ReactorLab.net Laboratory Simulations

Richard K. Herz
Chemical Engineering Program &
Mechanical and Aerospace Engineering Department
University of California, San Diego, USA 92093-0411

Abstract

ReactorLab.net provides simulations of a variety of chemical reactors for use in chemistry and chemical engineering education. The overall software framework is field-independent; only individual lab modules are field-specific. The software framework is that of a "rich client" or "Internet application," with full Internet communication capabilities. The software is scripted in a very-high-level, platform-independent language, Runtime Revolution, with only the core engine application files being specific to processor-OS platforms. The overall development and deployment strategy is similar to that of Sun's Java and Microsoft's .NET but can be implemented and maintained faster and with less training.

A student initially downloads the core software - engine application file and byte-code-compiled scripts - from ReactorLab.net. Students have the option of working on- or off-line. Whenever a student goes on-line, the core software on the student's computer automatically updates itself when new versions of files are available on the server. The core engine application file contains only enough script to "bootstrap" itself and, thus, is not updated except when major new features are added to the core engine, e.g., annually or less often. Lab modules a student accesses on the net are saved locally so that the student can work off-line. When they go back on-line and new versions of a lab are available, the local copy is automatically updated.

Students can perform experiments, view results, and save data for analysis with commercial data analysis software such as Excel or Matlab. Many labs have quizzes in which each student gets a unique set of unknowns, with student responses being scored automatically with virtual $. The Lab "conference room" allows students to post messages to a bulletin board, and "chat" with other on-line users who have the conference room feature active. Non-networked versions of the Lab are available in Spanish and Portuguese.

Introduction

Early in my teaching career I realized that only a small subset of people, which includes college professors, can learn solely by reading or by listening to someone talk for extended periods. In order to enhance the learning of all students, especially those that do not belong to this subset, I wanted to supplement lectures with demonstrations and "hands on" experiments in a lab.

Space, cost, environmental, and safety considerations were a barrier to doing this. I decided that interactive software simulations of the chemical reactors my students studied could help them learn by supplementing the static material in their textbook.

The goal was a laboratory, in software, filled with chemical reactors which the students could operate. A variety of reactors with different features and complexities would be present to illustrate different concepts. Since there are several outstanding texts in the field of chemical reaction engineering, the focus was to provide the experimental apparatus and data collection methods rather than complete pedagogical units. Specific reactors and reactions, or classes of reactions were developed since the Lab was not meant to be a general reactor simulator. Thus, development of the Reactor Lab began.

There were few educational simulations available as models in 1993. A graphical user interface was desired to simulate working in a laboratory, not just a collection of faceless computer programs. The World Wide Web had just started to expand beyond CERN, and the first popular web browser, Mosaic, had just been released, so interactive web simulations were far in the future. HyperCard[1] was chosen as a development tool because of its ability to seamlessly couple graphical user interface objects to a very-high-level computer language, or script[2].

The development of the Reactor Lab followed a "bottoms up" or evolutionary approach. One reactor simulation, or lab, was developed and used as a homework assignment in the course. Student responses guided development of a second simulation, and so forth. After several labs had been developed and a navigation system devised, patterns and design guidelines emerged. At several times during the past ten years, the design guidelines that emerged were used to restructure the lab and its underlying script.

The development tool also changed from HyperCard to cross-platform MetaCard[3], and then to Runtime Revolution[4] when it acquired MetaCard in 2003. Each new tool included the scripting language and object model of the previous tool as subsets, so no development effort was lost in the transitions and much was gained with the enhanced capabilities of the new tools.

Some of the recent additions to the scripting language are commands that implement communication over the Internet. The Reactor Lab recently evolved from a standalone program into an "Internet application" which provides lab modules on demand and updates when available, and enables communication between students and instructors.

The following sections describe the current version of the Lab, plans for the future, and recommendations to educators interested in developing their own simulations.

Student interaction with the Lab

Students download the Lab from ReactorLab.net, expand a compressed archive file, and then open the Lab. Installation in local school labs can also be arranged. After a welcome screen, and network-connection and update progress displays, the student sees the Lab Directory, as shown in Fig. 1.

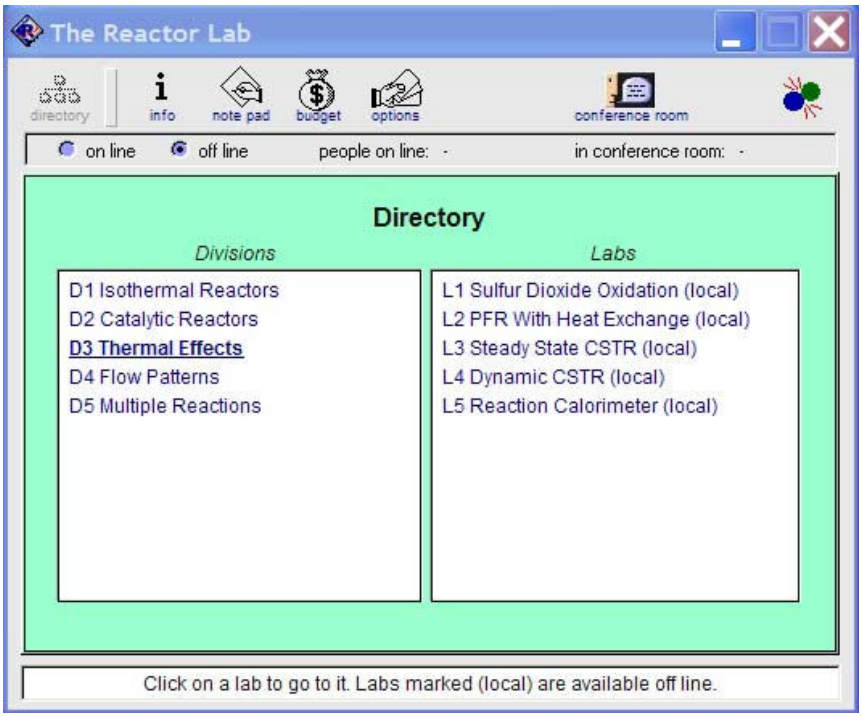

Fig. 1

The Directory and most other windows have a set of navigation buttons along the top and an information field at the bottom that displays information about objects on the window that the cursor passes over. This information window, along with the consistent design of the lab from module-to-module, makes the Lab easy to operate with little training.

The modules, or "labs," in The Reactor Lab, are grouped into divisions. The list of divisions is shown in the field on the left of the Directory. When the student clicks on a division, the labs in the division are listed in the field on the right. When the student clicks on a lab, the lab window opens and the Directory window closes.

Fig. 2 shows a lab. This lab simulates an exothermic reaction in a Continuous Stirred Tank Reactor (CSTR) and shows the consistent graphical layout of lab modules: navigation buttons at top, inputs the user can vary on the left, process in the center, and output on the right. The plots on the right are continuously updated, here showing sustained oscillations in reactant concentration and temperature that result because of nonlinear coupling between reaction kinetics and heat transfer.

The student can return to the Directory by clicking the Directory navigation button in the button bar of the lab window. Other navigation buttons open windows with context sensitive information, a quiz, a table of the data obtained, a plot of the data obtained, option settings, quiz budgets, and a conference room for posting messages to other users.

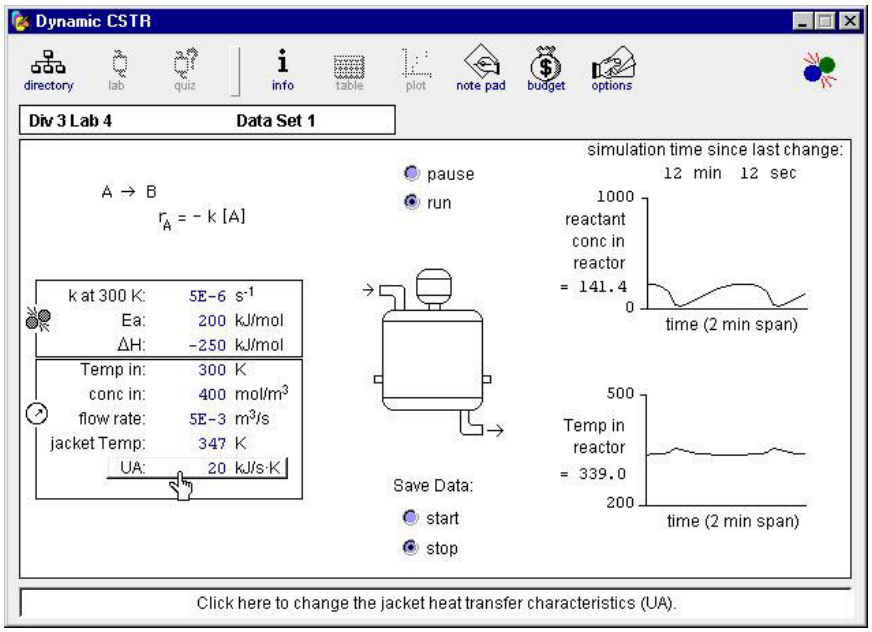

Fig. 2

Pavia[5] listed 24 high priority elements that should be exhibited by a laboratory simulation. The Reactor Lab currently incorporates 23 of these elements. The exception is: "charge students more for the use of expensive tests than cheap tests." Currently, all experiments in Lab quizzes charge the same virtual dollar cost for an experiment.

Fig. 3 shows the quiz section of a lab module that has animation, with the pressure gauge dial moving and the color of the reactor contents changing from blue (reactant) to red (product) as the reaction proceeds.

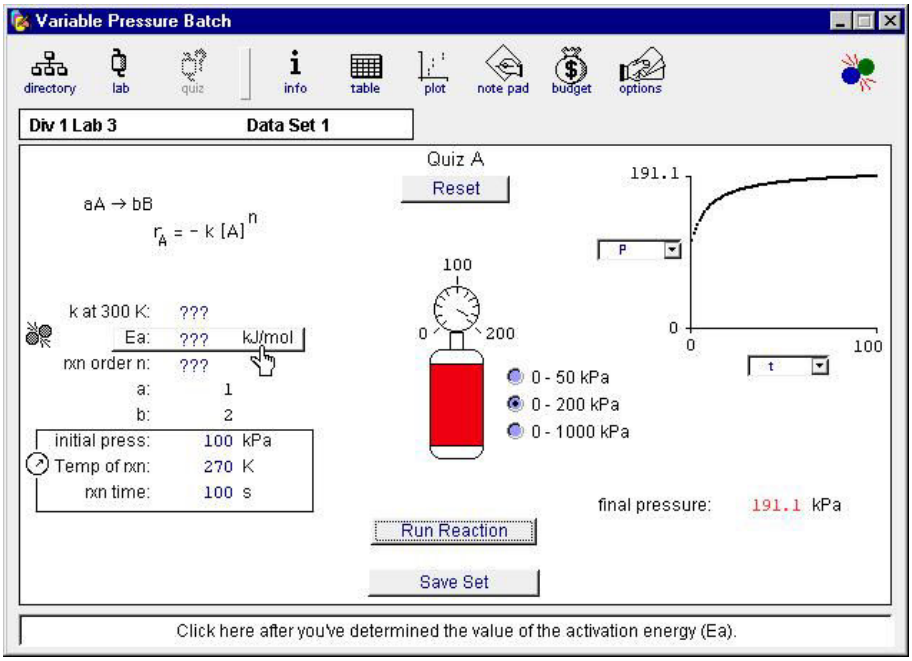

Fig. 3

Sliders, dials, and direct input fields are also available as input controls. Input controls and output displays may be distributed over a process image as well as being grouped in blocks.

Fig. 4 shows a module in which the image of the reactant concentration profile inside a porous catalyst pellet is continuously updated as the inputs are varied with slider controls.

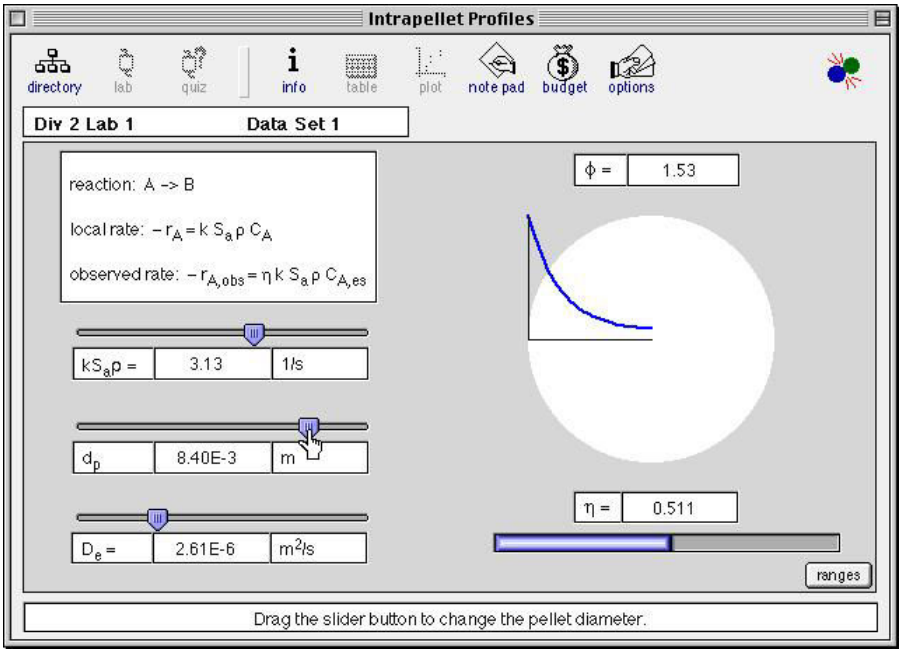

Fig. 4

Types of labs available include:

(a) Single data point per experiment, where "data point" includes values of several output values at one time during a transient experiment or at the output of a continuous steady state process.

(b) Time series of data points in one experiment on a transient or dynamic process conducted to a final time. (e.g., Fig. 3)

(c) Continuous output during a continuous dynamic simulation. (e.g., Fig. 2)

(d) Simulations in which results are displayed on the screen but not recorded by the program for saving to disk. Plots of results are provided in the lab window. (e.g., Fig. 4)

In lab types (a)-(c), student can save data to disk in one of several formats. Data saved to disk can be imported into program of student's choice, such as Excel and Matlab, for plotting and analysis. A simple plotting function is provided in the Lab itself for data saved to disk in the current session in order to compare results of a series of experiments to make sure good results have been obtained.

Student assignments

Several types of assignments can be made. In qualitative assignments, students are asked to perform experiments and write a comparison of behavior between two reactor types or, for example, comment on the sensitivity of self-sustained oscillations in the Dynamic CSTR (Fig. 2) to changes in parameter values.

Automatically scored quizzes can be assigned, such as that shown in Fig. 3. Each student is given a unique set of unknowns upon each entry into a quiz. The student has to decide what experiments to perform in order to get data that can be analyzed in order to obtain the unknown input values. When they supply their answer, the program tells them whether or not it is correct. Correct answers are awarded virtual $, which are recorded in the Lab's Budget. Students are required to turn in copies of their data and analysis work, in addition to a code-authenticated copy of the Budget.

In longer assignments, one can ask students to determine the complex kinetics of the catalytic methanol synthesis reaction, or use the SO_2 oxidation reactor to design and optimize a multi-stage reactor-heat exchanger system.

Student responses

Students learn from two main differences from textbooks and exercises: (1) they get to interact with a model of the process, and (2) they have to design experiments and generate data themselves that will lead to solving a problem.

Some students appear shocked to encounter an assignment so different from a textbook problem. After running an experiment, they stare at the screen, then after a while ask me, "what do I do?" They ask this, not because they don't understand how to use the software, but because they are only comfortable working the usual textbook problems where a necessary and sufficient set of input is given them and a single quantitative answer is asked of them.

The vast majority of my students have liked using the lab in the course. They have said that it helps them get a better qualitative feel for the material.

Some have made good suggestions which have been implemented, for example, make info text printable and copy-able, and make the last data point in the thumbnail plot a different size/color to distinguish it from previous points.

Much can be learned by watching students perform assignments. One lesson is that they value program responsiveness and speed of operation over graphical richness (good because the current artwork is minimal) because their number one objective is to get the assignment done fast.

Implementation

The lab consists of a single application file and folders of other files that contain script and screen displays. Individual lab modules are contained in separate files. The application file contains the core execution "engine" plus a minimal amount of "bootstrap" script. The platform-specific application file can be built for most widely used OS-processor platforms. Script and screen display files are partially compiled to a "byte code" during development, and these files are platform independent.

The Lab is built with the software development tool Runtime Revolution, which is licensed by Runtime Revolution, Ltd. Revolution provides a high level scripting language and tools which allow construction of all of the components of the Lab, including user interaction, graphics and animation, text handling, computation, Internet communications, and server common gateway interface (CGI) programming. Revolution is a mature and stable development tool whose predecessors have evolved over the last 17 years. Apple Computer's HyperCard appeared in 1987 and established an object model and scripting language for rapid application development. MetaCard appeared in 1992 and extended HyperCard's model to Unix, and later to Windows, Mac and Linux platforms, while keeping HyperCard's object model and scripting language as a subset. Runtime Revolution Ltd. acquired MetaCard in 2003 and continues its development as Revolution.

Revolution is similar to Sun Computer's Java[6] in that both use a platform-specific engine or "virtual machine" to execute platform-independent byte code. There are many differences. When the development tool was re-evaluated before Internet connectivity was added, Revolution was chosen through use of the 80/20 rule: it was considered to provide 80% of the power of Java in 20% of the development time. Development speed is a vital consideration for part-time development by a professor and students.

In the Reactor Lab, all files other than the application file are automatically updated whenever new versions are available on the server at ReactorLab.net and the user works "on-line" in the Lab. New divisions or labs posted on the server appear automatically in the Directory. Since the lab files are platform independent, only one set has to be maintained on the server.

In order to reduce download times, only a few sample lab files are contained in the Lab distribution file that a student initially downloads, although all labs available on the server are listed in the Directory, with labs saved locally denoted by a (local) label. When a student selects a lab in the Directory, its files are downloaded and saved to the student's disk if current versions are not already present on the disk. Thus, after opening a lab for the first time, students may work either on or off line.

The application file only needs to be updated by a student when significant upgrades are made to Revolution and are implemented by the Lab, and this implementation is expected to be no more frequent than annually. The update would be done by downloading a new Lab distribution from the server.

When a student saves simulated experimental data to disk, each data set is saved in a separate text file in a session data folder. Also saved in the session data folder is the Note Pad text file, which is a record of automatic notations relating data file name to division and lab number and list of variables changed, plus observations recorded by the student.

Because lab files are downloaded in response to the same hypertext transport protocol (HTTP) requests that web browsers use to get web pages, such requests are logged the same way all web servers log requests. Thus the patterns of usage from individual Internet protocol (IP) addresses or IP zones can be obtained by analysis of server logs. Additional activity logging can also be implemented by posting information to CGI scripts on the server, which can also be written in Revolution.

Note that the software framework is field independent. Only the specific process models relate to chemical reactors. With the planned lab construction kit, students and other teachers could develop lab modules in their own field for distribution via the Lab framework.

Distribution and languages

Whereas the Lab was originally intended to help my own students, I thought about offering it to other instructors in the U.S. from the beginning. After it had developed to a stage where it could be distributed, the Web had evolved to a point where it was a natural choice for low cost distribution.

At the download page, users are asked, but not required, to enter their name, email address, and school or company. This information is saved in a log file on the server. In most cases, users appear to provide valid information. Invalid information can be identified by entries of gibberish, outlandish names, or information that isn't consistent with the IP address of the requesting computer.

With the first downloads, I was interested to see that most users were located outside the U.S. After a couple years, I got a proposal from a professor in Brazil to join me in translating the text of the Lab into Portuguese. Then it struck me, there is a great demand for non-English versions of educational software. Then I panicked when I realized that the display text was scattered throughout the script since I hadn't considered translation. Fortunately, the powerful development tool allowed scripts to be written which scanned the Lab and extracted display text, which was sent to the translator, and then insert the translated versions. Much work was left to be done, however, and I recommend that developers plan for display text and language translation from the start and follow the rule to "keep code and data (i.e., display text) separate" in projects. The Portuguese translation was followed by a Spanish translation. Currently, the Spanish version of the Lab is downloaded at roughly the same rate as the English version.

Over 8,700 people downloaded the Lab from 96 countries in the first three years of download logging (through April 2003). We have heard from students and instructors using the Lab in courses in Bolivia, Brazil, Chile, Italy, Mexico, The Netherlands, South Africa, Spain, Turkey, and the U.S.

Communication and collaboration

With connection of the Lab to the Internet, communication between users can be implemented. Since users are distributed around the world, just saying “hello” to each other would be interesting. Communication between users would also allow them to help each other with use of the Lab.

The initial communication mode implemented is a "conference room" where users can post messages and see their message and those of others appear essentially immediately after they are sent. The conference room functions both as a bulletin board and as a chat room between users who are on-line at the same time.

The conference room was introduced recently. Messages have been posted from 11 different time zones in first few months of use. So far, it has been used mainly by my own students doing homework assignments in the Lab and helping each other, or getting help from me when I'm on-line. One day, by chance, we had a three-way chat between me, a postdoctoral fellow in the Mid-West U.S. who had been educated in Turkey, and an undergraduate student at a university in Turkey.

In the future, different message boards will be added for different languages, for specific labs, and for specific instructors and their courses. Activity tracking will be added to record and post logs of session history such as lab modules accessed, time spent, and number of experiments performed.

Plans

Under development is a “lab construction kit” which will allow other teachers and students to develop new lab modules. This potentially will increase the number of labs available and also will allow the framework of the Lab to be used in other fields.

One new type of lab module under development is a simplified plant simulator where reactors, separators, and heat exchangers can be added and connected in arbitrary configurations. Individual unit models in the plant can be located on the local student computer or can be located on other computers on the Internet, e.g., those of other students or student teams. Messages exchanged between unit models are in plain text with information elements specified in extensible markup language (XML) to provide model-language and platform independence. Initial tests of a two-unit plant, with one unit in San Diego and one unit in Houston, obtained an average messaging time of 0.18 s per time step for 4^{th}-order Runge-Kutta integration of the process models via TCP/IP socket connections at the high Internet traffic time of mid-day and mid-week.

Lessons and recommendations

The main lesson is that software simulations engage students in active learning and are a powerful teaching and learning tool.

Another lesson is that, whereas development of the Lab has been rewarding, it has also been exceptionally time consuming. Software has to work when used by impatient students in diverse computing environments, and getting the design right and bugs out takes time. Educators who want to develop more than a couple individual simulations should seek partners and significant financial support at an early stage.

For someone starting to develop a set of simulations in the present day, there are enough models available to allow some initial "top down" design. However, I still encourage one or two simulations to be produced and used before the design process progresses very far. Other recommendations are discussed in sections above.

Acknowledgement

Support has been provided by the U.S. National Science Foundation through grant DUE-0125076.

Bibliographic information

1. Ousterhout, J. K., "Scripting: Higher Level Programming for the 21st Century," IEEE Computer, March 1998, and on the web at <http://home.pacbell.net/ouster/scripting.html>.
2. HyperCard (software tool), Apple Computer, Inc. <http://www.apple.com/hypercard/>.
3. MetaCard (software tool), MetaCard, Inc. <http://www.metacard.com>.
4. Runtime Revolution (software tool), Runtime Revolution, Inc. <http://www.runrev.com>.
5. Pavia, D. L., "SQUALOR: The design of a laboratory simulation," *Academic Computing*, February 1990, p. 8.
6. Java (computer programming language and software tools), Sun Microsystems, Inc. <http://java.sun.com>

Biographical information

Richard Herz's research interests are chemical reaction engineering and catalysis over surfaces. He can be reached at herz@ucsd.edu and his research home page is at <http://mechanics.ucsd.edu/research/herz/>.