

Eastern Michigan University
DigitalCommons@EMU

Master's Theses and Doctoral Dissertations


Master's Theses, and Doctoral Dissertations, and
Graduate Capstone Projects

2006

Implementation of labcreator and the integration of cyberlab

Kang Zhao

Follow this and additional works at: <http://commons.emich.edu/theses>

 Part of the [Education Commons](#), and the [Software Engineering Commons](#)

Recommended Citation

Zhao, Kang, "Implementation of labcreator and the integration of cyberlab" (2006). *Master's Theses and Doctoral Dissertations*. 81.
<http://commons.emich.edu/theses/81>

This Open Access Thesis is brought to you for free and open access by the Master's Theses, and Doctoral Dissertations, and Graduate Capstone Projects at DigitalCommons@EMU. It has been accepted for inclusion in Master's Theses and Doctoral Dissertations by an authorized administrator of DigitalCommons@EMU. For more information, please contact lib-ir@emich.edu.

IMPLEMENTATION OF LABCREATOR
AND THE INTEGRATION OF CYBERLAB

by

Kang Zhao

Thesis

Submitted to the Department of Computer Science

Eastern Michigan University

in partial fulfillment of requirements

for the degree of

MASTER OF SCIENCE

in

Computer Science

Thesis Committee:

Matthew Evett, PhD, Chair

Augustine Ikeji, PhD

William McMillan, PhD

October 25th, 2006

Ypsilanti, Michigan

APPROVAL

IMPLEMENTATION OF LABCREATOR
AND THE INTEGRATION OF CYBERLAB

Kang Zhao

APPROVAL:

Matthew Evett, PhD, Chair
Thesis Chair

Date

Augustine Ikeji, PhD
Committee Member

Date

William McMillan, PhD
Committee Member

Date

William McMillan, PhD
Department Head

Date

Deborah de Laski-Smith, PhD
Interim Dean of the Graduate School

Date

ACKNOWLEDGEMENTS

I take this opportunity to thank my advisor, Dr. Matthew Evett, for his support and encouragement during the development of the project, and for his comments and revision of the thesis.

I am very grateful to Dr. Ikeji and Dr. McMillan for serving as my committee members and providing many helpful suggestions.

I would also like to thank the faculty, staff, and fellow students at the Department of Computer Science, who made my study here a very pleasant experience.

Last, but not least, I thank my fiancée Yu. It is your love that drives me forward.

Kang Zhao

ABSTRACT

With the development of the World Wide Web, online courses are becoming more and more popular in modern science education. CyberLab aims to solve an important issue in distance science education -- laboratory experiments in online courses. It is a toolkit that handles creation, exportation, and execution of virtual experiments (within web browsers). It consists of LabCreator and LabExecutor. With LabCreator, instructors can create virtual experiments and export them into intermediate files. Students can download those files from online course websites and execute them in LabExecutor on their own computers.

The paper reports on the completion of two important tasks in the development of CyberLab: (1) the implementation of LabCreator and (2) a system allowing exportation of the experiment to intermediate web accessible format and the loading of the experiment into LabExecutor.

The feasibility of the design and structure of CyberLab is proved by integrating the LabCreator and LabExecutor for the first time. The advantage of CyberLab is shown through a demonstration of the deployment of a virtual experiment.

TABLE OF CONTENTS

ACKNOWLEDGEMENT.....	i
ABSTRACT	ii
CHAPTER 1 INTRODUCTION.....	1
1.1 Online courses	1
1.1.1 Advantage and disadvantage.	1
1.1.2 Existing online course platforms.	2
1.2 Virtual laboratories for online science courses.....	2
1.2.1 The importance of laboratories in science courses.	2
1.2.2 What is a virtual laboratory?.....	3
1.2.3 The benefits of using virtual laboratories.	3
1.3 Related works in online courses laboratories.	4
1.3.1 Simulations implemented in APL.....	4
1.3.2 Multimedia based online laboratories.....	4
1.3.3 Online hybrid laboratories.	6
1.3.4 Pre-developed online virtual laboratories.....	8
1.3.5 Online virtual laboratory creation tool.	13
1.4 Brief introduction to CyberLab	14
1.4.1 Purpose of CyberLab.	14
1.4.2 Structure of CyberLab.	14
1.4.3 Model of experiments in CyberLab.....	15
1.4.4 Previous work in CyberLab.....	16

1.4.5 My work in this thesis.	17
CHAPTER 2 IMPLEMENTATION OF LABCREATOR	19
2.1 Graphic User Interface (GUI) of LabCreator.	19
2.2 Experiment widgets.	21
2.3 What You See Is What You Get (WYSIWYG).	22
2.3.1 Drag and Drop.	22
2.3.2 Appearance of widgets.	23
2.4 Data structure of CyberLab virtual experiments.	24
2.5 An example of how a virtual experiment works.....	26
2.6 Create virtual experiments with LabCreator.....	27
2.6.1 The simple heating experiment.....	27
2.6.2 The procedure to create the heating experiment.....	29
2.7 Preview experiments in LabCreator.	33
CHAPTER 3 INTERGRATION OF LABCREATOR INTO CYBERLAB.....	37
3.1 Java Object Serialization.	37
3.2 Export experiments from LabCreator.....	40
3.3 Load and execute experiments in LabExecutor.....	41
3.4 Usability test and feedback.....	42
CHAPTER 4 CONCLUSION AND FUTURE WORK.....	45
4.1 Conclusion.	45
4.2 Future work.....	47
REFERENCES	52
APPENDIX A: An excerpt of the Java code for widget-related classes.	58

APPENDIX B: An excerpt of the Java code for widgets drag and drop.....	59
APPENDIX C: An excerpt of the Java code for Java serialization.....	61

LIST OF FIGURES

Figure 1. Structure of CyberLab.....	15
Figure 2. Graphic User Interface of LabCreator.....	19
Figure 3. The appearance of a CyberButton.....	21
Figure 4. The appearance of CyberTextField.....	21
Figure 5. Dialog for CyberButton initial settings.....	23
Figure 6. Right-click pop-up menu for CyberButton	24
Figure 7. FSA diagram of the heating experiment	28
Figure 8. GUI of the heating experiment.....	29
Figure 9. FSA Menu of LabCreator.....	30
Figure 10. Dialog for creating new variable “temp.”.....	30
Figure 11. The right-click pop-up menu for Text Display widget	30
Figure 12. Dialog for creating new state “Medium.”	31
Figure 13. Set Starting State as “Medium.”.....	31
Figure 14. Dialog for creating a FSA edge.....	32
Figure 15. GUI of LabCreator after the heating experiment is created.....	33
Figure 16. Preview of heating experiment (initial state “Medium”)	35
Figure 17. Preview of heating experiment (initial state “Hot”).....	35
Figure 18. Preview of heating experiment (initial state “Cold”).....	36
Figure 19. Part of the “.exp” file for the heating experiment	41
Figure 20. The heating experiment in LabExecutor	44
Figure 21. The FSA of the free falling experiment.	48
Figure 22. The prototype for the graphic user interface of “Equation Editor”	50

CHAPTER 1

INTRODUCTION

1.1 Online courses.

Online courses have been widely used in distance education. Many universities in the United States, including Eastern Michigan University, have offered online courses. The credits from online courses can now be used to fulfill the requirements for obtaining a degree or certificate. Some universities even offer degree programs totally online, such as the online M.S. in Computer Science program at George Mason University [28].

1.1.1 Advantage and disadvantage.

Online courses owe a great deal of their popularity to their flexibility [21]. They are usually described as “any time, any place” learning [9]. With online materials, a student is able to adjust study time and pace according to his or her own situation and habit. With Internet connections at home, students and instructors do not have to spend time and money to commute any more.

Other advantages of online courses may include low requirements for teaching facilities, better monitoring of each student’s progress, and helping students to develop ability in independent learning [7].

Every coin has two sides. It is argued that course completion and program retention rates are generally lower in distance education courses than in their face-to-face counterparts [2]. It may also take time for instructors to learn to use online course tools. Universities have to spend a fair amount of money to build and maintain online course systems.

1.1.2 Existing online course platforms.

There have been a lot of comprehensive online course delivery platforms. Widely used commercial products include WebCT [11, 13], BlackBoard [22] and eCollege [8]. There are also a number of open-sourced systems. One of the most popular open-source platforms is Moodle [4].

1.2 Virtual laboratories for online science courses.

Science courses are important in education. However, some theories and equations are very abstract and thus difficult to understand. Therefore, students consider them boring while instructors sometimes have no interesting ways to teach them. However, a study showed that online science courses, especially those with laboratory experience, if properly designed and implemented, could make students more interested and active in science courses [31].

1.2.1 The importance of laboratories in science courses.

Instructors of science courses have long recognized the need for laboratory experience, through which students can deepen their understanding of the conceptual material [18]. It is therefore necessary and helpful to give students laboratory or experiment experience when they take science courses online.

However, existing online course systems mainly rely on features like web pages, emails, multimedia files (audio and video), and forums to deliver the course materials. Little has been done to replicate actual science experiments in online course systems due to the technical difficulty [26].

1.2.2 What is a virtual laboratory?

Because there are many definitions in literature, it is difficult to give “virtual laboratory” a decisive definition. It is described as “a computer simulation which enables essential functions of laboratory experiments to be carried out on a computer” according to [14]. Another definition is “a virtual reality environment that simulates the real world for the purpose of discovery learning” [34].

1.2.3 The benefits of using virtual laboratories.

As we described in 1.2.1, it is important to provide laboratory experience to students who take online science courses. Besides all the advantages it shares with online courses, virtual laboratory gives students the chance to apply what they learned into experiments so that they can strengthen their knowledge. It could facilitate a range of different learning processes, such as solution of complex problems, discovery of new content and new assessment of already known information by discovery learning, construction of general principles from experimental work, and comparison of individual phenomena (inductive learning) [14]. Research [7] has found that virtual laboratory in online science courses helped students to better understand abstract concepts and theory.

Additionally, virtual laboratory has some advantages over a traditional lab. It gives instructors and students increased access to laboratory resources, especially during experiments that require expensive or fragile equipment or that present safety hazards. Virtual laboratory allows instructors to offer more interesting experiments. At the same time, students are free to explore all the possibilities without fear of damage or danger [26].

On the other hand, it is also important to mention that experiments in virtual laboratory are not for everyone and cannot completely replace traditional physical experiments [7, 10].

1.3 Related works in online courses laboratories.

Research has been conducted in order to introduce laboratory and experiments into online science courses. We found that the previous work falls mainly into the following categories:

1.3.1 Simulations implemented in APL.

A simulation of stochastic process was developed in [39]. It could be used in a great number of physics and mathematics courses. Though this simulation could successfully simulate a lot of phenomena in its subject, it is written in APL, an array programming language. However, the small user base of APL in mathematics and finance engineering make this approach unpopular among virtual laboratory researchers. Besides, APL simulations are designed for offline purpose, which means additional work has to be done to integrate it into online science courses.

1.3.2 Multimedia based online laboratories.

As broadband internet connection becomes popular, it is easier for students to get access to online audio and video streams. Research has been done to make use of online real-time multimedia technology for experiments in online science courses.

JETS (Java-Enabled Telecollaboration System) was developed by [33]. Instructors and students can share educational resources such as whiteboard, 3-D viewing in a real-time way. They can also interact with each other via audio and video. Its advantages in sharing multimedia information and real-time interaction make it possible for instructors to conduct experiments in a real laboratory and show the results to students who are in front of computers with high-speed Internet connection. Nevertheless, its synchronous nature requires instructors and students to be online at the same time, which sacrifices the flexibility of “learn at any time” of online courses.

Asynchronous multimedia teaching, when compared with its synchronous counterparts, does not require instructors and students to be online at the same time. It is thus more welcomed by students because it adds usable hours to the student’s day. NEW (Network Education Ware) [29] provides both synchronous and asynchronous multimedia teaching. Another example is the online video physics laboratory for high school students developed by [30].

Though real experiments are shown to students instead of simulation in an interactive way, the multimedia approach has one major disadvantage when being used for laboratory experiments. All the experiments are conducted and controlled by instructors while students watch the demonstration most of the time. Even though students can also interact with instructors in order to observe experiments conducted in the way they are interested in, it is still an observation process rather than hands-on experience, from which students would benefit more.

1.3.3 Online hybrid laboratories.

These laboratories and experiments are special in that they are based on real experiment rather than virtual unreal simulations though they are placed online. They are the hybrid of “physical” experiments and web-based simulation.

DropBall [24] is such an online physics experiments for Newtonian mechanics. The major components of DropBall include mechanical apparatus, force sensors, electronic control device, turntable, and web interface.

Students interested in the experiment can visit the DropBall website (<http://dropball.cs.emporia.edu/>) and request an experiment to be conducted with customized parameters such as drop height and ball type. The system gets requests from the web server and automatically conducts the experiment by invoking real experiment instruments in a real laboratory in Emporia State University. For example, the turntable will choose the ball chosen by the user and the electromagnetic lifting apparatus will lift the ball to the specified height and drop it.

Students can observe the experiment process via real-time online video. What is more interesting is that the system could acquire data, such as velocity and acceleration, from multiple sensors, which are located in the laboratory, during the experiment and conduct data analysis. Data observed by sensors, statistical data generated by DropBall’s analysis system, and graph are presented in students’ web browsers. Students could also choose to have these data sent to their email boxes so that they can save them for further study.

Another similar system is the Remote Dynamical Systems Laboratory at Stevens Institute of Technology (online at <http://dynamics.soe.stevens-tech.edu>).

There is another variation in the category of hybrid laboratory. The ideas are similar to DropBall. Real experiments are also conducted with real instruments in real laboratories. The difference is that instead of displaying the experiment process with online video, they use virtual reality technology to show the ongoing experiment process as animations in web browsers. They are therefore more accessible for students whose Internet bandwidth does not support real-time video streams. Examples include Interactive Chaotic Pendulum at Mercer University [20] (online at <http://physics.mercer.edu/pendulum>) and FVRLE at Chuang Yuan Christian University [40].

The hybrid approach is appealing in that students can still get firsthand experiment data without having to go to a real laboratory and conduct real experiments. As with most student experimentation, actually generating and even saving the data provides students with ownership of their data and analysis, which transcends simply analyzing canned or textbook data. All the four foregoing systems are now used as part of statistics or introductory physics courses, and students' response to using this hybrid approach has been favorable.

However, though little human interference or assistance is needed, the hybrid approach suffers from limited laboratory resources, just like a traditional laboratory. Right now, DropBall has only one set of experiment instruments, which means it could satisfy users' requests only one by one. On its website, there is a queue for those who want to conduct this experiment. For example, if student A's experiment is running in the laboratory, student B can only watch A's experiment and wait till A's experiment is over

before B's experiment starts. This is very inconvenient if a group of students in one class are asked to use this system.

Besides, such a hybrid system also requires a hybrid of technology. The development will involve researchers from mechanics, automation control, signal sensing and processing, and web programming, which will raise the cost for development and maintenance.

1.3.4 Pre-developed online virtual laboratories.

Systems in this category present purely virtual laboratory experiments. There are no real instruments or laboratories behind them. They rely mainly on modeling of knowledge in specific science subjects and programming techniques. Though some of them are very powerful and can be referenced by instructors when teaching online courses, they share one major disadvantage: instructors and students have the same access privilege to those systems, which means instructors cannot modify the experiments according to their special needs, unless they are the original developer. Even if given the source code and resource to replicate those systems, instructors are required to be proficient programmers of various programming languages or competent users of certain complicated software packages in order to create, customize, and deploy their own experiments.

Virtual Chemistry, as its name implies, is an online virtual chemistry laboratory [35] developed by the Department of Chemistry at Oxford University. It provides a lot of information about chemistry experiments, ranging from experiment manuals and

experiment designs to result analysis. The most notable and interesting feature is “LiveChem,” the Macromedia Flash-based virtual laboratory. “LiveChem” provides a platform to show the visual results of chemical reactions when mixing different salts and reagents. All the chemicals have photos, showing their color and state. Students are first asked to select one from thirteen kinds of salts and one reagent out of twenty-two options. After the “Play” button is pressed, the virtual experiment will start. A short Flash animation, which comes from the video of the actual reaction, will appear in the web browser, showing how the reaction happens and the visual result. This virtual chemistry laboratory is vivid and easy to use.

Students are also given a great number of options to explore and observe different reactions. Detailed analysis about reactions is also accessible in order to help students better understand what they have seen on the screen. At the same time, the limits of the system are also obvious. The system is designed in an ad-hoc way without underlying models of chemistry knowledge. One video clip or animation is specific for one reaction only. As a result, if a new salt is to be added into the laboratory, developers have to conduct the real experiments, in which the new salt is mixed with each of the existing twenty-two reagents in the laboratory, videotape the visual effects of those reactions, convert video files into Flash, and integrate those animations into the system.

VRPS [15] is a real-time virtual laboratory providing interactive experiments in physics courses, such as electric current testing, spread of water waves and a simple oscillator. The experiments are shown as 3D animations in web browsers. It is created with 3D Webmaster, a software toolkit for creating 3D web pages, and Supercap

Control Language (SCL), a script language used to control the action of objects in a virtual reality environment.

It was claimed that students' performance in learning was enhanced by using VRPS. But the usage of the system is limited. First, users have to be very familiar with virtual reality technology and must be able to use 3D Webmaster and to program with SCL. Second, similar to "LiveChem," there is no model or data structure behind those virtual experiments. Experiments are designed in an ad-hoc way. The reusability of the existing experiments is thus limited. New experiments have to be created almost from scratch. Besides, since there is no library for experiment widgets or instruments, one has to design all the visual models, including 3D models, for experiment apparatus one by one and spend a lot of time on the layout of apparatus.

VPL [3] was developed by the Distance Education and Learning Telecommunication Applications (DELTA) group at the German National Research Centre for Information Technology and is a modularized online virtual laboratory for physics and astronomy experiments. It consists of three modules: (1) Simulation: the experiments are modeled as computer programs, which are implemented in general purpose programming language like C++ or Java. This module is not directly accessible online. (2) Visualization: this is the module that supports the user interface and displays experiments to students. It is based on Java, Java-3D and Virtual Reality Modeling Language (VRML). (3) Translation and Interface: this module links Simulation and Visualization. Graphic user interface is dynamically activated using Common Gateway Interface (CGI) scripts.

The VPL system works in the following process: when students visit the system, they are asked to select the experiment and input parameters from the user interface. The Translation and Interface module then sends those customized requests to the Simulation module, which will simulate the experiment according to requests. Real-time outputs from the Simulation are sent to the Translation and Interface module, where those outputs are translated for the Visualization module. With the translated data, the Visualization module could virtually rebuild the experiment process and present it in web browsers.

The modular design of VPL benefits from the Simulation module, which could model very complex experiments with the power of general programming language. It is also possible to reuse some existing simulations by adding interface to the Translation and Interface module. The Visualization module could also vividly represent ongoing experiments in 3D animations. Example experiments created with VPL include Geometric Optics, Hydrodynamics Entropy and Virtual Astronomy Solar System, and Kepler's Law.

On the other hand, though powerful and vivid, the complexity of each individual module adds to the complexity of VPL, which make it very difficult to implement and maintain. The cost of such a system will thus be higher than other virtual laboratories.

BioLab [5, 16, 27] is an online virtual biology laboratory developed at Eastern Michigan University. It aims at helping students from non-biology major to learn fundamental biology. Professors from both Computer Science and Biology departments, along with graduate students, built a virtual biology laboratory with Java Applet and designed a few experiments. For example, the enzyme activity experiment is used to

examine the effects of temperature and pH on the activity of the enzyme salivary amylase. The cellular respiration experiment can help students to understand how activity and temperature affect respiration rate. BioLab has been used for online biology courses at Eastern Michigan University. Though all the experiments in BioLab are hard coded, it was the development of BioLab that inspired the idea of CyberLab.

ViBE [34] is a virtual laboratory for biology experiments. The main characteristic of the ViBE is the three-tier architecture, which comprises the vertical tiers of presentation, domain logic, and storage. The modular Model-View-Controller (MVC) architecture is similar to the modular structure of VPL. The ViBE's presentation tier is virtually free of the application logic and deals with visualizing the experiments and accepting user inputs. The domain tier contains all the model objects and deals with the semantics of experiments and rules, as well as abstract data representation. Data about experiments are exchanged in eXtensible Markup Language (XML). Most of the system components, including all the virtual labs, are implemented as Java Beans.

ViBE can be extended to implement different laboratories, thanks to its modular design and the resulting separation of concerns and tasks. It supports augmentation by animation effects and realistic renderings of virtual objects. The software framework is lightweight and can be downloaded as an applet in a browser. Students can also save their lab reports in XML and review or edit them later.

A few interesting experiments have been created with ViBE, including Spectrophotometry, Cell Mitosis and Meiosis, and Virtual Microscope. The way those

experiments are created is relatively straightforward but one still has to be able to program with XML and Java Beans.

1.3.5 Online virtual laboratory creation tool.

Physlets [6], also known as Physics Applets, are small flexible non-commercial Java applets designed for virtual experiments in online science education (online at <http://webphysics.davidson.edu/Applets/Applets.html>). According to our research, it is the only tool that aims at allowing instructors to easily create their own science experiments. It is based on Java and provides a set of APIs (Application Programming Interface), which could be easily used in Java Script to construct an experiment. It is designed specially for online courses, which means they can run on (almost) any platform and be embedded in almost any type of HTML document. It is claimed that it can be used for almost any subject in mechanics and almost any topic in electrostatics with small changes in the Java Script associated with each experiment. Data taking and data analysis can be added using inter-applet communication.

With tutorial and well supported documentation, it may not be difficult for a competent Java Script programmer to create an experiment. However, though Java Script is not a complicated language for computer science majors, its user population is still limited, which limits the usage of Physlets among instructors of online science courses.

1.4 Brief introduction to CyberLab

1.4.1 Purpose of CyberLab.

After an extensive literature review, we found that existing online virtual laboratories are either pre-designed for specific experiments or require programming skills for instructors to create experiments.

After the development of BioLab (described in 1.3.4), the idea of CyberLab was proposed by [10] as a set of tools for virtual online laboratories. It aims at helping instructors with little programming background to create their own interactive experiments for online science courses and giving students a convenient online access to those experiments and an easy way to execute them.

1.4.2 Structure of CyberLab.

CyberLab consists of two main components: LabCreator and LabExecutor (Figure 1). Instructors of science courses will use LabCreator to design what the laboratory experiments will look like and how they will function. LabCreator then generates a file that contains the definition and description of the experiment, which is called an “experiment descriptor.” This descriptor file is then stored on the instructor’s web server of online courses.

LabExecutor is used by students to conduct experiments. The LabExecutor is a Java Applet, which can be executed in most web browsers. To run the online experiment, students will visit the instructor’s website and invoke the LabExecutor. LabExecutor then downloads the experiment descriptor file from the instructor’s web server, uses it to

determine the construction and design of the experiment, displays the experiment in web browsers, and allows students to conduct the experiment.

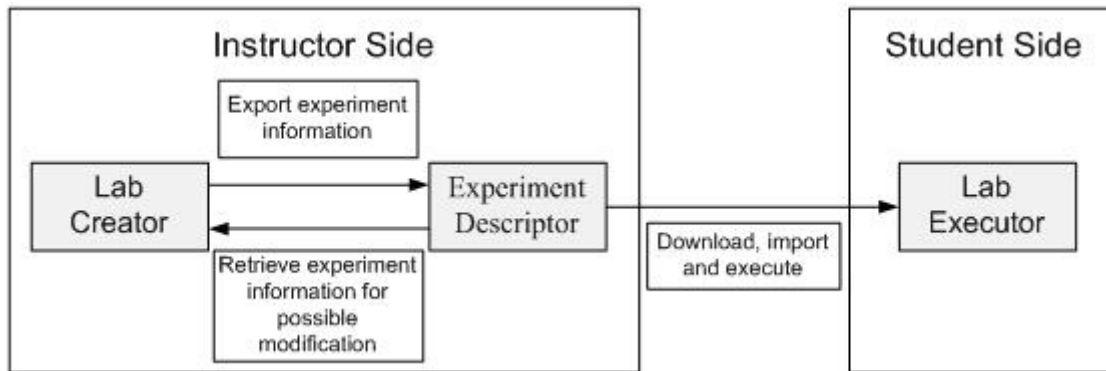


Figure 1 Structure of CyberLab

1.4.3 Model of experiments in CyberLab.

In CyberLab, each experiment is represented as Finite State Automata (FSA). FSA [36] is a model of behavior composed of states, transitions, and actions. A state stores information about the past; i.e., it reflects the input changes from the system's start to the present moment. A transition indicates a state change and is described by a condition that would need to be fulfilled to enable the transition. An action is a description of an activity that is to be performed at a given moment.

There are two types of FSA: Deterministic and Nondeterministic. In CyberLab, we are using the Deterministic FSA, in which there is a deterministic next state given a pair of current state and input. In this paper, when we mention FSA, we mean Deterministic FSA.

1.4.4 Previous work in CyberLab.

Dr. Evett and a few graduate students at Eastern Michigan University have been working on CyberLab since 2003 and have made some progress.

Research on the data representation of the intermediate file--experiment descriptor was conducted [32]. Because experiments in CyberLab consist of a collection of Java objects, the major topic in this research was the best way to import Java objects from LabCreator and export them to intermediate files. Three approaches are considered as candidates: XML processing via Document Object Model (DOM), Java Serialization, and Java XMLEncoder/Decoder. After comprehensive comparison, Java Serialization is selected for its advantages of less restriction to the object class definition, simple object processing programming with standard Java APIs, secure intermediate file format, and less complexity in CyberLab system implementation.

A prototype for LabExecutor was developed [23]. Since the descriptor file was not available at that time, another class called the Driver, which simulates the role of descriptor and feeds LabExecutor with experiment information, was created for testing purpose. Since the Driver has to be coded for each individual experiment, the LabExecutor was tested with only two experiments generated by the Driver. The first experiment implements two buttons as toggle buttons, and only one of them is active at any time. The second experiment simulates a heating experiment, where temperature can be incremented and decremented. The temperature in the heating experiment is limited by a minimum and maximum value, and current temperature is displayed.

The LabExecutor prototype showed that the ideas of using FSA to model experiments and using Java Applet to display experiments are feasible. However, it is not

tested with descriptor and thus lacks the feature of loading descriptor, which left some work to the integration of CyberLab.

1.4.5 My work in this thesis.

The thesis mainly deals with the implementation of LabCreator in CyberLab, which did not exist yet, and its integration into the existing components of CyberLab.

LabCreator has four major functions: design experiment layout, deploy underlying model, associate model with experiment components, and export descriptor.

Layout tools are designed to be used in much the same way as the form layout tools in Integrated Development Environments (IDEs). LabCreator will support the “Drag and Drop” feature, which allows instructors to select experiment widgets and components from a palette, place them on the screen, and specify how users can interact with them. When a student is running an experiment, the interface will look almost exactly like what has been designed within the layout tools. Therefore, layout tools are said to be “what you see is what you get” (WYSIWYG) and free instructors from having to write source code themselves. To realize such functions, various experiment components must be developed, such as containers (flasks, treadmills, etc.), meters (thermometer, pHtester, etc.), other devices (lamps, agitators, etc.), and control components (menus, buttons, etc.).

Underlying models of experiments consist of variables (dependent or independent), expressions, and FSA. Models specify how the experiment will behave at a certain point. For example, a variable may represent the density or temperature of a fluid within a flask, or the amount of time elapsed since the start of the experiment.

Visualization deals with the association between variables and experiment components. It is necessary to specify a correlation between the various visual elements of the experiment (e.g. the color of fluid within a beaker or the brightness of a burner) with the variables of the underlying model so that users can observe changes in the ongoing experiment.

The exporting of descriptor happens after instructors finish the design of the experiment. LabCreator can export all the necessary information about the experiment into “experiment descriptor” with proper data format, which will be available online in the course server.

The integration of LabCreator into CyberLab will focus on how the descriptor is loaded and how the original experiment is re-constructed in LabExecutor.

implemented, tabular views of widgets for different science subjects are provided for the convenience of future development.

The Experiment Workbench is the canvas where the layout of the experiment is designed. After users select experiment widgets and specify their initial attributes, those widgets will appear on Experiment Workbench. The users are then free to move the widgets to desired positions (the “Drag and Drop” feature will be discussed later in this paper), delete them, or modify their attributes. The layout of the experiment executed in LabExecutor will look the same as the one designed on the Experiment Workbench.

The Experiment Information panel and FSA Edge Panel are used to show the information about the experiment being created so that users have a good idea about what has been done and what remains to be done.

Experiment Information is the rightmost vertical panel. It consists of two tabular panels. The Variable panel shows the name, type, and description of an experiment’s variables. The State panel shows the names of FSA states and the widgets associated with each.

The FSA Edge panel is located under the Experiment Workbench. It shows the information about each edge of the experiment’s FSA, including its starting state, ending state, trigger widget, transition condition, and consequence. This panel is separated from Experiment Information panel because FSA edges usually have more information to display than variables and states, which requires more and wider columns in the table. It is therefore difficult to display them in the rightmost vertical panel. Besides, the FSA edge panel will display equations for transition conditions and consequences. Though not implemented now, it may be helpful to add an “equation editor,” which will help users to

construct an equation and prevent typos. In this case, having a stand-alone FSA Edge panel will leave more space and flexibility for future development of more features.

2.2 Experiment widgets.

All the experiment widgets in CyberLab are implemented via the interface “Widget.” An excerpt of the Java code for widget-related classes can be found in Appendix A.

So far, only two types of widgets have been implemented: CyberButton and CyberTextField.

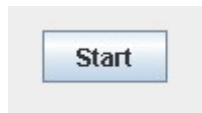


Figure 3. The appearance of a CyberButton

CyberButtons are used as control buttons in experiments and make use of Java’s JButton class as the base class. The appearance of CyberButton on the Experiment Workbench is shown in Figure 3. The text “Start” on the button is specified by the label attribute. The CyberButton class is defined as:

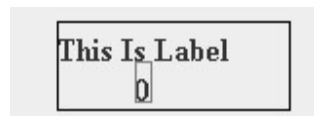


Figure 4. The appearance of CyberTextField

CyberTextField is a text display of dynamic numeric values (Figure 4). The label of the widget is displayed on the upper part of the widget, while the numerical value of

the associated variable is displayed under the label. Because there is no similar GUI component in Java, we had to extend the JComponent class and overwrite the “paintComponent” method to draw the rectangles and texts.

2.3 What You See Is What You Get (WYSIWYG).

WYSIWYG (pronounced “wiz-ee-wig”) is used to describe a system in which content during editing appears very similar to the final product. It is commonly used for word processors (e.g. Microsoft Word) and Web (HTML) authoring (e.g. Macromedia Dreamweaver). LabCreator is designed to be a WYSIWYG experiment editor.

LabCreator users, mostly instructors, could determine the appearance of the experiment, such as the layout and the label of experiment components; and students, when executing the experiment, will get the same appearance as that designed in LabCreator. Just like Dreamweaver, which does not require its users to know HTML, LabCreator enables those who do not know programming to create virtual experiments in an intuitive and easy way.

2.3.1 Drag and Drop.

An important feature in WYSIWYG experiment editor is the “Drag and Drop” of widgets when positioning them. When creating a widget, users can specify its initial position on Experiment Workbench. After that, users can select the widget (by pressing the left button of the mouse), drag it to anyplace on the Experiment Workbench canvas (hold down the left button while moving the mouse), and drop it there (by releasing the left button).

Widget “Drag and Drop” is implemented with the help of “MouseAdapter”—the adapter interface class for receiving mouse events. Each widget added into the

Experiment Workbench will have an object implemented from MouseAdapter as its action listener. The “mousePressed” and “mouseReleased” methods of MouseAdapter are implemented to capture the user’s action on the mouse. The following is an excerpt of the code for the implementation of the “Drag and Drop” for CyberButtons.

The Java code excerpt for widgets drag and drop is shown in Appendix B. In the code excerpt, the “MouseMoved” method determines whether the widget has been moved to a new position. In the “moveWidget” method, the position information is stored in the widget object, which will allow the widget to be positioned correctly both in the Experiment Workbench and by the LabExecutor.

2.3.2 Appearance of widgets.

As described in 2.2, every widget will have a label that can be customized, either during or after creation. For example, when a new control button (i.e. CyberButton) is to be added, a dialog box (Figure 5) appears that allows the author to specify an initial position as well as its label. To change an existing button’s label, users can right-click the widget and select “Label” from the pop-up menu (Figure 6). The same menu also provides for the deletion of a widget.



Figure 5. Dialog for CyberButton initial settings

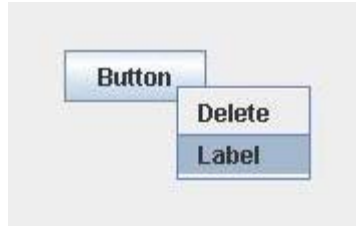


Figure 6. Right-click pop-up menu for CyberButton

2.4 Data structure of CyberLab virtual experiments.

Each virtual experiment has two parts: the GUI and an underlying model that represents the state of the experiment.

The GUI part is relatively straightforward. Originally designed in the LabCreator's Experiment Workbench, the GUI is represented by a list of widgets and a frame ("ExperimentFrame" class in CyberLab), where the widgets are displayed.

The underlying model is a collection of FSA, independent and dependent variables, and the set of equations that define the relationships among them.

Execution of an experiment causes the FSA to move from state to state via edges. Each state has a widget list indicating which widgets are active in this state, and a table for possible transitions starting from this state. Once an edge is created, its information is stored in the transition table of its starting state. The information in the transition table includes the name of the edge, required interaction with a particular widget from the experiment user (e.g. press a button) for the traversal of the edge, what is necessary in order to move from one state to another (i.e. the condition for the transition), and certain manipulation of the model (i.e. the consequence of the transition) as the result of the edge traversal. The table also provides the next state the FSA has to go to as a result of the transition.

In the aforementioned classes related to FSA, transition conditions are Boolean expressions (e.g. $\text{temperature} > 1$, $\text{speed} == 40$) and consequence are mostly mathematical expressions (e.g. $\text{speed} = \text{original_speed} + 10$). Expressions consist of experiment variables (e.g. temperature of liquid, speed of falling ball), operators, and numbers. There are two types of experiment variables: dependent and independent. A dependent variable is associated with a function consisting of various operators and references to other independent or dependent variables (e.g. $\text{speed} = \text{speed} + \text{acceleration} * \text{time}$). Each independent variable has an associated initial value. Its value is independent of other variables (e.g. $\text{temperature} = \text{temperature} + 1$). Variables and their values are stored in the model.

In order to determine the variable value and whether a transition condition holds, it is necessary to design a mechanism to evaluate expressions (both Boolean and mathematical expressions) and variable values.

For a mathematical expression, the evaluation process will first parse the right hand side of the expression. If the right hand side contains a variable, the value of the variable will be retrieved from the model. The evaluated value of the right hand side will be given to the left hand side variable, and the new value of the left hand side is stored back in the model.

Boolean expressions are different from mathematical expressions. The left hand side of a Boolean expression may be a combination of variables, operators, and numbers, while the left hand side of a mathematical expression usually has only one variable. Another difference is that relational operators are used to link left hand side and right hand side. As a result, both left and right hand sides of Boolean expressions are evaluated,

and their values are compared based on the relational operator. A Boolean expression is evaluated to integers. Zero represents False, while other values are treated as True. No variable's value is going to be changed after the evaluation of a Boolean expression.

2.5 An example of how a virtual experiment works.

As for implementation, each experiment in CyberLab will correspond to one "Experiment" object, which contains a list of widgets, one "Model" object and one "FSA" object. The numbers of "FSASState," "FSAEdge," "Expression," and "ExpressionNode" objects in an experiment may vary.

The following example should better illustrate how a virtual experiment works.

When an experiment starts, the "Experiment" object will ask the "FSA" object to check the current state. Active widgets associated with the current state are then displayed on "ExperimentFrame."

When a student manipulates a widget in the experiment (via LabExecutor), the "FSA" object is responsible for determining which widget is manipulated and hand over the action to current "FSASState," which will then look into its transition table. If the manipulated widget is found in the table, the associated condition expression is evaluated. If the condition holds, the corresponding transition (i.e. FSAEdge) will be traversed. Whether FSA moves to a new state with the traversal depends on the ending state of the corresponding "FSAEdge." As part of the transition, the consequence of the transition (usually a mathematical expression) is also executed. As a result, the value of the left hand side variable of the consequence expression is evaluated and updated in "Model" object.

After the transition, the “ExperimentFrame” is redrawn so that the current state of the experiment is shown to the student.

This process is carried out until a termination state is reached, if there is one. If there is no termination state, the experiment executes until the student chooses to close the experiment.

Another possible situation for an experiment with animations is that there will be a trigger to start the animation. At the end of the animation, the experiment won't terminate but will usually sit there, possibly allowing the student to provide new execution parameters or to restart the animation.

2.6 Create virtual experiments with LabCreator.

This chapter will demonstrate how to create a virtual experiment in LabCreator. We will use a simple heating experiment as the example, which will be used to test LabExecutor later.

2.6.1 The simple heating experiment.

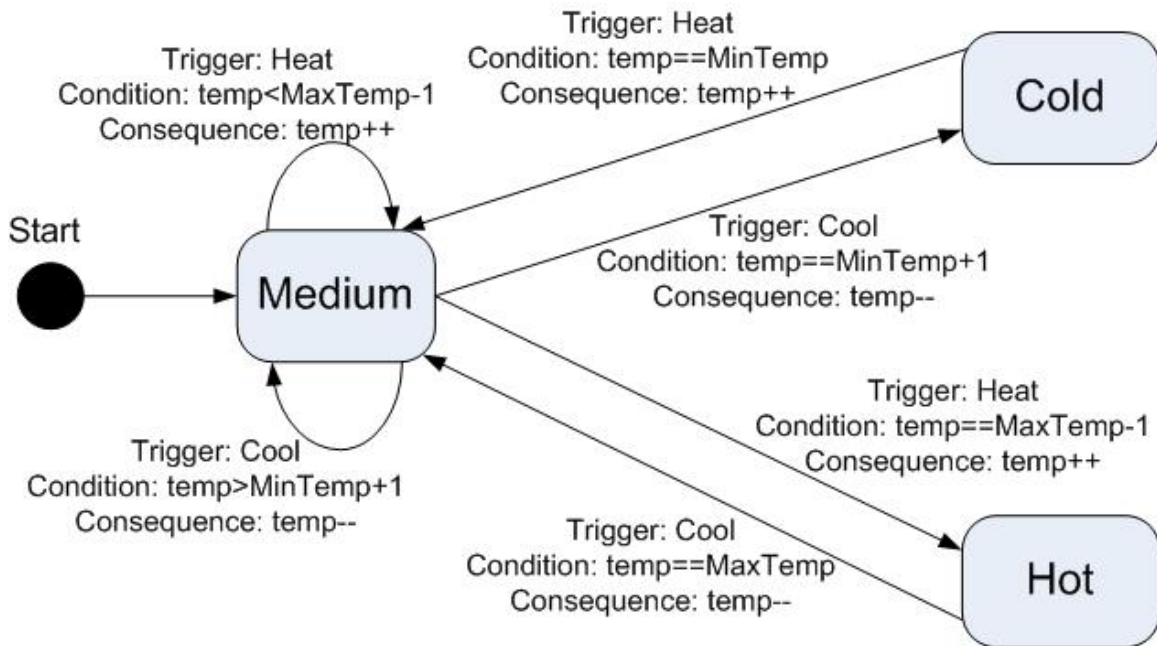


Figure 7. FSA diagram of the heating experiment

This experiment simulates how the user’s decision on heating or cooling the air changes the temperature. The temperature is represented by the independent variable “temp.” It has maximum and minimum values, which are represented as MaxTemp and MinTemp, respectively.

Three widgets are involved in this experiment: a thermometer (represented by a text box labeled as “Temperature”), a heating element (represented by a button labeled as “Heat”) and a cooling element (represented by a button labeled as “Cool”). Though the thermometer does not directly interact with users, manipulation of the experiment can be obtained by pressing either of the two buttons.

There are three states: (1) Medium, which is the initial state; (2) Cold, which has the lowest temperature; and (3) Hot, which has the highest temperature. There is no termination state in this experiment.

In state “Medium,” temperature is neither too high nor too low. Users are thus free to raise or lower the temperature. When the heating element is used, the temperature increases by one degree. When the cooling element is used, the temperature decreases by one. After the temperature drops to its minimum value (state “Cold”), the cooling element is not active anymore. Users can use only the heating element. Similarly, when the temperature reaches its peak (state “Hot”), only the cooling element is available for users’ manipulation.

The detailed FSA of the experiment is shown in Figure 7.

2.6.2 The procedure to create the heating experiment.

First, we will design the experiment GUI by selecting widgets from “Generic” tab of the Experiment Widget panel and positioning them on the Experiment Workbench. In this experiment, there are two Control Buttons: “Heat” and “Cool.” There is also one Text Display: “Temperature” (Figure 8). Note that “Temperature” is set to zero as the text display widget is not associated with any variable.

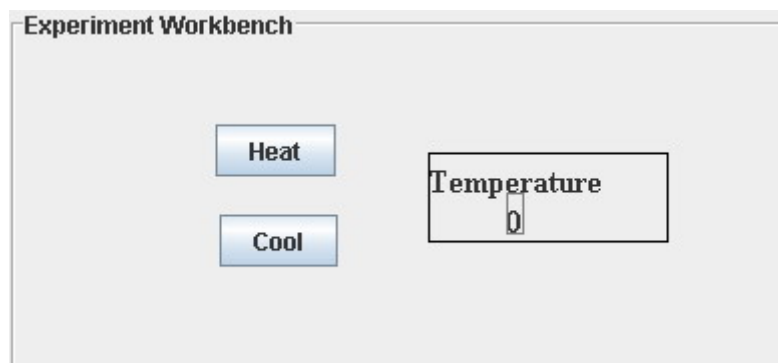


Figure 8. GUI of the heating experiment

Next, we focus on constructing FSA for the experiment. Basically, this is done by going through the menu items in “FSA” menu (Figure 9).



Figure 9. FSA Menu of LabCreator.

First, we create the independent variable “temp,” which stands for the temperature (Figure 10). Remember that the value of the variable (i.e. the temperature) is to be displayed in Text Display widget “Temperature.” Therefore, we have to associate the variable with the Text Display widget by accessing the right-click pop-up menu item “Expression” of widget “Temperature” (Figure 11) and typing the variable name “temp” in the following dialog. After this step, the widget “Temperature” displays the value of variable “temp,” which is 0.0 at the very beginning.

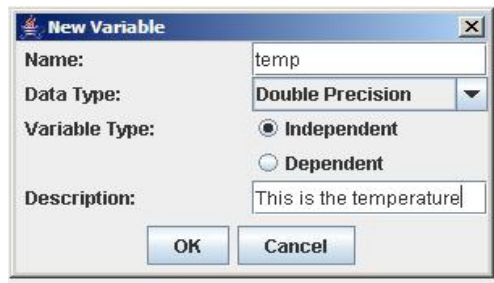


Figure 10. Dialog for creating new variable “temp.”

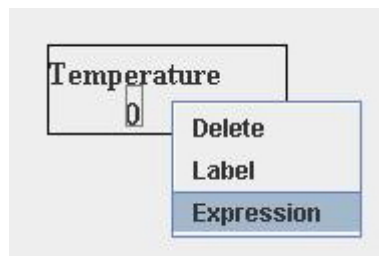


Figure 11. The right-click pop-up menu for Text Display widget.

Second, three states are created by typing names and selecting active widgets.

Figure 12 shows how state “Medium” is specified.



Figure 12. Dialog for creating new state “Medium.”

After the three states are created, we have to specify the FSA starting state from a scroll-down menu (Figure 13).



Figure 13. Set Starting State as “Medium.”

The last step in FSA construction is to create FSA edges. This process is a bit more involved than the aforementioned steps. Users have to assign a Boolean expression as condition and a mathematical expression as consequence. For this experiment, six edges are needed for the FSA. Figure 14 shows the dialog for the edge, which starts from and ends in state “Medium” and causes the temperature to rise.

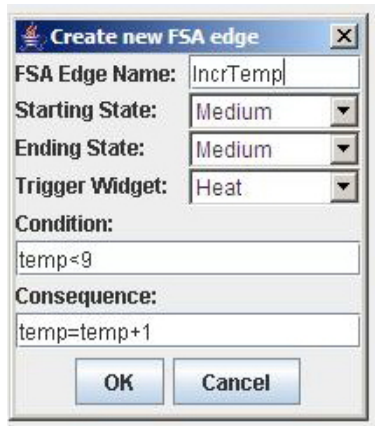


Figure 14. Dialog for creating a FSA edge

Note that in this demonstration, the temperature limits--MaxTemp and MinTemp--are set as 10 and -10 respectively.

During the construction of the FSA, the Experiment Information panel and FSA Edges panel in LabCreator will display updated information about the FSA being constructed. Figure 15 is the LabCreator GUI after the experiment creation process is completed.

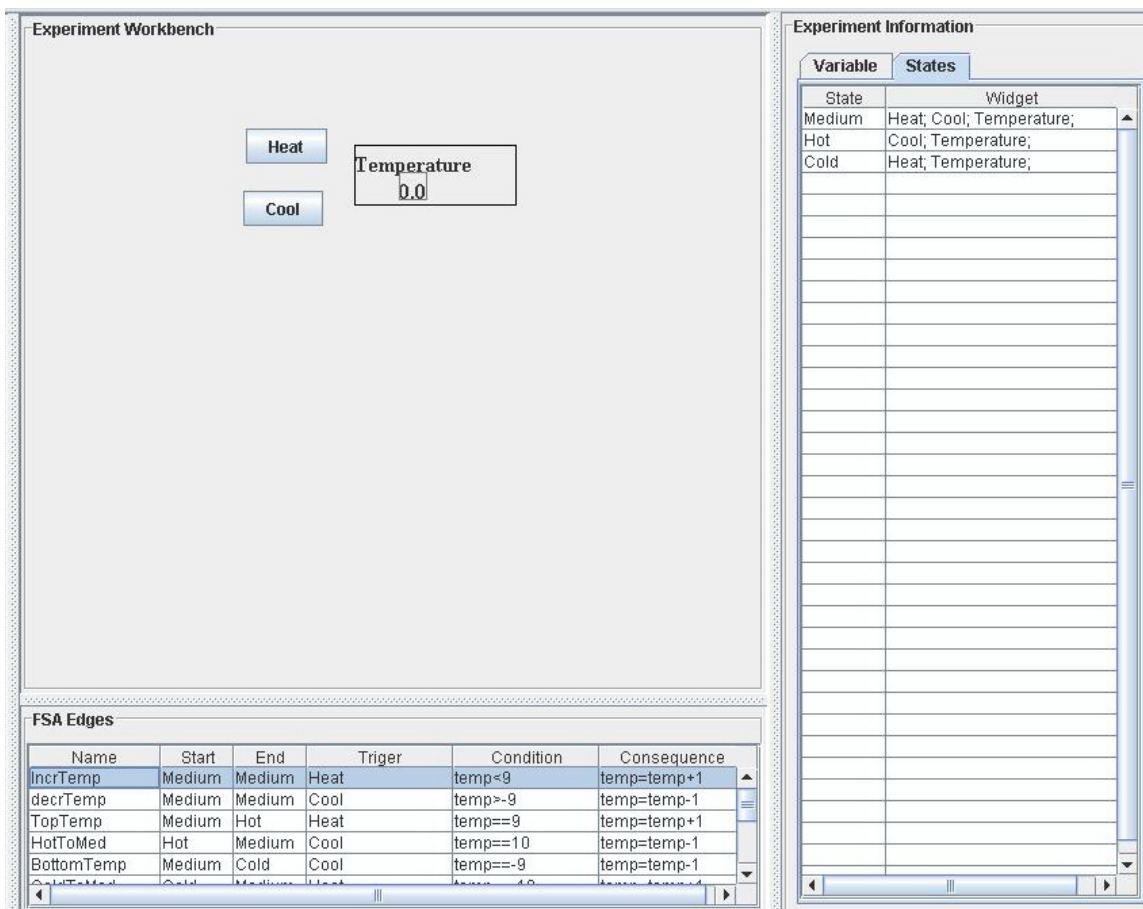


Figure 15. GUI of LabCreator after the heating experiment is created.

2.7 Preview experiments in LabCreator.

After creating an experiment, instructors may want to review and test the experiment before giving it to students. The experiment preview feature in the LabCreator allows instructors to check how the experiment will look and function in the LabExecutor.

When the preview is invoked, a new window will appear. The preview window looks quite similar to the window of LabExecutor because the preview is designed to mimic LabExecutor. Instructors can manipulate widgets and observe the experiment, just as students could do when using LabExecutor. If instructors found something wrong or

are not satisfied with the appearance of the experiment, they can stop the preview and re-design the experiment in LabCreator.

In terms of implementation, an inner class “PreviewExpAL” is defined to handle the action on the “Preview” menu item. Two objects are important in the preview of a CyberLab experiment: the “Experiment” object, which corresponds to the experiment (discussed in section 2.5), and an “ExperimentFrame” object.

First, LabCreator will check the experiment to see whether there is anything missing. For example, if starting state is not specified or there is no active widget in the starting state, a warning dialog will pop up and the preview will not start.

Then all widgets of the experiment are loaded into the “Experiment” object. The “ExperimentFrame” object is then created. It loads the “Experiment” object and displays widgets of the starting state in a new window, where instructors can manipulate the experiment. Manipulation of the buttons will be handled by the “FSA” object, which is part of the “Experiment” object. Figures 16 to 18 show the preview of the heating experiment. It works properly as designed in LabCreator.

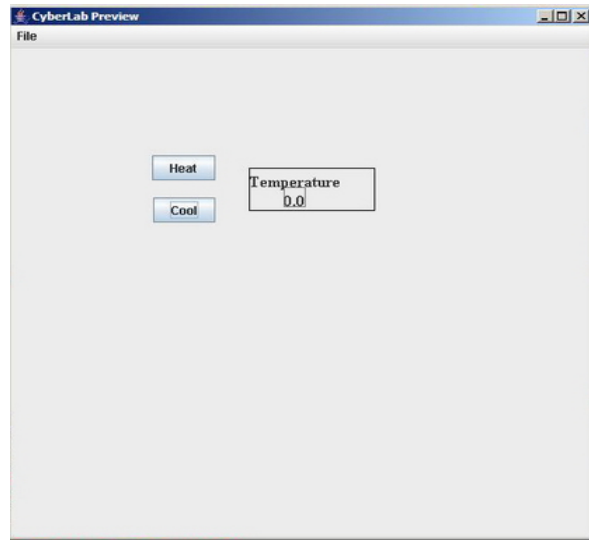


Figure 16. Preview of heating experiment (initial state “Medium”)

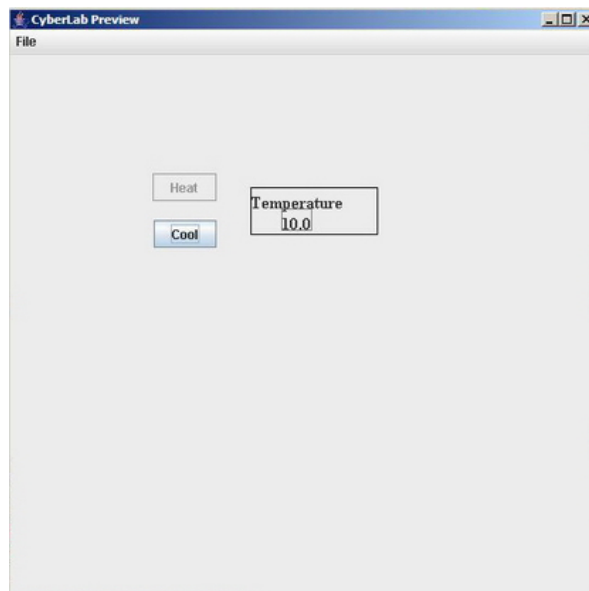


Figure 17. Preview of heating experiment (initial state “Hot”)

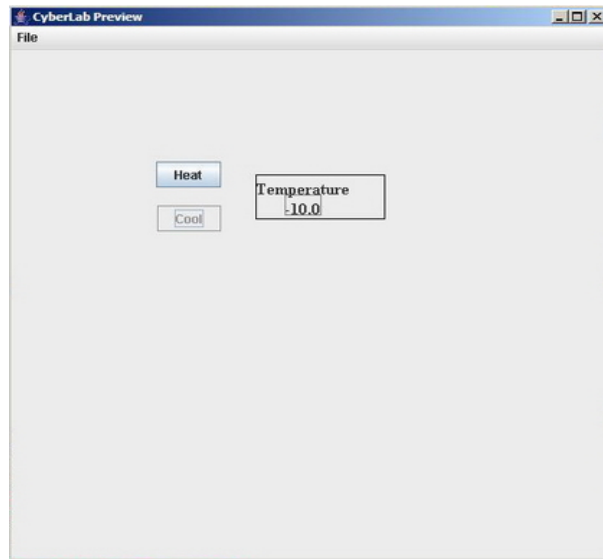


Figure 18. Preview of heating experiment (initial state “Cold”)

CHAPTER 3

INTERGRATION OF LABCREATOR INTO CYBERLAB

Since LabCreator has been successfully implemented, it is now necessary to integrate it with LabExecutor into the CyberLab system. The main tasks include exporting experiments created in LabCreator into intermediate files, loading the files in LabExecutor, and executing the experiment.

After the integration of the CyberLab system, a simple usability test was conducted.

3.1 Java Object Serialization.

As mentioned in section 1.4.4, Java Object Serialization is selected to generate the intermediate file.

Java Object Serialization provides support for objects' input and output, so that a whole object in a Java program can be written to or read from a raw byte stream. This is an important feature for an object-oriented programming language. When dealing with files in order to load or write data, Pascal or C programmers usually face a lot of annoying problems, such as the EOF mark, the EOLN mark and file pointers [17]. However, with the help of object serialization, Java programmers do not have to worry about these problems anymore.

According to [19], Java Object Serialization “allows Java objects and primitives to be encoded into a byte stream suitable for streaming to a transmission medium or storage facility. It also supports the complementary reconstruction of the object graph from the stream without damaging the persistence of objects.”

Therefore, Java Object Serialization is widely used to store objects into files, retrieve objects from files, and transmit objects between Java Virtual Machines.

To serialize an object, it is also necessary to traverse all the objects and primitives that are referenced in the object. According to [17], “since an object’s state usually consists of information stored in internal data, most objects will contain references to other objects, which will need to be preserved if the original object is to regain its state when it is de-serialized.”

As for implementation, serializing an object requires one of the two criteria to be satisfied: the class of the object must either implement the “Serializable” interface or the “Externalizable” interface.

The “Serializable” interface has no methods, which means that it is not necessary to write any additional code in the class. It can be implemented by simply adding the “implements Serializable” clause.

On the other side, to use the “Externalizable” interface, we have to define two methods: “writeExternal” and “readExternal” in the class to be serialized. According to [17], the “Externalizable” interface is often used “when it is necessary to define an object which has complete control over its serialization and re-constitution process.” In other words, this interface could give users the control of “the encoding used to send the information and which fields are serialized.”

In this paper, we use the “Serializable” interface because it already satisfies our needs for serializations, and it is easy to implement. During the serialization, each object that is referenced by the serialized object and is not marked as transient must also be serialized.

Some may ask why Java objects are not serializable by default. There are three primary reasons, according to Wikipedia [37]:

- (1) Not all objects capture useful semantics in a serialized state. For example, a “Thread” object is tied to the state of the current Java Virtual Machine. There is no context in which a de-serialized “Thread” object would maintain useful semantics.
- (2) The serialized state of an object forms part of its class’s compatibility contract. Maintaining compatibility between versions of serializable classes requires additional effort and consideration. Therefore, making a class serializable needs to be deliberate and is not a default condition.
- (3) Serialization allows access to non-transient private members of a class that are not otherwise accessible. Classes containing sensitive information (for example, a password) should not be serializable or externalizable.

When implementing object serialization in CyberLab, two classes-- `ObjectOutputStream` and `ObjectInputStream`-- are used. An `ObjectOutputStream` object is used to serialize primitive data types and Java objects to an output stream. It is used together with a `FileOutputStream` object so that the output stream of an object can be written to a file. On the other hand, `ObjectInputStream` object is used to de-serialize objects from an input stream and restore them. Similar to `ObjectOutputStream`, it is accompanied by a `FileInputStream` object.

3.2 Export experiments from LabCreator.

As discussed in section 2.4, the information of an experiment created in LabCreator is stored in the corresponding “Experiment” object. Therefore, we have to export this object to an intermediate file.

As discussed before, all classes associated with the “Experiment” class will implement the “Serializable” interface. Besides, we found that the parent class of the serialized class must have a constructor with no argument.

To serialize an object, we have to go through the following steps: (1) Produce a valid filename to write to; (2) Open a `FileOutputStream`; (3) Attach an `ObjectOutputStream` to the `FileOutputStream`; (4) Write object data to object stream; and (5) Flush the object stream and close it down. The code that deals with object serializations in LabCreator is shown in Appendix C.

The intermediate files will have “.exp” as their suffix. Figure 19 shows part of the file generated for the heating experiment in hexadecimal format. All Object Serialization files start with the 2-byte “magic number”: AC ED, followed by the version number of the object serialization format, which is currently 00 05. Then, it contains a sequence of objects, in the order that they were saved.

```

00000000h: AC ED 00 05 73 72 00 13 63 79 62 65 72 6C 61 62 ; ..sr..cyberlab
00000010h: 2E 45 78 70 65 72 69 6D 65 6E 74 0A A9 FD 6A D4 ; .Experiment. j?
00000020h: 6A 19 E4 02 00 06 4C 00 05 6D 79 46 53 41 74 00 ; j.?..L..myFSAt.
00000030h: 0E 4C 63 79 62 65 72 6C 61 62 2F 46 53 41 3B 4C ; .Lcyberlab/FSA;L
00000040h: 00 07 6D 79 46 72 61 6D 65 74 00 1A 4C 63 79 62 ; ..myFramet..Lcyb
00000050h: 65 72 6C 61 62 2F 45 78 70 65 72 69 6D 65 6E 74 ; erlab/Experiment
00000060h: 46 72 61 6D 65 3B 4C 00 07 6D 79 4D 6F 64 65 6C ; Frame;L..myModel
00000070h: 74 00 10 4C 63 79 62 65 72 6C 61 62 2F 4D 6F 64 ; t..Lcyberlab/Mod
00000080h: 65 6C 3B 4C 00 07 6D 79 50 61 6E 65 6C 74 00 14 ; el;L..myPanelt..
00000090h: 4C 6A 61 76 61 78 2F 73 77 69 6E 67 2F 4A 50 61 ; Ljavax/swing/JPa
000000a0h: 6E 65 6C 3B 4C 00 09 6D 79 57 69 64 67 65 74 73 ; nel;L..myWidgets
000000b0h: 74 00 15 4C 6A 61 76 61 2F 75 74 69 6C 2F 41 72 ; t..Ljava/util/Ar
000000c0h: 72 61 79 4C 69 73 74 3B 4C 00 0F 77 69 64 67 65 ; rayList;L..widge
000000d0h: 74 50 6F 73 69 74 69 6F 6E 73 71 00 7E 00 05 78 ; tPositionsq...x
000000e0h: 70 73 72 00 0C 63 79 62 65 72 6C 61 62 2E 46 53 ; psr..cyberlab.FS
000000f0h: 41 5E 15 7F 7E 10 DC BB C4 02 00 0A 4C 00 0D 61 ; A^ .~.莒?...L..a
00000100h: 63 74 69 76 65 57 69 64 67 65 74 73 71 00 7E 00 ; ctiveWidgetsq...
00000110h: 05 4C 00 11 63 6F 6D 70 6C 65 74 65 57 69 64 67 ; .L..completeWidg
00000120h: 65 74 53 65 74 71 00 7E 00 05 4C 00 08 63 75 72 ; etSetq...L..cur
00000130h: 53 74 61 74 65 74 00 13 4C 63 79 62 65 72 6C 61 ; Statet..Lcyberla
00000140h: 62 2F 46 53 41 53 74 61 74 65 3B 4C 00 0A 65 64 ; b/FSAState;L..ed
00000150h: 67 65 57 69 64 67 65 74 71 00 7E 00 05 4C 00 05 ; geWidgetq...L..
00000160h: 65 64 67 65 73 71 00 7E 00 05 4C 00 05 6C 69 73 ; edgesq...L..lis
00000170h: 74 73 71 00 7E 00 05 4C 00 0C 6D 79 45 78 70 65 ; tsq...L..myExpe
00000180h: 72 69 6D 65 6E 74 74 00 15 4C 63 79 62 65 72 6C ; rimentt..Lcyberl
00000190h: 61 62 2F 45 78 70 65 72 69 6D 65 6E 74 3B 4C 00 ; ab/Experiment;L.
000001a0h: 04 6E 61 6D 65 74 00 12 4C 6A 61 76 61 2F 6C 61 ; .namet..Ljava/la
000001b0h: 6E 67 2F 53 74 72 69 6E 67 3B 4C 00 0A 73 74 61 ; ng/String;L..sta
000001c0h: 72 74 53 74 61 74 65 71 00 7E 00 08 4C 00 06 73 ; rtStateq...L..s
000001d0h: 74 61 74 65 73 71 00 7E 00 05 78 70 73 72 00 13 ; tatesq...xpsr..
000001e0h: 6A 61 76 61 2E 75 74 69 6C 2E 41 72 72 61 79 4C ; java.util.ArrayL
000001f0h: 69 73 74 78 81 D2 1D 99 C7 61 9D 03 00 01 49 00 ; istx佉.藪a?...I.
00000200h: 04 73 69 7A 65 78 70 00 00 00 03 77 04 00 00 00 ; .sizexp....w....
00000210h: 0A 73 71 00 7E 00 0C 00 00 00 03 77 04 00 00 00 ; .sq.....w....
00000220h: 03 73 72 00 14 63 79 62 65 72 6C 61 62 2E 43 79 ; .sr..cyberlab.Cy
00000230h: 62 65 72 42 75 74 74 6F 6E 4D 7D 9D 0A 35 78 A6 ; berButtonM}?5x?

```

Figure 19. Part of the “.exp” file for the heating experiment

3.3 Load and execute experiments in LabExecutor.

We already have a LabExecutor prototype [23], but it does not provide any support for intermediate files. Therefore, we have to implement the loading and execution of experiments in this paper.

The object de-serialization process is very similar to the serialization process: (1) Produce a valid filename to read from; (2) Open a FileInputStream; (3) Attach an

ObjectInputStream to the FileInputStream; (4) Read the object data to object stream; and (5) Flush the object streams and close it down.

After an object is de-serialized, it is usually necessary to cast the object to its original class type. The Java code handling the de-serialization of objects in LabExecutor is shown in Appendix C:

After successful loading of the intermediate “.exp” file, the experiment is reconstructed in the window of LabExecutor. Students can then manipulate the experiment in LabExecutor. As shown in Figure 20, the heating experiment is loaded into LabExecutor. The layout of the experiment is the same as designed in LabCreator, and the experiment functions well in LabExecutor (Figure 20).

3.4 Usability test and feedback.

After the integration of CyberLab, it is helpful to know how end users feel about it. As CyberLab is designed for online science courses, I invited two students, both of whom major in natural science subjects, for a simple and short CyberLab tryout. Clearly, such a small sample size is insufficient for a formal efficacy analysis. Nonetheless, the observations are of interest.

Student A got his Master’s degree in chemistry and has two years of programming experience with C, C++, and Java. Student B is a graduate student from EMU’s Geography and Geology Department. Though knowing little about programming, B is quite familiar with the commonly used software in the Microsoft Windows platform.

The tryout had four steps: (1) Introduction to CyberLab; (2) Demonstration of how to use LabCreator and LabExecutor; (3) Creation and execution of experiments by participating students; and (4) Discussion.

The major obstacle we encountered in the tryout concerned the FSA. Student A was already familiar with FSA's, while Student B had never heard of FSA's before this tryout. Apparently, though, it is not necessary for the user to be an expert in FSA in order to use LabCreator. A fifteen-minute introduction of FSA fundamentals and how the experiment in CyberLab makes use of FSA's was sufficient for student B to complete the creation of the heating experiment.

The rest of the tryout went smoothly, and both students were able to create, export, and execute their heating experiments. Student A also created his own simple experiments.

In the discussion session, both students expressed their satisfaction with CyberLab. They agreed that CyberLab provides an easy way for instructors to create virtual experiments for online science courses. The data flow is straightforward, and the operations of both LabCreator and LabExecutor are not difficult to learn.

On the other side, they also had complaints. One thing both of them mentioned is the way an FSA is constructed. The dialog and text-based approach for constructing FSA, especially FSA edges, was not sufficiently intuitive. The information in the "FSA Edges" panel did not give them a clear global view of the FSA they were constructing. Another feature they want is be able to directly modify the experiment from the Experiment Information panel if a mistake is found.

They also pointed out some other features that need to be improved or added, which are discussed in Chapter 4.

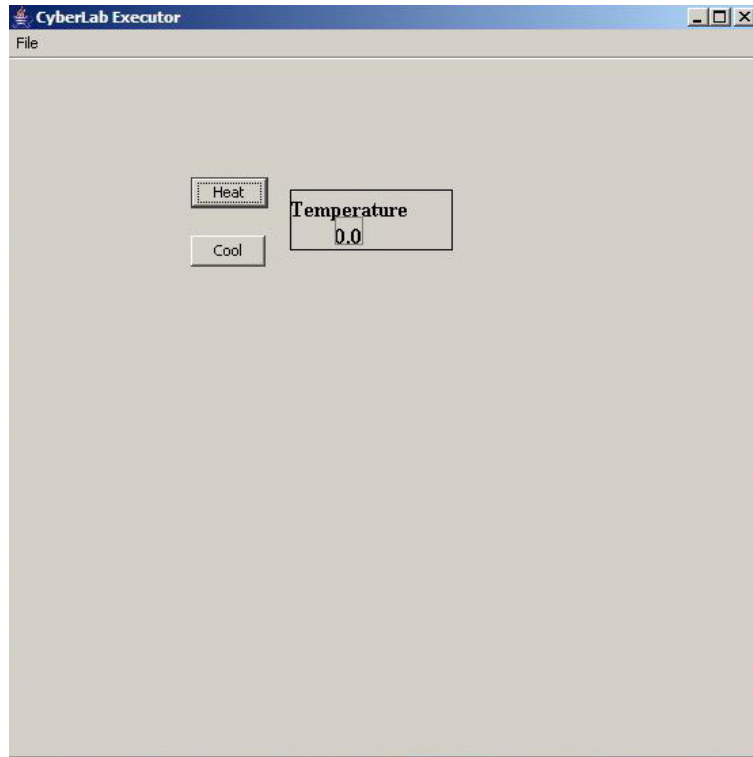


Figure 20. The heating experiment in LabExecutor

CHAPTER 4

CONCLUSION AND FUTURE WORK

4.1 Conclusion.

Through the implementation of LabCreator and the integration of CyberLab system in this paper, it has been proven that the design of CyberLab is feasible.

CyberLab can provide a package of services for the constructions of online virtual laboratories, ranging from experiments design, creation, and export in LabCreator, to experiments loading and execution in LabExecutor.

We consider CyberLab to be an important progress with unique characters in the research of online virtual laboratories.

It has been discussed that laboratory experience is very important for science education. Then why have virtual experiments rarely been integrated into online science courses? When compared with the popularity of online science courses and the great demand for online virtual experiments, the number of researchers dedicated to the development of online virtual laboratories is obviously too small to serve all needs for online virtual laboratories. It is therefore difficult to popularize online experiments unless instructors are able to design and create online experiments for their own online science courses. Another reason we must take into consideration is that though computers are now widely used, instructors with programming skills are still only a small part of all the instructors who need to create online virtual experiments.

According to our literature survey, though there have been a number of online virtual laboratories [3, 5, 15, 20, 24, 29, 30, 33, 34, 39], CyberLab is one of the only two tools (the other one is the Java Applet-based Physlets [6]) that allow online course

instructors to create online experiment themselves. More important, CyberLab is the first tool that does not require experiment creators to know programming. By freeing instructors of online science courses from learning programming languages and coding virtual experiments themselves, CyberLab owns a major edge over Physlets, which requires its users to know Java Script programming.

As a toolkit designed to construct virtual laboratories for online science courses, CyberLab inherits the advantages of online education, such as “Any time, any place” learning [9] and low requirements for laboratory facilities, while providing valuable laboratory experience to students who take online science courses.

Meanwhile, by enabling instructors with no programming experience to create online experiment, CyberLab could greatly expand its user population in online science course instructors, which in turns facilitates the popularization of virtual experiments in online science education.

Admittedly, experiments created with CyberLab are not as vivid as those video-based or 3D virtual experiments. Both LabCreator and LabExecutor are still primitive. There are a lot of things that need to be done before releasing the system to the public (discussed in section 4.2).

Nevertheless, we believe that given more time, CyberLab will become a very popular virtual laboratory system, which will make more virtual experiments available in online courses and, therefore, greatly improve the effectiveness of online science education.

4.2 Future work.

At present, the development of CyberLab is ongoing, and there is still a long way to go before it can be widely used in the creation of virtual laboratories for online courses. We analyzed the current implementation of CyberLab, reviewed the initial requirements and purpose of CyberLab, considered users' feedback, and summarized some work for future development.

First, it is definitely necessary to enrich the widget library so that CyberLab can be used to create more experiments from different subjects. For example, we may need balls and speedometers for free fall experiments, rope and brackets for pendulum experiments, tubes and pH testers for enzyme experiments, light bulbs and green plants for photosynthesis experiments. The number and variety of widgets supported will be very important for the popularization of CyberLab.

Second, it is necessary to introduce Thread programming or add internal timer components to the existing FSA of CyberLab experiments, in order to implement animations in CyberLab experiments. At present, the traversals of FSA edges in an experiment can only be triggered by users' manipulations of widgets. But when an animation is needed, the elapse of time will be used to trigger FSA edge traversals. For example, in a free fall experiment, after students press the "Start" button, the ball starts to fall from a certain height, and its position is shown as an animation. The FSA of the experiment is shown in Figure 21. In the experiment, the falling process consists of repeated traversals of the "Edge 2" in the FSA. The traversals of "Edge 2" can be implemented as threads. They can also be triggered by the passage of time since the start of the experiment. After the experiment starts, the timer tick keeps increasing at a certain

pace automatically and will stop increasing after the ball hits the ground. Once the timer tick changes, “Edge 2” will be traversed, the speed of the ball is updated, and the position of the ball is redrawn in the animation.

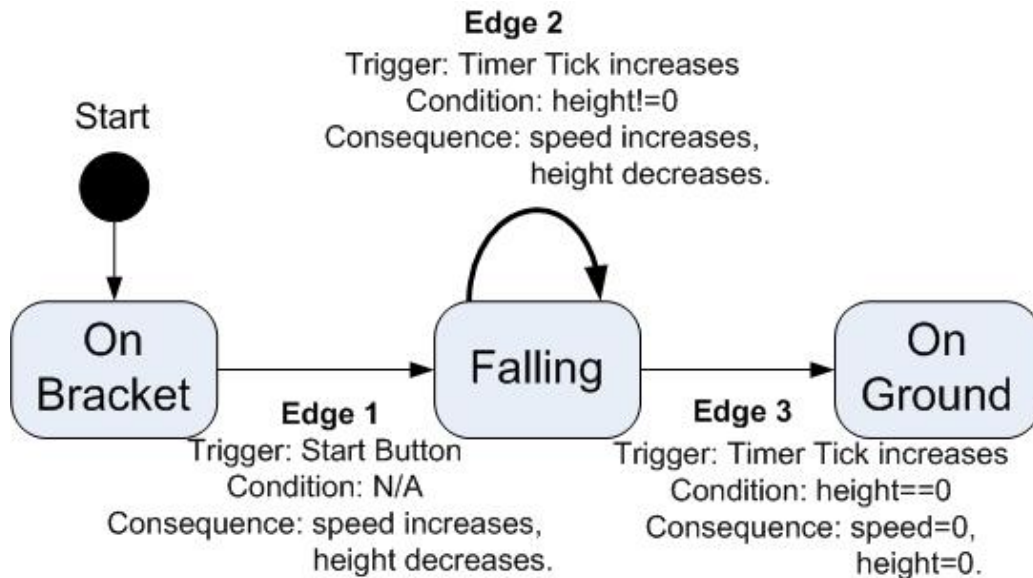


Figure 21. The FSA of the free falling experiment.

Third, to address users’ concerns on FSA construction in LabCreator, we suggest a graphical way to design the experiment FSA. It could be a diagram-based editor, where users can create an FSA by drawing diagrams similar to Figure 7 and Figure 21. Users can create an FSA state by drawing a rectangle and adding text labels as state names. When selecting active widgets for the state, users can directly drag widgets from the Experiment Workbench into the rectangle, instead of using the check boxes in the dialog window. All FSA states that have been created will appear in the editor as rectangles. Users can then create an FSA edge simply by drawing a directed line to link FSA states. Compared with the current dialog and text-based approach, the new graphical approach is

more intuitive. It will also give users a global view of the FSA they are creating, so that they can easily know what has been done and what is missing.

Fourth, users who have little knowledge about FSA may find it difficult to follow the correct procedure when creating the FSA for an experiment. For example, if one creates an FSA edge with a variable, which has not been created in the condition or consequence expression, exceptions will happen when LabCreator tries to evaluate the expression. Adding an “Experiment Design Wizard” in LabCreator may help. The wizard will give users step-by-step tutorials and instructions, which guides users to go through all the required steps in the correct order.

Fifth, when creating FSA edges in LabCreator, expressions for condition and consequence are created from strings inputted by users. During the usability test, there were complaints that it was easy to make mistakes when creating those expressions. For example, some may type “=” instead of “==” for a Boolean expression, or some may type a variable that is not created for the experiment. If the expressions are not correct, neither the preview nor the exported file will work properly. In this case, users have to check the design carefully to find out their mistakes. Though mistakes can be avoided by referring to the “Variable” panel frequently and typing carefully, the expression creation process can become more convenient to users by using an “Equation Editor.” The editor will present an equation field and a panel with numbers, available variables, and operators. When constructing an expression, users are not allowed to add any element into the equation field directly. Instead, elements of an expression are added to the equation field via clicking the elements on the panel. A prototype for the graphic user interface of the editor is proposed in Figure 22. Before adding the expression created by users into the

FSA, the editor will also check the internal model of the experiment to make sure the expression is valid, so that the internal model can focus on the expression parsing instead of checking.

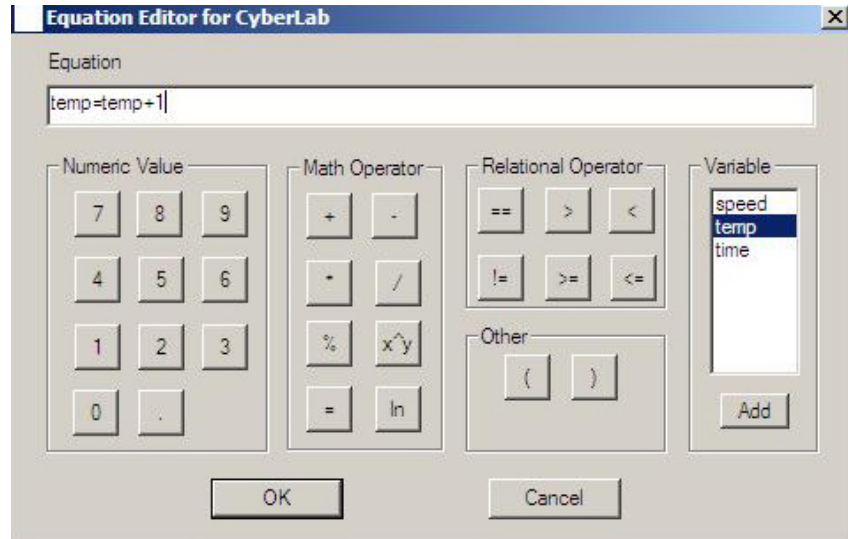


Figure 22. The prototype for the graphic user interface of “Equation Editor”

Sixth, Java Object Serialization is currently used to generate the intermediate files. This is mainly because this technology is mature and easy to implement. However, if students’ and instructors’ computers have different versions Java Runtime Environments, there might be problems when executing the experiment. At the same time, XML files have the advantage of platform-independence. Therefore, we may choose XML instead of Java serialization as the intermediate file format in the future, so that those files can be executed on different computing platforms.

Last, but not least, inspired by DropBall [26], which is described in 1.3.3, we think it is useful to add the data collection feature to LabExecutor, so that students can save the data and analyze it later. For example, in a free fall experiment, the speed and height of the falling ball are collected every second. After the experiment, students can

use the data collected by LabExecutor to generate a “Speed vs Time” graph and verify whether the speed of the ball during the falling process was in accord with the laws of Newton mechanics.

Of course, what we summarized at this moment cannot cover all the work that needs to be done to improve CyberLab. However, we hope this chapter could give a brief guideline for the future development of CyberLab.

REFERENCES:

- [1] Bell, J. The biology labs on-online project: producing educational simulations that promote active learning. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning*, (February 1999).
<http://imej.wfu.edu/articles/1999/2/01/index.asp> (Accessed October 4, 2006).
- [2] Carr, S. As distance education comes of age, the challenge is keeping the students. *Chronicle of Higher Education Information Technology*, (February 2000).
<http://chronicle.com/free/v46/i23/23a00101.htm> (Accessed October 2, 2006).
- [3] Chakaveh, S., Zlender, U., Skaley, D., Fostiropoulos, K. and Breitschwerdt, D.. DELTA's virtual physics laboratory (case study): a comprehensive learning platform on physics and astronomy. In *Proceedings of the conference on Visualization '99: celebrating ten years*. (San Francisco, California, USA, 1999). 421-423.
- [4] Chaven, A. and Pavri, S. Open-source learning management with Moodle. *Linux Journal*, 128 (December 2004). <http://www.linuxjournal.com/article/7478>
(Accessed October 20, 2006)
- [5] Chen, X. *Online virtual biology laboratory*. Master of Science Thesis, Eastern Michigan University, Ypsilanti, MI, 2001.
- [6] Christian, W. and Belloni, M. *Physlets: Teaching Physics with Interactive Curricular Material*. Prentice Hall, New York, NY, 2001.
- [7] Chu, K. C. What are the benefits of a virtual laboratory for student learning?. In *Proceedings of the HERDSA Annual International Conference*. (Melbourne,

- Australia, July 1999). www.herdsa.org.au/branches/vic/Cornerstones/pdf/Chu.PDF
(Accessed September 28, 2006).
- [8] Colace, F., De Santo, M. and Vento, M. Evaluating on-line learning platforms: a case study. In *Proceedings of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)* (Hawaii, USA, January 2003). 5, 154-162.
- [9] Dehoney, J., Booth, L., Lau, K.F., Reichgelt, H., Rutherford, R.H., and Stewart, J. Many cooks improve the broth: developing an inter-institutional, online, bachelor of science degree in information technology. In *Proceedings of the 4th Conference on Information Technology Curriculum (CITC4'03)* (Lafayette, Indiana, USA, October 2003). Distance education session, 155-159.
- [10] Evett, M.P. *CyberLab, a tool for constructing laboratory experiments for on-line science courses*. Proposal for a Spring-Summer Award for Research and Creative Activity, Eastern Michigan University, Ypsilanti, MI, 2003.
- [11] Fuller, A., Awyzio, G. and McFarlane, P. Using WebCT to support team teaching. In *Proceedings of the Second IEEE International Conference on Advanced Learning Technologies (ICALT'01)* (August 2001). 315-318.
- [12] Garapati, S. *Online scientific lab kit*. Research Project Report, Eastern Michigan University, Ypsilanti, MI, 2003.
- [13] Goldberg, M.W., Salari, S. and Swoboda, P. World Wide Web - Course Tool: an environment for building WWW-based courses. *Computer Networks and ISDN Systems*, 28, (1996), 7-11.
<http://www.ra.ethz.ch/CDstore/www5/www156/overview.htm> (Accessed September 29, 2006).

- [14] Harms, U. Virtual and remote labs in physics education. In *Proceedings of the Second European Conference on Physics Teaching in Engineering Education*, (Budapest, Romania, 2000). <http://www.bme.hu/ptee2000/papers/harms1.pdf> (Accessed September 29, 2006).
- [15] Kim, J.H., Park, S.T., Lee, H., Yuk, K.C. and Lee, H. Virtual reality simulations in physics education. *Interactive Multimedia Electronic Journal of Computer-Enhanced Learning* (February 2001). <http://imej.wfu.edu/articles/2001/2/02/> (Accessed October 4, 2006).
- [16] Kudire, P.K. *Drag and drop toolkit and on-line biology laboratory experiments*. Research Project Report, Eastern Michigan University, Ypsilanti, MI, 2002.
- [17] Kurotsuchi, B.T. The Wonders of Java Object Serialization. *ACM Crossroads*, 4.2(Winter 1997). <http://www.acm.org/crossroads/xrds4-2/serial.html> (Accessed October 4, 2006).
- [18] Lee, Y., Ma, W., Du, D.H.C. and Schnepf, J.A. Creating a virtual network laboratory. In *Proceedings of the International Conference on Multimedia Computing and Systems (ICMCS'97)* (June 1997). 642-643.
- [19] Lunney, T. and McCaughey, A. Object persistence in Java. In *Proceedings of the 2nd International Conference on Principles and Practice of Programming in Java*. (Kilkenny City, Ireland, 2003). Information systems session, 115 - 120.
- [20] Marone, M. The Mercer Online Interactive Chaotic Pendulum. *Computing in Science and Engineering* (July/August 2002). 94-97.
- [21] McDonald, M., Dorn, B. and McDonald, G. A statistical analysis of student performance in online computer science courses. In *Proceedings of the 35th*

SIGCSE Technical Symposium on Computer Science Education (SIGCSE '04)
(Norfolk, Virginia, USA, March 2004,). 36, 1, 71-74.

- [22] Parker, B. and Hankins, J. AAHE's seven principles for good practice applied to an online literacy course. *Journal of Computing Sciences in Colleges* 17, 4 (2002), 39-48.
- [23] Parlapalli, S. 2005. *Implementation of the student-side component of CyberLab*. Master of Science Thesis, Eastern Michigan University, Ypsilanti, MI, 2005.
- [24] Pheatt, C.B. and Ballester, J.L. Developing web-based experiments. *Journal of Computing Sciences in Colleges* 18 (2003), 238-245.
- [25] Physics Experiment Online Website.
<http://freespace.virgin.net/gareth.james/virtual/index.html>. 2006. (Accessed October 4, 2006).
- [26] Pniower, J. C., Ruane, M., Goldberg, B.B. and Ünü, M.S. Web-Based Educational Experiments. In *Proceedings of the 1999 ASEE National Conference*, (Charlotte, NC, USA, June 1999). <http://ultra.bu.edu/papers/asee-web-99.pdf> (Accessed September 29, 2006).
- [27] Pradhan, M.. *Online biology laboratory experiments and Java toolkit*. Research Project Report, Eastern Michigan University, Ypsilanti, MI, 2002.
- [28] Pullen, J. Scaling up a distance education program in computer science, In *Proceedings of the 11th Annual SIGCSE Conference on Innovation and technology in Computer Science Education (ITICSE '06)* (Bologna, Italy. June 2006). 38, 3, 33-37.

- [29] Pullen, J. Applicability of Internet Video in Distance Education for Engineering, In *Proceedings of 31st ASEE/IEEE Frontiers in Education*, (Reno, NV, USA, October 2001). T2F, 14-19.
- [30] Rodriguez, F.G., Silva, J.L.P, Rosano, F.L., Contreras, F.C. and Vitela, A.M. Creating a High School Physics Video-Based Laboratory. *IEEE MultiMedia* (July 2001), 78-86.
- [31] Seng, L. and Mohamad, F.S. Online learning: Is it meant for science courses? *Internet and Higher Education*, 5 (2002), 109–118.
- [32] Shi, Z. *Portable Intermediate Representations of Collections of Java Objects*. Reserch Project Report. Eastern Michigan University, Ypsilanti, MI, 2004.
- [33] Shirmohammadi, S., Saddik, A.E., Georganas, N.D. and Steinmetz, R. Web-based multimedia tools for sharing educational resources. *ACM Journal on Educational Resources in Computing (JERIC) 1*, 1 (2001), Article No.9.
- [34] Subramanian, R. and Marsic, I. ViBE: Virtual Biology Experiments. In *Proceedings of the tenth international conference on World Wide Web (WWW '01)* (Hong-Kong, May 2001). 316-325.
- [35] Virtual Chemistry Website. <http://www.chem.ox.ac.uk/vrchemistry/>. 2006. (Accessed October 4, 2006).
- [36] Wikipedia. *Finite State Machine*. http://en.wikipedia.org/wiki/Finite_state_machine. 2006. (Accessed October 4, 2006)
- [37] Wikipedia. *Serialization*. <http://en.wikipedia.org/wiki/Serialization>. 2006. (Accessed October 4, 2006)

- [38] Wikipedia. *Usability testing*. http://en.wikipedia.org/wiki/Usability_testing. 2006.
(Accessed October 27, 2006)
- [39] Wittig, S. W. and Eggensperger, R. APL in Computer-Assisted Instruction: Simulation of Stochastic Processes in science teaching. In *Proceedings of the Eighth International Conference on APL* (Ottawa, Canada. 1976). 447-455.
- [40] Wu, Y., Chan, T., Jong, B., Lin, T. and Liang, Y. A Web-Based Dual Mode Virtual Laboratory Supporting Cooperative Learning. In *Proceedings of the 18th International Conference on Advanced Information Networking and Applications (AINA'04)* (March 2004) 1, 642-647.

APPENDICES

APPENDIX A: An excerpt of the Java code for widget-related classes.

```
/** This is part of code from Widget.java */
public interface Widget {
    public Model getCyberModel();
    public void setCyberModel(Model model);
    public String getName();
    public void setName(String name);
    public void setPosition(Point pos);
    public Point getPosition();
    public void setLabel (String label);
    public String getLabel ();
}
/*****
/** This is sampled from CyberButton.java */
    public class CyberButton extends JButton implements Widget {
        ...
    }
...
/*****
/** This is sampled from CyberTextField.java */
    public class CyberTextField extends JComponent implements Widget{
        ...
        public void paintComponent(Graphics g){
            ...
        }
    }
...

```

APPENDIX B: An excerpt of the Java code for widgets drag and drop.

```

/****This is sampled from Creator.java *****/
/****Demonstrate how Drag and Drop works for a control button*****/
... ..
// Create a control button
CyberButton cntrlBtn=new
    CyberButton(mod,ctrlBtnDlg.getButtonLabel());
// Set the button to the initial position specified by user.
cntrlBtn.setPosition((Point)ctrlBtnDlg.getButtonPos());
... ..
... ..
cntrlBtn.addMouseListener(new MouseAdapter(){
public void mousePressed(MouseEvent evt){
    if(evt.isPopupTrigger()){
        ... .. /* action for right button release*/
    }
    else{
        ... ..
        setBtnOrigin(evt.getX(),evt.getY());
    }
}
public void mouseReleased(MouseEvent evt) {
    if(evt.isPopupTrigger()){
        ... .. /* action for right button release*/
    }
    else{
        if( MouseMoved(evt.getX(),evt.getY()))
            moveWidget(evt);
        ... ..
    }
}
});
... ..
... ..
// Set the original position of the widget
// used for comparison with new position
private void setBtnOrigin(int x, int y){
    OrgMouseX=x;
    OrgMouseY=y;
}

// Determine whether a widget has been moved by user
private boolean MouseMoved(int x, int y){
    return !(x==OrgMouseX && y==OrgMouseY);
}

// Move the widget to new position
private void moveWidget(MouseEvent evt){
    Object evtSrc=evt.getSource();
    for (int i = 0; i < widgetsOnCanvas.size(); i++)
        if (widgetsOnCanvas.get(i)==evtSrc) {
            // Select widget to be moved
            Widget tmp=(Widget)(widgetsOnCanvas.get(i));
            // Get new position
            evt.translatePoint((int)tmp.getPosition().getX(),

```

```
                (int)tmp.getPosition().getY());
// Set new widget position unless out of bound
if(PosInBound(evt.getX(),evt.getY()))
    tmp.setPosition(evt.getPoint());
    else
        System.out.print("Out of Bound");
// Repaint the workbench
// and display the widget at new position
mycanvasPanel.update(widgetsOnCanvas);
jFrame.pack();
return ;
    }
return;
}
... ..
```

APPENDIX C: An excerpt of the Java code for Java serialization.

```

/*****This is for serialization in LabCreator*****/
... ..
try{
ObjectOutputStream out=new ObjectOutputStream(
                                new FileOutputStream(filename)
                                );

out.writeObject(exp);
out.close();
}
catch (Exception e){
    e.printStackTrace();
}
... ..

/*****This is for de-serialization in LabExecutor*****/
try{
ObjectInputStream in=new ObjectInputStream(
                                new FileInputStream(filename)
                                );

newexp = (Experiment) in.readObject();
in.close();
}
catch (Exception e){
e.printStackTrace();
}
... ..

```

