June 5, 2003

"Design Choices in Deployment of Interactive Simulations via the Internet"

prepared for the

NSF/SRC Engineering Research Center for Environmentally Benign Semiconductor Manufacturing

by

Richard K. Herz
Chemical Engineering Program, and
Department of Mechanical and Aerospace Engineering
University of California, San Diego
email: herz@ucsd.edu

web site for ERC prototype: https://github.com/RichardHerz

Abstract

Simulation software provides a safe and cost-effective way to allow students to learn about systems in which processes interact in complex ways. Capabilities added by integration of Internet communications into simulation software are discussed. Options for the development and deployment of interactive simulations via the Internet are compared. A prototype that implements one of the four major options is introduced.

Introduction

Systems that involve several interacting processes often exhibit complex behavior. Gaining an understanding of such a system can be difficult for a student, especially when only traditional explanatory media such as lectures and text with static graphics are available. The addition of "hands on" experiences of manipulating and using the system aids understanding greatly. Providing such experience with actual systems can be expensive and hazardous in many cases, and even impossible with systems such as molecular, geophysical and astrophysical systems. Simulations of systems with computer software programs provide an affordable and safe way to provide "hands on" learning experiences.

We are primarily interested in interactive simulations in which the student can change inputs and make operating changes, rather than simply viewing demonstrations such as movies or "slide" presentations where the user can only affect the path through the slides. Interactive simulations provide responses that differ according to choices of user. For example, user changes input value in a simulation and output of simulation changes. Another case would be where user enters a response to a question and program responds with evaluation of input. Preferably, the software can provide animation of its display elements rather than simply update of displays.

The terms "learning" and "educational," as used here, are meant to apply to researchers who are trying to gain previously unknown understanding as well as novices who are being introduced to a system. "Student" here refers to any user of the program.

This paper reviews the options for deploying simulations via the Internet. We discuss simulations that rely on communication over the Internet for a significant part of their functionality. Simulations that do not use the Internet during operation and merely use the Internet as a means of distribution from supplier to user are not considered here.

After listing the capabilities provided by communication over the Internet, we discuss the two main choices that have to be made,

- Do the main simulation computations on the student's workstation vs. on the network server.
- Use a web browser to access simulations vs. a dedicated program.

These two sets of two choices thereby produce four major combinations that need to be considered.

Finally, a prototype software project is introduced which partially implements one of these configurations.

Internet communication

The term "client" refers to the computer that an individual student is using. The term "server" refers to a computer that responds to requests submitted by a client. Servers are usually machines that are dedicated to providing Internet communication with a large number of clients. In the case of two personal computers communicating "directly" with each other over the Internet in a "peer-to-peer" connection, the server is the computer that is responding to the request in a given transaction.

Use of the Internet by a simulation can provide one or more of the following functionalities:

- Accessing information (programs and data) that change with time, especially at frequent and unpredictable intervals: new versions of client program components (not yearly program updates), revisions or additions to database records, information on external web sites.
- Accessing information that cannot be downloaded in its entirety to client such as large or proprietary databases, or where a fee is charged per record accessed, copyrighted information at external sites.
- Accessing computation or other information processing on server that cannot or is not desired to be provided on client: proprietary or licensing reasons, computation program only runs on one type of platform, computation has unusual requirements (speed, memory).
- Communicating with other people: email, bulletin boards, chat, receiving assignments, submitting results.
- Communicating between the program running on different clients, e.g., distributed simulations.
- Logging on the server of the history of how clients use the program.
- Posting of software updates and new modules at one server and having them automatically distributed to all clients.

Client- and server-side computation

Simulations usually involve significant numerical computations. These computations can be done on the client and they can be done on the server. In most cases, for a given simulation, the computations will be done either on the client or on the server. However, there is no reason other than complexity that the computations can't be split between the client and the server. In some cases with large scale simulations, the computational duty for one simulation may be divided among more than one server. Here we focus on simulations in which the main computations are done on either the client or the server.

Client-side computation

When simulation computations are done on the client, the process starts with the server delivering to the client a module of program code which is executed, or caused to be executed, by the main program active on the client. This code module then processes input information and displays the results.

Server-side computation

When simulation computations are done on the server, the student enters input information on a "form" in the client program window that contains various input "controls" such as text fields, menus and push buttons. Then the client sends the input information to the server with a request that the simulation calculations be performed. A Common Gateway Interface (CGI) program on the server receives the input information and directs it to another program on the server or another computer to be processed. Finally, the results of the simulation are sent back to the client for display.

Comparison

Client-side computation reduces the resources that have to be provided at the server, reduces delays caused by several clients simultaneously accessing the simulation, and also reduces time delays due to communication of input and results between the client and server over the network.

Client-side computation can also allow a simulation to be used at times and places when an Internet connection is not available.

Client-side computation is not possible in cases where the computation program can only run on the server. Reasons for this include:

- High cost of licensing the computation program on clients.
- Developer only has a compiler for the OS/hardware platform of the server (or a separate computer accessed by the server's CGI program) but wants clients on other platforms to run the simulation.
- Special hardware requirements such as very large memory or parallel processors not available on clients.
- Special case of results at server end actually being produced by a real physical process and not a software simulation (Elgamal, 2003; Henry, 2002).

Web browser as client program

The main program active on the client can either be a web browser or a dedicated simulation program.

Most computers are sold with a web browser program included. This has the apparent advantage that a developer has to write less code than for dedicated simulation program and doesn't have to ask the student or lab technicians to download and install it. In reality, the developer can't assume that all clients will have the most recent version of web browser, or browser plug-ins, required to run their simulation. Therefore, some users will have to acquire another browser or download and install required versions of plug-ins. Therefore, this apparent advantage of browser-based simulations may only apply to simulations which do solely server-side computation or use JavaScript code embedded in the web page source for client-side computation.

Another apparent advantage of a browser is that all students know how to perform at least the basic operations in a web browser and don't require instruction in the use of the browser. The simulations, however, will have their own set of controls that the student will need to learn. The combination of all of the controls of the browser, many of which are not required to use the simulation, and the simulation's own controls present a complex interface to the student.

An additional complication of the simulation + browser interface is that the graphical user interface (GUI) controls that appear in a simulation page often will have a different appearance than the same controls (fields, buttons) of the web browser. Such will be the case with Java applets using the Java Swing interface controls and Flash applets. Therefore, this second apparent advantage of a browser will only be obtained for the most simple simulation interfaces. The advantage of interfaces that are easier to learn is held by dedicated simulation programs in all other cases.

A significant consideration which can be counted as a disadvantage of using a browser is that the client must maintain a connection to the Internet at all times. Until high speed Internet connections become ubiquitous this will prevent many students from using the simulation in many circumstances.

Client-side computation with browser-based simulations

Most standard web browsers can process Netscape JavaScript code embedded in web page source for client-side computation. The Internet Explorer browser processes JavaScript with its Jscript interpreter and can also process VBScript. The Netscape browser does not process VBScript, therefore JavaScript is preferred as an embedded scripting language.

Java applets are another way to perform client-side computation. Java applets are units of Java "byte code" that are delivered to a web browser and that are processed by a Java virtual machine that is invoked by the web browser. A Java applet can be displayed inside the main browser window or in an auxiliary browser window. Java source code is processed by a Java compiler to byte code. The byte code for a given program is the same for all OS platforms for which a platform-specific Java virtual machine is available. This allows the developer to post only one cross-platform version of byte code on the server for each applet. Java virtual machines can be obtained for most OS platforms, and many computers are shipped with a Java virtual machine installed.

Flash movies (also known as applets) can also be considered for client-side computation in web browser windows. Flash was originally developed for displaying vector-graphic animation in web pages but it has a scripting language that can be used for at least basic computations. Flash movies can only be developed or "authored" on Windows and Macintosh PCs, although they can be played on many Unix and Linux platforms as well.

Squeak is a programming environment that uses a subset of Smalltalk, one of the first object-oriented programming languages. Simulations can be displayed in browser windows by a plug in. Currently, Squeak development efforts are directed primarily toward K-12 education students.

Two other Macromedia products in addition to Macromedia Flash can be used for can be considered for client-side computation in web browser windows. Director projects can be developed and delivered to browsers (as Shockwave movies) only on Windows PCs and Macs. Authorware can only be developed on Windows PCs and can only be delivered on Windows PCs and Macs.

Authorware includes many tools for developing computer- and web-based training (CBT, WBT) courses. There are many other proprietary products on the market for developing such courses, with these relatively expensive tools being marketed primarily to corporate training groups.

Server-side computation with browser-based simulations

Students are presented with a web page that displays controls with which they input simulation conditions. The input values are then "posted" (sent with the html post method) to a CGI program on the server. The CGI program can perform the simulation calculations itself, or direct the inputs to another program on the server or another computer for computation. Results are incorporated into a web page description which is then sent back to the client for display in the browser.

Any programming or scripting language that runs on the server can be used. Here, a programming language is considered to be a language that requires variable types to be declared prior to use and usually compiles code to executable files. A scripting language is a language that does not require variable-type declaration and which is compiled to a byte code or is interpreted rather than compiled. Scripting languages usually provide less or no control over bit- and byte-level processes than do programming languages but have the advantage that many types of high-level processes can be accomplished in many fewer lines of code than would be required in a programming language (Ousterhout, 1998). C, Fortran, Visual Basic, and Java are examples of programming languages. JavaScript, PHP, Perl, and Tcl are examples of the many scripting languages available.

The product webMathematica is a combination of a CGI program, a Mathematica program, and the Mathematica application, all resident on a web server. This product allows the developer to use all the mathematical tools and libraries in Mathematica to perform the simulation calculations and generation of output displays. The clients only require a standard web browser and not a copy of Mathematica. Professor Brian Higgins of the University of California, Davis has a web site with several webMathematica modules in chemical engineering.

Matlab Web Server is a similar product for server-side computation. It uses the Matlab mathematical tool to do the computation. In comparison to use of a general programming language like Java, there is a great advantage to the presence of the mathematical libraries in the two specialized math products.

The language Octave, which has an extensive mathematical library and a syntax similar to Matlab, has also been used for server-side computation. Examples are shown at the "Interactive Modules" website of Professor John Ekerdt, University of Texas.

LabView is a product that is frequently used for laboratory process control. It can also be used to construct software simulations (Davis, 1995). LabView can apparently be used in conjunction with a LabView Web Server for server-side or client-side computation.

Dedicated program on client

In this option, a program resides on the client that has the ability to communicate with servers, and possibly other clients in peer-to-peer connections, using various Internet communication protocols.

Programs in this class are referred to as "rich clients" or "fat clients" because of their ability to perform many functions and communicate with several Internet protocols, as opposed to web browsers with more limited capability and which can be termed "thin clients." Another term for a rich client is "Internet application" or "Internet app."

The advantage of a running the simulation with a dedicated program include:

- The GUI can be designed specifically for the simulation, rather than having the simulation GUI nested inside a browser with a different GUI.
- The Internet communication protocols can include ones not supported, or not fully implemented, by a web browser.
- The simulation computations can be closely linked to the rest of the simulation program if desired.
- A wide range of programming languages can be used to construct the simulation and perform computations.
- The ability to operate in the absence of an Internet connection.

An on-line article "Beyond the Browser" presents a good argument for choosing rich clients over browser-based clients (Gaskin, 2003). Sun Computer lists the following criteria for choosing between an HTML client (browser-based client) and a rich client:

"HTML clients are particularly good in the following circumstances:

- The application must collect form input from the user and requires little interactivity.
- The use model fits a "wizard" style, executing a well-defined sequence of steps.
- The application is used on an infrequent or ad hoc basis (less than an hour per day).
- The user requires ubiquitous browser access and cannot rely on dedicated client machines.
- The application never needs to be used offline."

"Rich clients should be considered in the following scenarios:

- The user requires direct-manipulations of data or must perform significant editing operations.
- The user needs to input international characters not supported on the keyboard and cannot rely on the browser's support for input methods.
- The application is used frequently (more than an hour per day).
- The application should or must be usable offline.

- The application needs to perform computations that can be handled more efficiently in the client (rather than requiring constant roundtrips to the server)."

In most cases with rich clients, simulation computations would be performed on the client, since the alternative of a web browser is a reasonable interface for server-side computations. Dedicated programs that perform computations on the client have the advantage that they retain much of their usability whenever and wherever a student has access to a computer, not just when an Internet connection is available.

A wide range of programming and scripting languages can be used to construct rich clients. Several of the programming tools that were mentioned above for use with browsers can also be used to construct rich clients. Java can be used to write dedicated programs with the ability to communicate over the Internet. The Java Web Start interface can be used in the absence of a web browser to download Java applets from servers. A Flash interface similar to Java Web Start is planned for release by Macromedia as "Macromedia Central" in Summer 2003. Macromedia Director can also be run on a client with Internet communications and differs from Flash in that it has access to the client file system through a security access system.

Security considerations

Whenever code and other information are transferred over a network, the potential exists for viruses or other rogue code received from the network to cause corruption of data on the client and transfer of private information from the client to the network.

Browser-based simulations hold an advantage here as a result of deliberate security measures embedded in the browsers. Browsers have very limited access to a client's disk file system, able to read and write data only from files called "cookies" that are restricted to a given location on the disk. However, students may encounter web sites where they can download and run executable code in the form of Microsoft ActiveX controls. These controls may have unlimited access to the client disk files.

Depending on the programming tool and programming choices used to construct a dedicated program, such a program may have access to the client disk, thus, presenting the potential for viruses to corrupt the file system. Restricting the dedicated program's access to receiving information from only "trusted" servers is one way to reduce this risk.

Programming tools

In addition to making decisions about where computations are done and what type of main client program to use, a developer must also choose one or more tools with which to construct the simulation. There is a very wide range of choices available, all the way from C to a commercial WBT authoring tool. Program modules may be written with more than one tool.

Factors involve in making the choices include:

- Number of staff available
- Existing skills of staff
- Availability of training staff in new programming tools
- Planned scope of the project
- Time table
- Budget
- Number of server and client OS platforms to be supported
- Communication protocols to be used
- Need for mathematical libraries to perform simulation computations

Several tools have been mentioned above: JavaScript, Java, Flash, Director, Authorware. We must also mention the new Microsoft .NET initiative. This initiative is designed to allow construction of many types of browser-based and rich client programs. A variety of Microsoft programming tools in the "Visual" family, including Visual C++, Visual C#, and Visual Basic can be used to make .NET programs and modules. One drawback, especially for widely deployed educational and academic simulations, is that this is platform is restricted to development on the Windows OS platform and to deployment only on Windows OS servers and, in many cases, only Windows OS clients.

Several scripting languages were mentioned above for writing CGI programs, including Perl and PHP. Some scripting languages can be used to write rich clients, in addition to CGI programs. These include Tcl and its GUI toolkit Tk, whose combination is referred to as Tcl/Tk, Python in combination with Tk or another GUI toolkit, and REBOL. These scripting languages are available for development and deployment on essentially all OS platforms (Win, Mac, Unix, Linux). Although these are scripting languages with many high-level instructions available, their syntax is fairly terse, similar to a standard programming language in many cases.

There exists a class of scripting languages whose lines read almost like English sentences. This can be a significant advantage for rapidly constructing programs with complex GUIs and Internet communication capabilities. Examples of tools that use such languages (and the OS platforms on which they can be developed and deployed) include Runtime Revolution (Win, Mac, Unix, Linux), Director (Win, Mac), and ToolBook, (Win).

Yearly licensing fees for Authorware and other WBT tools range from $2500 upwards. For the other tools mentioned here, yearly licensing fees range from zero, for open-source tools, to under $1000 for proprietary tools. Since the labor required to develop a significant software package will be large, we feel that yearly licensing fees under $1000 should not be a significant factor to educators when choosing a tool. Open source tools may be attractive to many because they are "free," however this factor is offset by the fact that a developer may, in some cases, have fewer or less convenient development tools available, thereby increasing labor requirements.

Prototype rich client for simulations

We currently have a standalone simulation program, the Reactor Lab, which does not have Internet communication capabilities [note: the English version of the Reactor Lab was integrated with the Internet in Fall 2003]. The Reactor Lab provides simulations of a variety of chemical reactors. We are developing a rich client simulation prototype that does have integrated Internet communication capabilities, and plan to use the lessons learned and code developed from the prototype to convert the Reactor Lab to a rich client.

The Reactor Lab is constructed using Runtime Revolution, and this tool was also used to develop the prototype here. Revolution, allow rapid development of cross-platform, interactive programs. In comparison to Java applets, which are also cross-platform, Revolution modules are smaller in size, initialize faster, display and manipulate GUI objects faster, and require a factor of three to ten times fewer lines of program code to be written. On the other hand, Java, is a more general programming tool that provides more options and capabilities. The choice among one of these or other possible tools depends on many factors listed in the section above.

The ERC Web Courses was developed in collaboration with and partial support from the NSF/SRC Engineering Research Center for Benign Semiconductor Fabrication.

The prototype contains only one OS platform-specific executable file, the "engine." The engine file contains the Revolution execution engine and a few lines of "boot strap" script that connect to the other files in the prototype, or to the "home" server in case the engine file gets separated from the other files. An "engine" can be built for all major OS platforms. Other files consist of "support" files which contain common script such as communication modules, and also "simulation module" files. A simulation module is usually a simulation of a specific system, with the prototype being able to access several and, eventually, many simulation modules. The support files and the main module files contain cross-platform byte code and, in some cases, plain text. Other module files can be executable files used for simulation computations, including Windows .dll files.

When a student starts the prototype for the first time, the "directory window" is displayed and a connection is made to the home server. With its connection to the home server, the directory first checks for new versions of the support files, and then downloads and runs any new versions in the current session, including new versions of the directory itself. Then the directory displays a list of modules available on the selected server. One or more servers with modules are available for selection by the student. When the student selects a specific module, the module is downloaded and opened on the client. In the current prototype, the module and its associated files are also saved to the client disk. In subsequent uses of the prototype, the student can work "on line" with a connection to the Internet, or can work off line and use modules with local copies saved to disk in prior uses. If a student is working on line and selects a module with a local copy, the directory opens the local copy if it is the current version, or downloads, saves and opens a newer version if available.

Revolution script executes relatively fast, since the byte code representing the script is processed using a virtual compiler, which itself is compiled C code. This means that many simulation computations can be done directly in the same script used to manipulate the GUI objects. However, in cases where large numerical arrays are involved, a standard programming language such as Fortran should be used for the simulation computations.

One goal of the prototype was direct use of Fortran programs developed during the course of research projects. This was accomplished in two of the modules in the prototype. A Fortran listing was received from each of two graduate students. One program simulated nonequilibrium adsorption of a trace contaminant in ultrapure water onto an adsorption bed. The other simulated photocatalytic degradation of a trace contaminant in ultrapure water.

For the initial purposes of the prototype, the simplest strategy was chosen for linking a GUI to the programs: the GUI writes input to a text file, tells the OS to run the computation program, and then reads the output text file produced by the computation program and displays results.

The Fortran programs were altered only to the extent to make them read input from a text file and write output to a second text file in specified formats. The programs were then compiled into executable programs that run without displaying a window. Although the GUI file is cross-platform, the computational program needs to be compiled separately for each OS platform. So far, the programs have only been compiled to run under Windows OS.

A student enters desired input parameters into a form in the GUI, then clicks a "run" button. The script in the GUI file writes the input values to a text file, issues a command to the OS to run the computation program executable file, waits until the computation program finishes, and reads results from the output text file written by the computation program. Finally the GUI displays the results.

Since accessing the input and output disk files only takes a fraction of a second, this simple strategy will work well for many simulations. In other cases, especially where the student must be able to make changes at arbitrary times during a continuously running simulation, it will be necessary to use a strategy that allows communication between the GUI and the computation program through RAM rather than disk.

So far, we have concentrated on designing and scripting the network communications and versioning functions. The modules available with the current prototype are simple and only demonstrate a small fraction of the possible capabilities. The small windows and crude graphics used in the prototype could be replaced by large windows with professional graphics with no change in program code required. Inclusion of other media types such as photographic images, movies, sound and animations is a standard capability of the tool we are using.

Our next objective is to consider security issues. Since the rich client will only be able to connect to trusted servers, security issues are not as critical as for programs like web browsers that have unrestricted access to web sites. Inclusion of "certificates of authenticity" in modules that are checked by the rich client are one possibility. An option can be provided to operate in "secure mode" which allows no disk access, with the disadvantage that modules can't be saved for offline use and external code resources like the Fortran executables can't be used when in secure mode.

One lesson that we learned from developing the Reactor Lab is to design software so that display text strings are kept separate from the program code for easy translation of the text to other languages. Courseware posted on the web is going to be accessed by people around the world. Having versions in several major languages will make the content more accessible to those outside English speaking countries.

Another lesson learned from Reactor Lab development is that developing a complex software package takes an incredible amount of time, even though we are using a very high level, English-like scripting tool. One reason is that software must work and must produce accurate results. A few typos in a text are usually expected and not a major problem. A few "typos" or logical oversights in a software program can prevent it from functioning or functioning accurately, a problem especially critical with simulations. Another reason is that there are few design guidelines in constructing software simulations. Everyone knows what a textbook looks like: contents, chapters with references and homework problems, index. In contrast, here are no standard models for software simulations.

Conclusion

Coupling interactive simulations with resources on the Internet is an exciting approach to education and research. Development of networked simulation software involves making many design choices and tool selections. We believe there is no one "best" set of choices, especially since the scope and goals of different projects will differ significantly. With future development, we expect that this will continue to be the case, although we expect that a few successful models will emerge to help narrow the choices to be made. Educators that wish to construct a successful major software project must be dedicated enough to invest significant time.

Acknowledgements

This work was supported by the NSF/SRC Engineering Research Center for Environmentally Benign Semiconductor Manufacturing through a seed project and the National Science Foundation through grant DUE-0125076.

References

Davis, R.A. "Create Virtual Unit Operations With Your Data Acquisition Software," Chem. Eng. Educ., Fall 1995, pp. 271-274.

Ekerdt, J. G. "Interactive Modules" in chemical reaction engineering using Octave http://www.che.utexas.edu/reactor/index.html (2003).

Elgamal, A. "Web Shaker" on line experiment, University of California, San Diego http://webshaker.ucsd.edu (2003).

Gaskin, R. "Beyond the Browser" http://www.fourthworld.com/embassy/articles/netapps.html (2003).

Henry, J. "Resource Center for Engineering Laboratories on the Web," University of Tennessee at Chattanooga http://chem.engr.utc.edu/ (site last updated 13-August-2002).

Higgins, B. webMathematica modules http://www.higgins.ucdavis.edu/webmath.php (2003).

Ousterhout, J. "Scripting: Higher-Level Programming for the 21st Century," IEEE Computer March, 1998 http://www.tcl.tk/doc/scripting.html

SOFTWARE

Authorware http://www.macromedia.com/software/authorware/
Director http://www.macromedia.com/software/director/
ERC Web Courses https://github.com/RichardHerz
Flash http://www.macromedia.com/software/flash/
Flash display format http://www.macromedia.com/software/flashplayer/
Java http://java.sun.com/
Java Swing http://java.sun.com/products/jfc/#swing
Java Web Start http://java.sun.com/products/javawebstart/
JavaScript http://devedge.netscape.com/central/javascript/
Jscript http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/js56jsoriJScript.asp
LabView http://www.ni.com/labview/
Macromedia Central http://www.macromedia.com/software/central/
Mathematica http://www.wolfram.com/products/mathematica/index.html
Matlab http://www.mathworks.com/products/matlab/
Matlab Web Server http://www.mathworks.com/products/webserver/
Microsoft .NET http://www.microsoft.com/net/
Octave http://www.octave.org/
Perl http://www.perl-foundation.org/
PHP http://www.php.net/
Reactor Lab http://www.reactorlab.net/
REBOL http://www.rebol.com/
Runtime Revolution http://www.runrev.com/
Python http://www.python.org/
Tcl/Tk http://www.tcl.tk/
ToolBook http://home.click2learn.com/en/toolbook/index.asp

Shockwave display format http://www.macromedia.com/software/shockwaveplayer/
Squeak http://www.squeak.org/
Squeak K-12 http://www.squeakland.org/
Sun Computer, choosing between HTML- and rich-client http://java.sun.com/products/jfc/index.html
Visual C++ http://msdn.microsoft.com/visualc/
Visual C# http://msdn.microsoft.com/vcsharp/
Visual Basic http://msdn.microsoft.com/vbasic/
VBScript http://msdn.microsoft.com/library/default.asp?url=/library/en-us/script56/html/vtoriVBScript.asp
webMathematica http://www.wolfram.com/products/webmathematica/index.html