tools, notebooks and portfolios, etc. all should play a role in a deep MCV strand (see http://www.ed.gov/Technology/Futures/kaput.html for some scenarios that use these tools).

From the diversity of kinds of functionally required, we conclude that the technology to enable a grade school through university MCV strand cannot be a single computer program. Moreover, no single research team can author the necessary suite of software products as the

amount of funding and the size of the development group required would be on the order of the largest commercial efforts (such as the Microsoft Office suite). Indeed, from our perspective, a centralized development group is not even desirable, as the necessary expertise is not localized. Instead we need a method by which curriculum authors, teachers and others can fluently assemble and integrate complementary functions from a very large set of research and commercial partners.

The Need for and Prospect of Scalable Integration

Software today is locally effective, but globally fragmentary. For example, today it is highly awkward to combine software tools that are each highly valuable in their own niche, and theoretically complementary in ensemble. In science education (and, of course, in science itself), it is natural to want to compare data gathered from the real world (MBL) with theoretically-motivated simulations. Because MBL and simulations are presently authored by different research groups, as independent software applications, with different data formats and screen layouts, such comparisons are nearly impossible. Similarly, in Mathematics it would be nice to use a video analysis tool (such as CamMotion) with simulations written by students, say in Logo. But as yet, there is no way to construct CamMotion in Logo, and no way to bring Logo into CamMotion. A further schism exists between electronic communication tools (such as web browsers) and the whole world of dynamic representations and notations. How can the many innovations of educational research become integrated into a practical suite of tools that support large-scale curricular reform?

In a progressive technological discipline, each innovation and experiment should accumulate not only as so many loose pebbles in a bucket, but also as the integrated structures and systems of a functioning whole. Just as internal combustion engines, electronics, and mechanical linkages all fit together to form an automobile, so should graphs, tables, MBL, video, simulations, notebooks, journals, e-mail and collaboration tools all fit together to form a vehicle for 21st century school mathematics.

Thus, a critical challenge for technology in SME is **scalable integration.** From decades of previous research, much is known about the kinds of tools that can dramatically advance student learning. But little is known about how to integrate these tools so students can have the best of all worlds: MBL, [LBM] and simulations, communication,

video, and dynamic visualization, microworlds and portfolios. Moreover, while technology can be extremely successful in an locally managed setting, not enough is known about how to bring educational technology together to meet the challenges of systemic reform. Small-scale innovations, like a better dynamic graph or simpler spreadsheet, must readily plug and play in larger scale activities, curricula, and assessments. In order to focus energy where it is needed most, on students learning and teaching practice, we need scalable integration of independently produced innovations to become as easy as publishing a web page or producing a newsletter.

Preview

In this article, we report on a collaboration among four projects that explored the potential of component software architecture to meet this challenge. We begin by setting the emergence of component architecture into historical context; arguing that component software is a natural successor to three long-standing lines of research in educational technology. Next we describe how component software architecture overcomes the problem of fragmentary, incompatible software applications by allowing individual software modules to co-exist, smoothly sharing the space inside a window, the storage inside a file, the resources in the processor, and the user interface. The four collaborating research groups utilized Component Integration Lab's OpenDoc™ (http://www.cil.org/) to produce a set of interoperable objects for learning simple MCV concepts. The target of this effort was to demonstrate three critical educational features: (1) live linking across multiple representations, (2) the integration of data gathering and simulation tools, (3) the embedding of microworlds in a variety of containers such as notebooks, word processors, and textbooks. We followed this experiment with an extended Internet electronic-mail-based discussion of the potential advantages and difficulties of component software, which we will summarize for the reader. Finally, we close the article with two generalizations of our notion of scalable integration to include devices other than computers and media other than computer programs.

A Short History of Educational Software Architecture

Software architecture means the design of a framework to enable

diverse functionalities to co-exist and share resources on a computer. The design of a particular learning tool always place inside an architecture, although usually that architecture is taken for granted. For example, most educational software programs use the stand-alone application architecture that is standard on both the MacOS and Windows95. Industry standard architectures have evolved from time-shared mainframes, to client-server workstations, personal computer operating systems (such as Windows and the MacOS), and now possibly towards network computers. Each of these architectures has allowed the development of specific kinds of communities of practice in workplaces and schools.

Our major concern in this article is surpassing the taken-for-granted architecture of the MacOS and Windows95. This architecture confines educational innovations to "application islands." By this term, we mean that software is organized into programs which run independently with their own collection of resources such as windows, menus, and files, and have minimal capabilities to interoperate.

Application island architecture provides weak mechanisms for integrating independent innovations. Consequently, educational software succeeds in small scale design experiments, but fails in large scale systemic reform. The learning technology employed in the R&D centers has not yet achieved widespread availability and use (Chipman, 1993). Examination of recently produced curricular materials and teacher preparation and enhancement activities reveals that technology remains at the margins of most innovation and reform (Bork, 1995). Our nation's best educational technology has not had a strong enough impact on mainstream teaching and learning (OTA, 1988).

In this section we introduce a historical perspective in order to show that application islands are a short-term artifact of the personal computer era, and that educational technologists have long been pursuing better architectural ideas. After reviewing the history of these ideas, we argue that emerging industry-standard component software architectures are well-aligned with research-based principles for educational software architecture. The demonstration project that we later present is thus contextualized both by long-standing principles as well as present problems.

Three lines of prior investigation have developed architectural concepts. First, educational programming languages have allowed educators to compose high-quality learning activities in software

without requiring extensive technical backgrounds. Second, designers of intelligent tutoring systems have emphasized modularity as a means for controlling the expanding complexity of learning technologies. Third, the runaway success of the World Wide Web has brought attention to the value of open standards in catalyzing grassroots authoring. Below, we briefly review these prior efforts, showing that each raises a critical architectural issue, but also has encountered serious obstacles. In the following section, we introduce Component Software Architecture and show that it has the potential to capture the insights of each line of research, while overcoming the obstacles.

Educational programming environments

Educational programming environments have a long history, beginning with Logo and BASIC. Here we start with "Dynabook" (Kay & Goldberg, 1977; Goldberg 1979), which provided the earliest conceptualization of an entire computer architecture, comprising hardware, operating system, and software, all designed specifically to match learners' needs. Kay sketched a portable computer that would allow children to construct, explore, and extend simulated worlds of their own imagination, while learning about school subjects like the dynamics of motion. The Dynabook hardware explicitly addressed children's needs: portability, ruggedness, graphic displays, and low cost. The operating system was open-ended, in-line with the expansive possibilities Kay imagined, but would directly support simulations, drawing, word processing, and communications (including a networked electronic library). The software included a child-centered programming language to enable students to construct software to express their own ideas. Kay sought to design an architecture in line with Papert's (1980) admonition that children should control computers, not be controlled by them.

The cornerstone of Kay's architectural insight was recognizing the need to create a medium in which children could *compose* ideas without extensive technical training. Kay's method for supporting composition centered on Smalltalk, an extensible, object-oriented programming language that was designed specifically for children. As diSessa & Abelson (1985) later argued in the Boxer project, computers were to become "reconstructable computational media" and simplified programming languages would become the basic tool for composition. For this architecture to work, learning to program would have to

become as easy as learning to use a pencil.

Although early Smalltalk experiments generated many local stories of success, Smalltalk and the Dynabook concept gradually parted ways: The Dynabook led to the Xerox Alto & Star systems (Smith, Irby, Kimball, & Verplank, 1982) and later to the Macintosh, and modern graphical user interfaces. Along the way, the concept of a simple programming language as the core architectural building block for all software was dropped. Instead Applications Programmer's Interfaces (APIs) were instituted, enforcing a separation between professional programmers and "the rest of us."

Today, many non-technical people compose ideas on their desktop computers but few use a programming language. Ironically, today Smalltalk is used predominantly by major investment banks, and rarely by children. Other programming languages for children have been invented. Logo (which precedes Smalltalk) is the most famous and long-lived (Papert, 1980). More current examples are KidSim (Smith, Cypher, & Spohrer, 1994) and AgentSheets (Repenning & Sumner, 1995), which provide graphical if-then rules rather than linguistic codes in an effort to make programming more comprehensible to children. Yet while Kay's vision as a medium for composition has prevailed, programming as a ubiquitous skill has not (Nardi, 1993).

On a more technical level, the idea of a programming language as system architecture has proved problematic. Boxer, in particular, has made a virtue out the fact that *every object* in the reconstructable medium must be written in the same programming language (diSessa, 1985). On one hand, this does make it possible for learners to inspect, modify, and extend any object in the system, a principle diSessa calls "naive realism." On the other hand, this requires "detuning" (diSessa, 1985, p. 5) and "shallow structuring" (diSessa, 1985, p. 6) of the programming language, in order to control the complexity presented to the novice. Consequently it is easy to learn Boxer, and simple to build a wide variety of useful education widgets. But it is difficult for Boxer to keep pace with rapidly advancing interactive technology. For example, there is no tool with the depth and power of Geometer's Sketchpad (Jackiw & Finzer, 1993) for dynamic geometry in Boxer, because it would be burdensome to program with Boxer's detuned and shallow structures. As computer systems have evolved, programming languages have become specialized: some languages are better for professional programmers (e.g. Lisp, C++, Java), while others are targeted for educational end-users (e.g. Logo, AppleScript, JavaScript). Speed,

efficient data structures, and low-level communications require one class of programming languages, while ease of use, rapid prototyping, and automation of tasks require another class. Building a Macintosh component like CamMotion in the Boxer programming language would not be feasible; Boxer does not allow the low-level control needed to control QuickTime, for example.

The concept of reconstructable computational medium remains highly compelling — ease of constructing and expressing powerful ideas should be a key criteria for all educational environments. But basing the medium on a single programming language that spans the gamut from operating system to educational scripting has yet to become pragmatic.[2] It appears that no single computer language can be both simple to learn and rich enough to build professional quality dynamic curricula. As we will see later, component software architecture retains the goal of a reconstructable computational medium, but allows for diversity in programming languages.

Intelligent Tutoring Systems

Intelligent Tutoring Systems (ITS) research provides a second line of educational technology with explicit architectural concerns. A classic ITS uses artificial intelligence techniques to formulate a model of a student's knowledge and a model of expert knowledge, and then intervenes with tutorial advice when differences become evident (Wenger, 1987). The earliest ITS projects recognized the complexity of the necessary code, and developed architectures that emphasize modularity. For example, Clancey's (1987) Guidon system explicitly presents a set of components for expert knowledge, student knowledge, interpreting knowledge relative to a task, and rules for tutoring the student.

The ITS emphasis on using modularity to control complexity continues to be a central theme in current ITS research, and is a well-established principle in object-oriented design (Booch, 1994). Yet ITS

---

[2] Java is the most recent incarnation of the language-as-system concept. Yet Java has quickly stratified into HTML, JavaScript, Java, and "native" levels, each which is appropriate for a different range of authoring problems. The long-term success of Java will largely depend on the implementor's ability to support a coherent community with quite varied authoring abilities and needs.

systems have not achieved much re-use, integration, or scalability (Murray, 1996). The critical flaw in ITS architecture is that the inherent modularity of ITS systems is only available to the developers. Once a classical ITS program leaves the shop, it is closed and monolithic. This prevents re-use, extension, or modification of the system except by its original developers.

More recently, ITS researchers are seeking to enable non-programmers to compose educational activities by designing authoring systems (e.g. Munro, 1995). These systems, however, are susceptible to the same critique as the Smalltalk and Boxer: every object must be constructed out of a uniform language. This language may be graphical (e.g. RIDES, http://btl.usc.edu/rides/) or may be a knowledge representation language (Murray, 1996). In either case, a closed, proprietary language becomes the feature-bottleneck that limits the possibilities of the architecture.

An important emerging trend in ITS design leads directly to component software architecture by emphasizing the use of open standards to integrate modules. Standards have been proposed for knowledge representation languages (e.g. KQML, Finin, Fritzon, McKay, & McEntire, 1994). Also Ritter & Koedinger (1995) are developing a technique for building modular tutors that communicate via industry standard scripting languages. As we argue below, combining modularity with open standards can enable scalable integration.

Indexing and Retrieval of Documents

The theme of open standards has been developed by a third line of research, focused on indexing and retrieval of documents. Starting with Bush's "memex" (1945), futurists have imagined an uniform machine for finding documents and speedily displaying them. Nelson's (1987) Xanadu system first proposed open standards for formatting and citing documents. The World Wide Web (WWW), which is driven by the HTTP (file transfer) and HTML (document format) standards, provides the best example of scalable integration to date. The WWW has grown exponentially to millions of servers and billions of documents in just a few short years, while retaining well-integrated modes of navigation, cross-referencing, and display.

The WWW is widely recognized as an important educational resource, but [until very recently suffered] from an important limitation: unlike Kay's Dynabook or ITS tutors, the WWW provides mostly static

information. There are few opportunities for students to compose or construct their own ideas, and the tools and format for doing so (the "web page") are fairly arcane and constrained. In contrast, educational research, particularly in math and science education, emphasizes the need for students to construct and explore dynamic representations, visualizations, simulations, and animations, using appropriate content-specific analysis tools. Web browsers draw a harsh line between composing and reading; the tools for constructing web pages are application islands separate from the tools for using them. The limitations on dynamic content and composition prevent the current WWW from living up to Kay's vision of a child's medium for constructing ideas.

Component Software Architecture

The key principles drawn from the history of educational software architecture (table 1) emphasize a **reconstructable medium** to support composition, **modularity** to control complexity, and **open standards** to scaffold a broad scale implementation. These principles are echoed by analyses of the software industry in general. Morris & Fergusson's (1993) analysis of the high technology industry shows convincingly that architecture is the major factor in long-term, large scale success. They emphasize the need for open standards that allow software to be flexibly adapted and extended to meet diverse needs. Cox (1996) argues that modular architectures are required to encapsulate and coordinate the complexity inherent in modern software systems, and emphasizes the transition of the computer from a technical means of computation to a widespread medium for composing, expressing, and communicating ideas.

Table 1: Principles and Obstacles in Prior Architectures
October 2, 2001
Architectural principle
Architectural obstacle
Dynabook and Boxer
composable medium
uniform programming language
ITS
modularity
closed system

WWW
open standards
sharp distinctions between browsing and composing
12

A software architecture that embodied all three principles would radically increase the potential for meeting the challenge of longitudinal curricular reforms. Research and development products could accumulate in a digital library of useful learning tools. Open standards could supply a unified target "platform" on which all the necessary software capabilities would operate, while not restricting implementations to use any particular language or development method. Modularity could allow the natural boundaries in mathematical objects (e.g. calculators, tables, graphs, simulations, etc.) to be the basis for division of labor among research and development efforts. A composable medium could allow the resulting plethora of powerful tools to be assembled in diverse combinations for particular children, grade levels, curricular goals, etc. The educational community could leap forward towards scalable integration by adopting these principles.

The design experiment described below explored this possibility using Component Integration Lab's OpenDoc (http://www.cil.org/), one of several emerging component software architectures which directly target these principles, while avoiding the obstacles. As we will describe, OpenDoc allows educators to compose objects into a single window, store them in a unified file, and arrange them to support a focused task. It supports simple programming to automate sequences of behaviors ("scripting"), but does not require it. OpenDoc enables separately developed modules to plug and play in a single process. In contrast to monolithic applications, the resulting "compound" windows are open to inclusion of computational objects from multiple developers. Moreover, OpenDoc provides open standards which encourage many projects to coordinate their efforts to achieve common goals.

Two alternative infrastructures for Component Software Architecture (CSA) are Microsoft's ActiveX™ (http://www.microsoft.com/activex/) and Sun's JavaBeans™ , a Java-based component architecture (http://splash.javasoft.com/beans/). All CSAs support similar capabilities such as:

- sharing interface resources such as windows and menus among components

- embedding components inside other components
- storing a layout containing multiple components in a single file
- linking and updating data dynamically among components
- scripting behaviors across components
- interoperating with Internet services

In contrast, most current educational software uses an application island architecture, simply because this has been the only available possibility on Windows and Macintosh computers. CSA shares some attributes with modern applications islands, but also provides some important differences. In both CSA and application islands, a programmer constructs a component as a modular object. A typical component in education, for example, might be a graph, a table, or a calculator. But in contrast to stand-alone applications, CSA produces open rather than closed systems. In open system, new objects can be added to an on-going project without the help of a programmer. Indeed, under CSA, a non-technical person can compose a graph and a calculator from different developers into the same window, by simple drag and drop operations. Under the traditional architecture, the original team of programmers must write, compile, and link code to add software from another development group into their program. CSA permits *assembly* of compound windows from standard parts, whereas older architectures require a programmer to laboriously *fabricate* each new combination with hand-crafted code (Cox, 1996).

Software for SME naturally consists of combinations of modular capabilities. Three kinds of capabilities are critical:

(a) core subject matter components, such graphs, simulations, and data tables.
(b) containers, such as word processors, notebooks, portfolios, and assessments.
(c) collaboration tools, such as e-mail and web browsers.

Ideally, CSA would allow educators and learners to plug and plug a variety of core components into appropriate containers for the activities at hand. The resulting compound windows could be shared objects that support collaboration. Curriculum authors, teachers, or students could easily compose projects from a suite of standard math and science components, favored containers, and Internet-savvy collaboration tools. Below we report on a preliminary initial design experiment that

investigated whether CSA (and OpenDoc in particular) could support integration of contributions from many research projects, and scaling up from focused innovations to broadly useful educational environments.

EduObject Testbed

Beginning in December, 1995, four National Science Foundation (NSF) projects formed the EduObject testbed to explore the potential of CSA. Table 2 describes the participating projects, which represent both commercial and academic institutions, and diverse NSF divisions. The participants primarily collaborated through the Internet, supplemented by a few face to face meetings. The sections below discuss the goals, techniques, and outcomes of the testbed through June 1996.

Table 2: Participants in the EduObject Testbed

Project & Leaders	Institution & Web Site	Funding
SimCalc Jim Kaput Jeremy Roschelle	UMass, Dartmouth http://tango.mth.umassd.edu/	NSF Application of Advanced Technology, RED-9353507
MacMotion Ron Thornton Steve Beardslee	Tufts University http:	NSF ???, #xxx-xxxx
DataSpace Bill Finzer	Key Curriculum Press	NSF Small Business Innovative Research, #xxx-xxxx
CPU (which spells?) Fred Goldberg Arni McKinley	San Diego State University with a subcontract to MetaMind, Inc.	

Goals

This testbed established 3 goals:

1. Integration of independently developed math and science modules such as simulations, MBL data collection, graphs, and tables into unified displays, interfaces, and files.
2. Dynamic linking of data across multiple representations, such that all representations update synchronously.
3. Composition of activities by mixing and matching core subject matter components (as in goal 1), containers (such as page layout or word processing), and collaboration tools (such as e-mail and web browsers).

We established these particular goals because of their relevance to educational software needs, and the difficulty of satisfying these goals in a traditional application island architecture. Integration among software modules, and between software and curriculum is crucial for the wide-scale adoption of technology in schools (Bork, 1995; Goldman, Knudsen, & Muniz, 1995). Dynamic linking across simulations, tables, graphs, and other representations of data has proven to be an important feature in many educational technology designs (Kaput, 1986, 1992; Kozma, Russell, Jones, Marx & Davis, J, 1996). Whereas traditional architectures prohibit dynamic linking between application layers, CSA potentially enables synchronous updating. Finally, each project in our group acknowledged the burden of providing all three areas of needed capabilities (core components, containers, and communications). CSA potentially allows each project to develop a complete activity by composing their own components with those of other projects.

Within these goals, we selected the physics and mathematics of motion as our target curriculum, on pragmatic grounds. Three of the four projects include the physics of mathematics of motion in their core agenda, so this target allowed us to draw upon substantial resources. Moreover, extensive research has been performed on the use of simulations, MBL, multiple representations, and dynamic linking in

teaching about motion. As a consequence, we could focus on the problems of scalable integration with confidence that the underlying learning paradigm is sound. (Indeed, this report does not cover empirical work with students. Each of the collaborators will be producing their own evaluations of learning, according to their own project goals.).

Techniques: Linking Multiple Representations

Each project in this collaborative had already selected OpenDoc for its own reasons, making OpenDoc the most convenient platform for the testbed. Although, both OpenDoc and ActiveX now run on both Macintosh and Windows computers, we focused on Macintosh PowerPC implementations. Nonetheless, lessons learned from our experience should apply equally well to ActiveX, or JavaBeans, and to Windows or other operating systems, as we did not exploit any features unique to one platform. [Is this true of drag & drop?]

Component Integration Labs calls components produced to the OpenDoc specification "LiveObjects™ ." Using the OpenDoc platform, we decided to develop a variety of LiveObjects that would be useful for learning about velocity and acceleration. As Table 3 illustrates, each project agreed to produce some of the necessary components. In addition, third party commercial vendors produced other LiveObjects that we were able to use.

Table 3: Kinds of Components Produced By Each Partner

Content
Containers
Collaboration
SimCalc
simulation
table
graph
equation editor
digital meter
3D visualization

MacMotion
MBL data collection

vector visualizer

CPU
vector meter
simulation
Dock 'Em™

Dataspace
graph
database

Apple Computer or Shareware
voice recorder
drawing tools
QuickTime movies
ClarisWorks™
Word Processor
Outliner
E-mail
Newsreader
Web browser
17

OpenDoc provides most of the infrastructure required to mix and match the components listed in Table 3 and thus achieve Goals 1 and 3 (listed above) Our primary challenge was live updating of multiple representations specified in Goal 2. In particular, we required that the graphs, visualizations, and tables produced by different projects should all update simultaneously. Moreover, we wanted the same graphs, tables and meters to display data regardless of its source (simulation, MBL, or database).

To achieve live updating of multiple representations, we created an open standard called EduObject. This standard specifies three things:

1. an interface to a shared object that represents a particle (position, velocity and acceleration vectors)
2. a change notification protocol (for synchronizing updates)
3. persistent storage routines (for maintaining cross-component links when saving, closing, and opening windows)

The standard is specified in CORBA IDL (Orfali, Harkey, & Edwards, 1996) a platform-independent, open standard for describing the interface to a shared object. It is implemented as a shared code library that implements a set of CORBA object classes. Each LiveObject viewer must follow the EduObject standard. In practice, this means the four projects agreed upon the EduObject shared library, and linked their specific components (simulations, graphs, tables, meters, etc.) to it. We negotiated this standard by e-mail discussion, and shipped the shared library to each site via the Internet. Because the shared library uses IBM's System Object Model, we avoided the classic "fragile base class problem" in which changes to a shared object require every developer to produce a new version of their module (Orfali, et al., 1996). We were able to revise the EduObject standard and the library many times without breaking any of the existing software modules.

The standard follows the classical model-view-controller architecture for synchronizing views of shared data (e.g. Krasner & Pope, 1988). In our testbed, the model was a collection of particles each with its own position, velocity, and acceleration. Various other components, such as graphs and tables, display views of the model. Each view registers its interest in the model by creating and registering an object that listens for changes. As the simulation updates the model, it notifies each listening view of the changes. A controller is a user interface component that can change the model (and hence the views). We designed controllers driven by the mouse, or by real-time MBL data collection. Figure 1 is a schematic diagram of how a simulation, graph, and meter component cooperate to synchronize multiple views of data collected from an MBL device. Importantly, each display view is an separate component. These views can be authored by independent programming teams, and yet they integrate immediately upon being dropped into the same window.

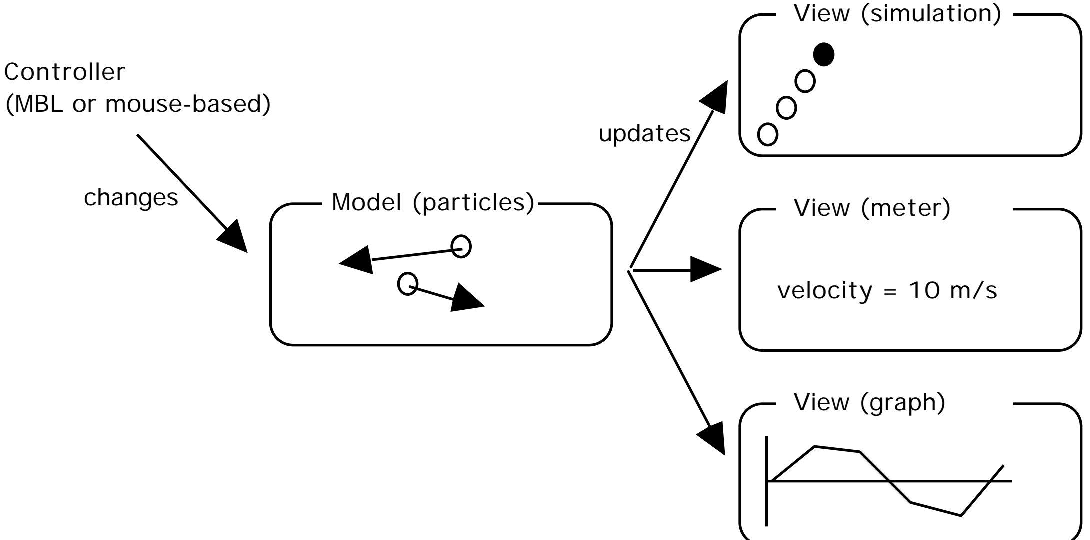

Figure 1: Linking multiple representations ala Model-View-Controller

Full technical details are described elsewhere (Roschelle, 1996) and sample code is available upon request.

Outcomes

We started in December, 1995. By early April, 1996, with only part-time effort on behalf of each group, we were able to demonstrate educational activities that were composed by mixing and matching LiveObjects from each of the four collaborating projects. Rarely do software components from different research projects work together at all. Yet this testbed was able to demonstrate sophisticated interoperability in a relatively short time. Below we discuss the range of features this testbed demonstrates.

Figure 2 shows a sample activity in which a student explores a race between an accelerating particle and a constant acceleration particle (above the graph). The outer container for this activity is Dock 'Em, which supports composing activities as a stack of pages and also supports typing text and drawing shapes. Within Dock 'Em, we have placed a simulation and a position graph, as well as a table of data. When the simulation runs, the graph (color-coded to match the colors of the particles) draws its plot from left to right, and the table highlights successive rows. The activity also includes a voice recorder component, which the student can use to record her thoughts. This activity consists

of 5 separate software modules, written by several different authors, which nonetheless behave as one integrated activity. Moreover, by clicking on the arrow in Dock 'Em, the student can flip to the next activity in the sequence, which can use a different mixture of components, as appropriate.

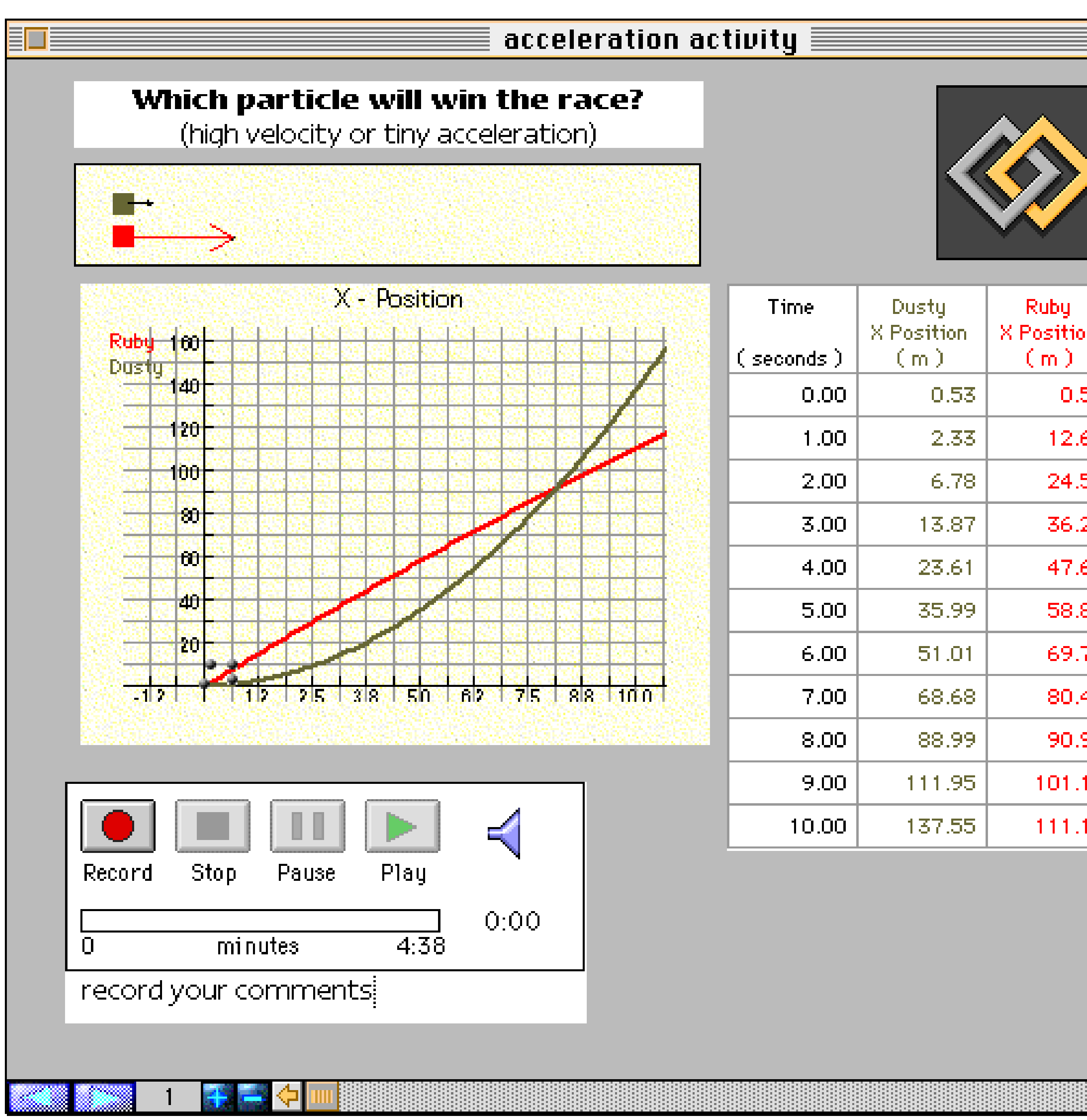

Figure 2: An activity composed of many independently developed components

The components developed by the testbed all interoperate and share data. To establish a link between multiple representations, a student or teacher drags a particle to the desired graph, table, or meter. Immediately the view will begin to update as the particle changes. The same views work with data sources that can be in a simulation, MBL data collection, or database.

Moreover, the content components can be embedded in a variety of containers besides Dock 'Em. For example, a student can keep a notebook by dragging and dropping components into an OpenDoc-savvy word processor. Many students and teachers currently use ClarisWorks, an integrated “office” package that combines word processing, drawing, spreadsheets, and other features. In an experimental pre-release version of ClarisWorks, a teacher or student can drag any of our components (a running simulation or vector visualizer) into their own document. (Claris has announced that this capability will be come standard in the next release.) Other containing formats such as draw programs, outliners, and stacks are available from other vendors.

Additional content can be added to any testbed activity in a variety of media types such as movies, sounds, pictures, 3D renderings, or virtual reality scenes. These content types are supported by LiveObjects provided by Apple. Similarly, Apple’s Cyberdog suite allows e-mail, web browsers, file transfer, and newsgroup readers to be added to any activity window (Figure 3). We have explored many of these potential combinations in the testbed, and all work smoothly with our components. The testbed has met our three joint goals of (a) integration, (b) dynamic linking and (c) layering of containers, content, and communications.

In separate work, we have explored the use of scripting languages to support user programming (Roschelle, DeLaura and Kaput, 1996). Scripting languages can enable many of the desirable features of Dynabook and Boxer without requiring ubiquitous programming skills or a singular programming language. Ritter & Koedinger (1995), for example, have developed a modular ITS agent that interacts with our MathWorlds software via a scripting language [url?]. Koutlis (1996) has enabled Logo to be used within OpenDoc, providing teachers and students with a familiar programming language for controlling computational objects.

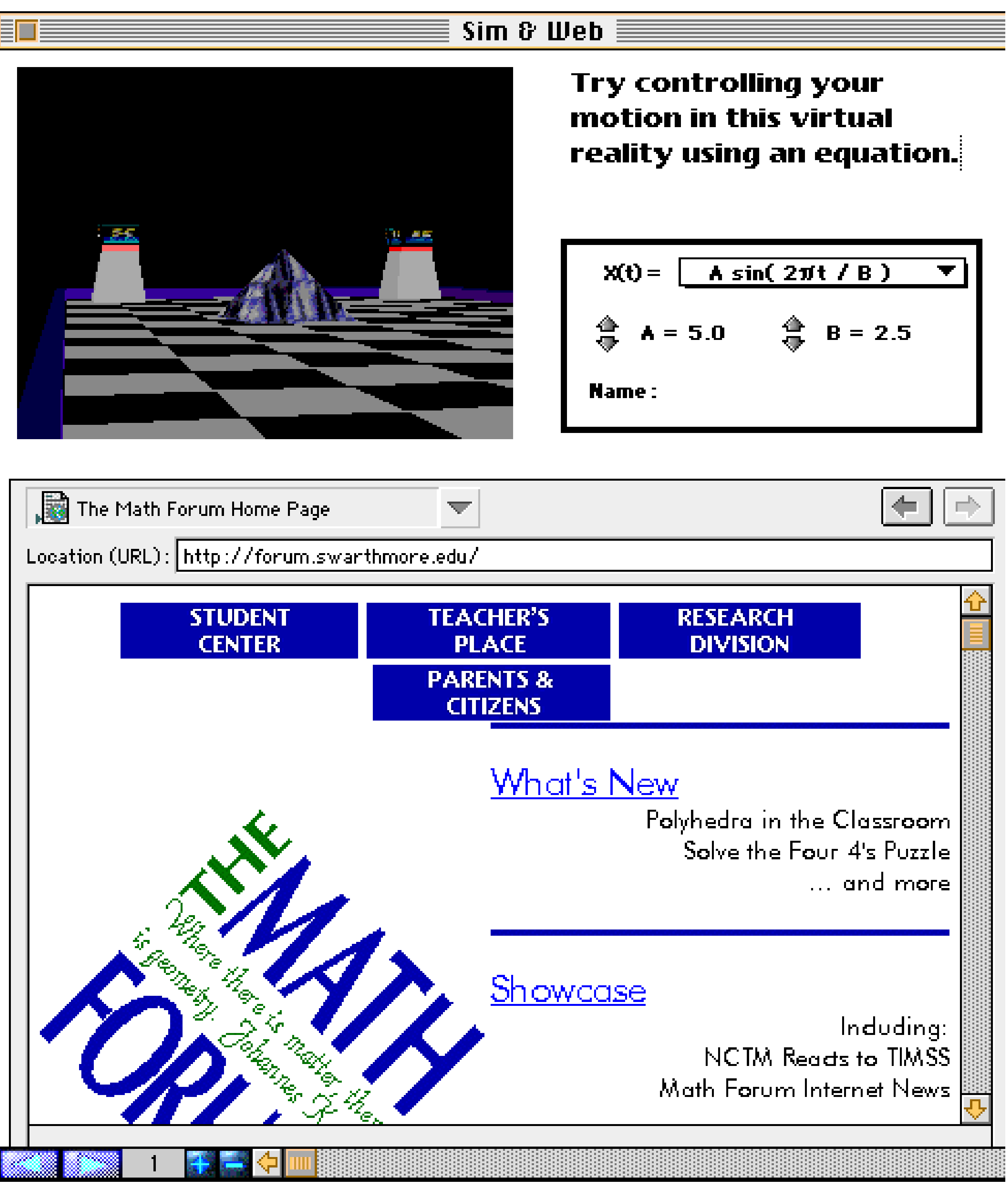

Figure 3: A simulation and a web browser combined in the same window

Our experiences with EduObject suggest that emerging CSA platforms such as OpenDoc, ActiveX, and JavaBeans, will offer important opportunities for educational projects to achieve scalable integration. Using CSA, we were able to readily integrate kinds of software that normally exist in distinct application islands. For example, our prototypes can integrate simulations and MBL, exploratory tools and multimedia, as well as dynamically editable graphs and communications tool. We also believe that CSA provides much stronger support for scaling up. Under traditional application island architecture, larger scale would require a programmer to compile, link, and build a monolithic program that grows larger and more complex with each new feature. In our testbed, however, programmers each worked on a limited component that encapsulated one level of complexity. At present, non-programmers in several of the cooperating projects are developing OpenDoc-based activities for students to use, and will be evaluating the success of those activities in schools.

Assessing the Potential and Pitfalls

Following the successful design experiment discussed above, we initiated an Internet-based electronic mail discussion group to evaluate further the potential of CSA to solve the problem of scalable integration in educational technology. Information on how to join the list and a complete set of discussion archives are currently available at http://forum.swarthmore.edu/edcomponents. This discussion list was widely announced and drew approximately one hundred participants during the 3 months of peak activity. Interestingly, the discussion spanned groups that traditionally have been mutually exclusive: representatives of the Intelligent Tutoring System community and Mathematics Education researchers, and both academic projects and commercial publishers. CSA is clearly an matter that interests a broad range of educational technologists.

During the discussions, three general issues were raised. Each presents a potential challenge to be met in order for CSA to be productive in educational research and development.

1. **Project Management and Intellectual Property.** Component software implies greater cross-project dependencies. Hence researchers are concerned about how they will cross-license

software, and manage schedules that require delivery of components from other projects. When larger scale products (such as a curriculum) are assembled, a system is required to ensure that each contributor gets appropriate financial and intellectual credit.

2. **Data-linking Standards**: In order to build software that dynamically links multiple representations, the community needs standards for linking data throughout components such as simulations, graphs, tables, and equations. At the moment, there is no process in place for arriving at technical standards within the educational community, although there are ample models in industry, government, and society-based standards groups.

3. **Appropriate Component Platform Choices:** Several component-like foundations are emerging from industry (including VisualBasic, Java, OpenDoc/LiveObjects, OLE/ActiveX, and Netscape plug-ins). Each serves some purposes better than others, and researchers and developers need a pragmatic strategy for the short-term that will enable creation of needed knowledge, without engaging in short-sighted platform wars or investing in development inappropriate to their real needs.

In order to pursue long term productive research within CSA, the educational technology community will need to come to a better understanding of these issues. We have little doubt that each of these issues can be resolved. The major obstacles are not technical, but social, economic and political: CSA implies not only a technical shift, but also a change in working relationships in the educational technology community. In place of centrally-controlled, autonomous research projects, there needs to be a shift to greater collaboration and mutual dependency, additional efforts to attain open standards, and more planning of how interoperability will be achieved. This shift may be difficult, but it appears to be the only route to building large scale solutions adequate to match the scale of the problems of SME reform.

Extensions: Scalable Integration Beyond Computers

So far, we have been considering the problems of integration among educational software tools running on a single desktop computer. Educational technology, however, is not limited to computers. A device of particular interest in mathematics and science is the graphing calculator. Graphing calculators have become nearly ubiquitous in high school and university math, science and engineering courses. They are also making significant inroads into mathematics education at the middle and even late elementary school level.

Graphing calculators have many advantages over desktop computers. Calculators are inexpensive and thus can be personally owned by most students. They are portable, and thus travel from mathematics class to science class, and to the students' home. Many calculators are now user programmable, and thus can run software beyond the burned-in mathematics calculations. Indeed, in the not-so-distant future, hand-held devices will be able to run programs written in the Java programming language, and thus calculators and computers will be able to exchange small Java programs [and their data].The idea of scalable integration thus extends to cover interoperability across different kinds of hardware.

Assuming the development and implementation of sufficiently robust local network protocols, we expect that before too long each student's calculator will become a fully accessible object to a more powerful computer acting as the server for a classroom of networked calculators. Students will download projects and assignments from the teacher's workstation to their calculators, perhaps by an infrared beam. Later they can upload results, examples and ideas to such a workstation, where the workstation would have a display projector so that the whole class could see ideas from any source. Using such a mix of technologies, teacher and students could easily compare results emerging from individual student's calculators as well as aggregate results (say of simple simulations or calculations) across students. The workstation could also support assessment tools such as a structured portfolio that would maintain a long term record of students' work. This kind of communication and computational interoperability would allow for close teacher and peer engagement with the activities of individual learners. Based on preliminary work in sixth through tenth grade math classes in Boston, we have reason to believe that this "one computer, many calculator" model of classroom integration can result in

significant learning gains related to the national standards in mathematics and science. [do we want to introduce the "HubCalc" term here - with a suitable copyright notice?]

Scalable integration also has a place in the production of curriculum. As we argued above, software products produced by research groups are usually at a smaller scale than an entire school year or university course. The same problem pertains to activities and teaching units developed by research groups, which usually cover less than a whole course. The fragmentation of the best curricular ideas presents the same problems for schools and teachers as the fragmentation of software.

One potential route towards scalable integration of distributed curricular innovations is to move towards a an "interoperable component curriculum." An interoperable component curriculum could be formed by threading together a set of units chosen from a large repository produced by independent authors adhering to agreed-upon standards. Schools and teachers could potentially select individual units to meet site-specific needs. Technology could scaffold such a system by providing the requisite distribution networks, search engines, and curriculum assembly tools. Such standards, architectures and mechanisms offer the potential to change significantly the incentive structures that currently limit participation in the construction of educational materials and likewise limit the affordable risk and hence levels of innovation (Kaput & Roschelle, 1995) [I laid out some of this in the Ed futures paper.]. Significant research and development effort will be needed to understand the changes in teaching practice required to take advantage of the possibilities of dynamic, modular curriculum.

Conclusions

Two decades of educational technology research and design has produced a large collection of prototype tools which have a proven ability to improve mathematics and science education. Yet, sadly, these programs currently exist as fragmentary application islands, each which holds only a piece of the solution. For example, in the area of the mathematics of change and variation, research has shown the power of a cluster of tools to significantly enhance student learning. And yet, due to lack of a mechanism for scalable integration, this potential impact is lost.

Emerging software infrastructures such as OpenDoc, ActiveX, and JavaBeans offer a platform that contains the needed mechanism in the form of component software architecture. CSA is consonant with the

principles of reconstructable media, modularity, and open standards. And these principles have an extended history in research on educational programming environments, intelligent tutoring systems, and digital libraries. Component software architecture thus has the potential to deliver on the promise of scalable integration of educational software: integration because CSA supports plug and play composition, and scale because CSA supports additive, layered composition of complementary components.

Consider the potential benefits of taking the challenge of scalable integration seriously with a project such as SimCalc. SimCalc's mission is broad: to understand how educational technology can democratize access to the mathematics of change. In our role as developers, we cannot afford to build from scratch high quality versions of each component we need. In fact, some components, such as a computer algebra, are so expensive to build that we cannot afford to build them at all. And yet in our role as researchers, we need an ability to compose different combinations of dynamic representational tools, to see which combinations and sequences help students learn. CSA could allow our development efforts to focus on narrow niches where we can make a unique contribution (such as piecewise linear functions) while allow our research efforts to draw upon a much wider collection of standard mathematical components.

CSA also affects our funders, the National Science Foundation, who would like an interoperable collection of technologies to aid in systemic reform, along with the research to guide their effective use. Instead of inefficiently funding replicated efforts to build many independent application islands that each only cover part of science and math education, the NSF could aggregate its production of innovative components across projects, resulting in considerable efficiencies and a much more useful end product. Publishers could re-use this collection of MCV components with different authors and produce flexible electronic curriculum targeted at different state frameworks and local needs. Importantly, the collection would never be closed to innovation — a new and improved graph, for example, could always be substituted for an older model without needing to rebuild the entire collection. Moreover, commercial and research-based innovations could easily be combined.

Our experiments with CSA suggest that the technical infrastructure for achieving scalable integration is largely available. Still, significant research and innovation will be needed to realize educational visions on

top of industry standard architectures. We know little, for example, about the appropriate granularity of educational objects, or what teachers will need by way of support and training to use them effectively in real classrooms. Many combinations of components will suddenly become possible, and educators will need to know which combinations and sequence work best. The potential of a suite of modular educational tools needs to be properly articulated with standards-based curricula and modernized assessment practices.

Yet the biggest obstacles to achieving scalable integration may not be technical. CSA will require a community of practice in educational technology R&D that emphasizes cooperation and coordination, in place of independent, autonomous research activity. Supporting authoring will have to become a major concern throughout the R&D enterprise, rather than something that waits until an innovation is licensed to a publisher. Funders will likewise need to put a premium on interoperativity and sharability of products that they support the development of. The scope of research will have to include not just short term, localized learning experiences, but also the articulation between these and larger systemic effects. Indeed, the nature of the learning sciences enterprise might need to grow to include a higher objective: to understand the relationship between technological affordances and the practices that enable a community to provide high performance learning experiences for all its members.

Acknowledgments

We first thank our collaborators: Steve Beardslee, Bill Finzer, Fred Goldberg, Ken Koedinger, Arni McKinley, Steve Ritter , and Ron Thornton. We're also grateful to the SimCalc "swamp" team, Rich deLaura, Jim Correia, and James Burke, for their efforts. The work reported in this article was supported by the National Science Foundation (Award: RED-9353507). The opinions presented are the authors, and may not reflect those of the funding agency.

References

Bochner, S. (1966). *The role of mathematics in the rise of science.* Princeton, NJ: Princeton University Press.

Booch, G. (1994). Object-oriented analysis and design with applications. Redwood City, CA: Benjamin-Cummings.

Bork, A. (1995). Why has the computer failed in schools and universities? *Journal of Science Education and Technology, 2(4),* 97-102.

Boyd, A. & Rubin, A. (1996). Interactive video: A bridge between motion and math. International journal of computers for mathematical learning, 1(1), 57-93.

Bush, V. (1945). As we may think. Life, September 10, 112-124.

Clancey, W.J. (1987). Knowledge-based tutoring. Cambridge, MA: MIT Press.

Cox, B. (1996). Superdistribution. Objects as property on the electronic frontier. New York: Addison-Wesley.

diSessa, A.A. (1985). A principled design for an integrated computational environment. Human-computer interaction, 1, 1-47.

diSessa, A.A. & Abelson, H. (1986). Boxer: A reconstructible computational medium. Communications of the ACM, 29(9), 859-868.

Finin, T., McKay, D., Fritzson, R., & McEntire, R. (1994). KQML: An information and knowledge exchange protocol. In K. Fuchi & T. Yokoi (Eds.), Knowledge Building and Knowledge Sharing, Amsterdam: IOS Press.

Finnin, T., Fritzon, R., McKay, D., & McEntire, R. (1994). KQML as Agent communication language. In the Proceedings of the Third International Conference on Information and Knowledge Management (CIKM '94). ACM Press.

Goldberg, A. (1979). Educational uses of a dynabook. Computers and education, 3, 247-266.

Goldman, S., Knudsen, J., & Muniz, R. (1995). *When Promise Outweighs Problems: Technology Integration in Math Classrooms.* http://www.ndec.sesp.nwu.edu/ndec/ProjectPages/mmap/mmap.html

Kay, A. and Goldberg, A. (1977). Personal dynamic media. IEEE Computer, 10, 3, 31-41.

Kaput, J. (1986) Information technology and mathematics: Opening new representational windows. *The Journal of Mathematical Behavior 5*:2, 187-207.

Kaput, J. (1992). Technology and mathematics education. In D. Grouws (Ed.) A handbook of research on mathematics teaching and learning. NY: MacMillan.

Kaput, J. & Roschelle, J. (1995) Connecting the connectivity and the components revolution to deep curriculum reform. In G. Solomon

(Eds.) *Technology Futures.* Dept. of Education Office of Research Publications Electronically Published Monograph, 1995(url: http://www.ed.gov/Technology/Futures).
Koutlis, M. (1996). Personal conversation.
Kozma, R., Russell, J., Jones, T., Marx, N., & Davis, J. (1996). The use of multiple, linked representations to facilitate science understanding. In S. Vosniadou, E. De Corte, R. Glaser, & H. Mandl (eds.), International perspectives on the design of technology-supported learning environments. Mahwah, NJ: Erlbaum. 41-60.
Krasner, G.E. & Pope, S.T. (1988.) A cookbook for using the Model-View-Controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, 1(3), 26-49.
Mokris, J.R. & Tinker, R.F. (1987). The impact of microcomputer-based labs on children's ability to interpret graphs. *Journal of research in science teaching, 24(4),* 369-383.
Morris, C.R. & Ferguson, C.H. (1993). How architecture wins technology wars. *Harvard Business Review,* 86-96.
Munro, A. (1995) Authoring interactive graphic models. In T. de Jong, D.M. Towne, & H. Spada (Eds.), The use of computer models for explication, analysis, and experiential learning, New York: Springer-Verlag.
Murray, T. (1996). From story boards to knowledge bases: the first paradigm shift in making CAI "intelligent." In Proceedings of Ed-Media 96 - World Conference on Educational Multimedia and Hypermedia, American Association of Computers in Education. Charlottesville, VA: AACE.
Nardi, B. (1993). A small matter of programming. Cambridge, MA: MIT Press.
Nelson, T. (1987). Computer lib / Dream machines. Redmond, WA: Microsoft Press.
Nierstrasz, O., Gibbs, S., & Tsichnitzis, D. (1992). Component-oriented software development, *Communications of the ACM, 9,* 160-165.
Orfali, R., Harkey, D., & Edwards, J. (1996). The Essential Distributed Objects Survival Guide. New York: John Wiley and Sons.
Papert, S. (1980). *Mindstorms*. New York, NY: Basic Books.
Repenning, A. & Sumner, T. (1995). AgentSheets: a medium for creating domain-oriented visual languages. IEEE Computer, 28, 17-25.
Ritter, S. & Koedinger, K. R. (1995). Towards lightweight tutoring agents. Proceedings of the 7th World Conference on Artificial Intelligence in Education, August 16-19, 1995, Washington, DC, USA. AACE:

Charlottesville, VA.
Roschelle, J. (1996). Updating parts dynamically with SOM. *MacTech Magazine, 12(8),* 73-89.
Roschelle, J. & Kaput, J. (1996a). Educational software architecture and systemic impact: The promise of component software. Journal of Educational Computing Research, *14(3),* 217-228.
Roschelle, J. & Kaput, J. (1996b). SimCalc MathWorlds for the Mathematics of Change. *Communications of the ACM*, 39 (8), 97-99.
Roschelle, J., Kaput, J, & deLaura, R. (1996). Scriptable applications: Implementing open architectures in learning technology. In P. Carlson & F. Makedon (Eds.), Proceedings of Ed-Media 96 - World Conference on Educational Multimedia and Hypermedia, American Association of Computers. Charlottesville, VA: AACE. 599-604.
Roschelle, J. & Teasley, S.D. (1995). Construction of shared knowledge in collaborative problem solving. In C. O'Malley (Ed.), *Computer-supported collaborative learning.* New York: Springer-Verlag.
Schrage, M. (1993). The culture(s) of prototyping. *Design Management Journal, 4(1),* 55-65.
Smith, D.C., Cypher, A. & Spohrer, J. (1994). KidSim: Programming agents without a programming language. Communications of the ACM, 37(7), 55-67.
Smith, D.C., Irby, C., Kimball, R, & Verplank, B. (1982). Designing the Star user interface. Byte, April, 242-282.
Thornton, R. (1992). Enhancing and evaluating students' learning of motion concepts. In A. Tiberghien & H. Mandl (Eds.), Physics and learning environments [NATO Science Series]. New York: Springer-Verlag.
Tucker, T. (1990). *Priming the calculus pump: Innovations and resources*. Washington, DC: MAA.
Wenger, E. (1987). Artificial intelligence and tutoring systems. Los Altos, CA: Morgan Kaufman.
White, B. (1993). Thinkertools: Casual models, conceptual change, and science education. *Cognition and Instruction, 10*, 1-100.