Design and Implementation of an Educational Rich Internet Web Application Capable of Physics Simulation within Real-Time Collaborative Interfaces

by

Tasha M Thomas

Submitted to the Department of Mechanical Engineering
On May 21, 2009 in Partial Fulfillment of the
Requirements for the Degree of

Bachelor of Science

at the

Massachusetts Institute of Technology

June 2009

Signature of Author.....
Department of Mechanical Engineering
May 21, 2009

Certified by.....
David L. Brock
Principal Research Scientist for the Laboratory for Manufacturing & Productivity
Thesis Supervisor

Accepted by.....
Professor J. Lienhard V
Collins Professor of Mechanical Engineering
Chairman, Undergraduate Thesis Committee

Design and Implementation of an Educational Rich Internet Web Application Capable of Physics Simulation within Real-Time Collaborative Interfaces

by
Tasha M Thomas

Submitted to the Department of Mechanical Engineering
On May 21, 2009 in Partial Fulfillment of the
Requirements for the Degree of
Bachelor of Science

ABSTRACT

This paper attempts to explore the design needs and implementation needs for the creation of a rich internet application (RIA) designed to run inside a user's web browser that simulates simple physics concepts and allows for multiple users to collaborate with one another in real time. The purpose of using such application would be for use in an academic environment in order to allow for a novel way for teachers to interact with students in the teaching of physics. The open source software Eclipse together with the Flex Builder plug-in was used as an application programming interface in order to develop such application that would be designed to run with Adobe Flash Player.

The application was designed to include user interactions that allows user to click and drag objects and watch the effects of inelastic collisions, acceleration and gravity, and frictional forces. Multiple users can connect to a client-server network in order to communicate and interact over multiple browsers running the same simulation. The application allowed for one user to click and move an object, while other users could watch the movement of said object in real-time within their own browser.

Results of the design and implementation of this program showed that creating an educational program that can be used in a network setting and allow for real-time collaboration to be possible. Improvements must be made, however, in information processing between connected platforms, time delays due to these processes, and problems arising for atypical user movements and interactions.

Thesis Supervisor: David L. Brock
Title: Principal Research Scientist for the Laboratory for Manufacturing & Productivity

Acknowledgements

I would like to thank Dr. David L. Brock for all his help, guidance, and teaching. He was gracious enough to give me his time and knowledge in helping me prepare this paper.

I would also like to thank my family for all their encouragement.

Finally, I would like to thank Jason S Pellegrino, for all his inspiration and support.

Table of Contents

1. Introduction

The web hosts a nearly infinite amount of data in which users all over the world can access, resulting in the exchange of information and ideas to occur at rates faster than ever before and allow that information to reach remote places in a much more efficient manner. With this high rate of information exchange, the abilities for users to learn from one another and collaborate with each other have expanded in enormous ways. Furthermore, another step has begun to develop in the way information is shared with the web. This step has begun working toward allowing users to interact with one another using common applications such as Microsoft Word and PowerPoint using different machines and allow users to manipulate, update, and discuss information in real time. An example of this would be the use of PowerPoint in a global meeting to display a presentation on a viewer's screen and stream the audio all live.

Recently, there have been a number of advances concerning this technology. A number of web services have been developed that address the ability to allow multiple-user communication and provide users with a way to access useful applications over the Internet. An important use of the realization of this new technology involves the opportunity to use it in an educational setting. Integrating this type of information exchange can allow learning possibilities such as improvements in the ways students can interact with lessons and teachers, provides additional avenues for receiving education, and can even allow lessons to be taught to many users across the globe at once.

Acknowledging the significance of this prospect, this paper will attempt to explore the current available means that can be used to develop this type of web service by designing

an original application that will attempt to simulate the type of application that could be used in a learning environment. The intention of the application will be its possible use in teaching of basic physics through interactions between both the teacher and the students. Any problems that arise due to limitations of programming or data-sharing will be detailed and discussed in terms of the application itself and then broadened to the general idea of information exchange.

2. Background

2.1. Web Services and Applications

Web services can be described as application interfaces that define certain standards to allow for the use of that application by another machine regardless of differences in operating systems, programming languages, and hardware between the machines. These systems implement the hypertext transfer protocol (HTTP) as a standard for transporting messages between the client and server. Messages that arise due to requests made by the client and responses made by the server are encoded in the standard Extensible Markup Language (XML) format using protocols such as Simple Object Access Protocol (SOAP) or XML-RPC (remote procedure call). As a result of these standards, messages can be received and read on either side of the connection regardless of the individual operating systems. Alternative approaches in the implementation of web services include Service-Oriented Architecture (SOA) and Representational State Transfer (REST), which define different standards for the communication between clients and servers.

Typically, web services are built using application programming interfaces (APIs) hosted by the service provider which are accessed by a client over the Internet. The interface uses a specific programming language in building applications and there are numerous APIs available, each adopting various programming languages. Since APIs are coded in a browser-supported language, web applications, a type of web service, can be easily created. Web applications are meant to execute within the client's browser and are designed to reduce processing on the client-side by leaving most of the processing to be done by the server. An example of a popular web application would be Google Calendar or eBay.

A subset of the web application is the Rich Internet Application (RIA) which acts just like a web application in that it also runs within a browser. While web applications run according to browser specifications, the RIA runs via plug-ins such as Adobe Flash or Java applets. Since RIAs rely only on Flash or Java to run, any problems that may arise due to inconsistencies between browsers and their specifications are virtually eliminated.

There are many advantages that can be seen with the use of web applications and RIAs. One advantage is that web applications eliminate the process of distributing software for companies and get rid of the need for customers to install that software on their personal computers. As a result, new software is able to reach more people much more easily and updates to the software become more manageable to implement and distribute to users. As the use of web applications and RIAs become more popular, users are beginning to see more advanced and complex applications being created. For example, some applications offer the ability to perform the same tasks as the software typically seen installed on the client's computer. This includes programs such as Google Docs, which acts just like a word

processor, allowing the user to edit and save documents, and Splashup, a service that offers the same features and abilities that image editors like Photoshop extend. However, unlike the software that must be installed on a client's computer, some web applications that mimic fully-functional commercial software reduce the user's need for disk space and update in a more efficient manner. Also, since these applications are built to run in a browser window, they gain ability to perform on multiple platforms, thereby widening the field of potential customers and users.

With the advantages that web applications bring, businesses have even begun to see the potential marketability that web applications create. Software companies are starting to emerge with the intention of distributing software as web applications for a price. The idea is that customers would pay a fee to use the program while the company manages the cost of providing and running the servers required to use the application. The advantage to this is that companies avoid the need to pay for the costs associated with packaging and distributing the software. These companies have come to be known as Application Service Providers (ASPs) and are beginning to gain a substantial place in the market.

2.2. Overview of the Application Programming Interface

As mentioned before, a web application is created using an API. There are many APIs available for any platform, each offering an array of features and capabilities. One such program is Eclipse[1], an open source software, that is written chiefly for the language Java, but also supports other languages through the use of plug-ins. It is built as an integrated development environment (IDE), which is a software platform that includes the tools

needed for software development such as a source code editor, a debugger and compiler, and build automation. Available as a plug-in or as a stand-alone IDE built on the Eclipse platform is Adobe Flex Builder. It is primarily used for the development of RIAs and is meant for operation with Adobe Flash. Adobe Flash Player[2] is the multimedia application that is meant to handle SWF (named so because of Shockwave Flash—an earlier version of a multimedia player released by Adobe) files and is typically used to run files built for animations or movie clips.

While as a stand-alone, Eclipse can only develop applications written in Java, the Flex Builder platform allows developers to write their applications in MXML and ActionScript. The markup language, MXML, is an extension of XML and is used in building the lay out for application interfaces. Used in conjunction with MXML, ActionScript is the language used in the development of RIAs and is used to generate the SWF files needed to run an application on Flash.

2.3. The Computer Network Architecture

A computer network allows for a group of computers or other related devices to communicate with each other in order to exchange any resources or information that each may have. There are numerous types of network set-ups designed for the type of communication one may need. For example, the simplest type of network connection occurs through an Ethernet network, which uses physical wiring to connect devices such as a printer to a personal computer.

However, on larger scales, networks must distinguish how each system communicates with each other in the exchange of information. One example of this type of structure is the Client-Server architecture, which is designed to separate connected computers according to either belonging to the client-side or the server-side of the network. Typically, the client system initiates processes and information exchange through requests made by the client software. When receiving such requests, the server-side program then accesses the information required to perform the desired request and sends the needed information back to the client. A downside to this type of architecture occurs in the event that the server drops out of the network connection, thereby making any sort of information exchange impossible.

Another type of network architecture is known as Peer-to-Peer networking (often abbreviated as P2P). This architecture does not distinguish between client-side and server-side and instead uses the collective processing power of all the computers connected within the network. P2P networks are typically used in file-sharing environments, such as torrents. Since every computer is acting as an equal in the network, if one computer disconnects from the network itself, the information exchange between all devices still within the network is not interrupted. Furthermore, the issue of storage space is reduced since not just one device is acting as the server and must house all required information. Instead all information is shared by all the computers in the network. Pure P2P networks are rarely in use today, with most networks using a sort of hybrid of P2P and client-server network architecture.

2.4. Current Physics-Simulated Web Applications

With the implementation of these powerful developing tools, complex and creative applications can be designed. For instance, an application that played a significant part of the inspiration for this paper is the Microsoft Physics Illustrator[3], also referred to as Magic Paper. This application was developed through the collaborative efforts of Microsoft and MIT. It features software designed to work on tablet PCs and allows the user to hand-draw sketches that feature mechanical devices and apply forces to the user's sketches in order to allow for interaction. However, this application does not support any networking capabilities and can only be implemented within the computer that houses the required software.

There are also thousands of games, animations, and applets available on the web that run on Flash and, specifically, there are many such applications devoted to simulating Newtonian physics. Game developers strive for a sense of physical realism in their games and applets displaying basic physics concepts can easily be accessed on the Internet. A popular online game which simulates common physics concepts is Crayon Physics[4], developed by Petri Purho. It requires the player to guide a ball by sketching simple shapes like inclines, boxes, and planks, to a designated spot on the game interface. Crayon Physics also features a level editor that allows users to design their own custom game levels and allows them to upload their custom levels in order to share with other players over the web. Though this program allows for the capability of sharing, it lacks the ability for users to participate in real-time within game play.

The concepts inherent to Newtonian physics does not typically translate well into code. Flash executes its file in two ways: changes based on time and changes based on frames. Converting the equations produced by Newtonian mechanics and rigid-body motion proves difficult when writing the movement and interactions of objects on a per frame basis. While working with time changes in governing how objects behave physically can be a bit easier, problems are still apparent since changes in time and distance are measured in pixels. Generally, developers work around these issues by defining different values for gravity, viscosity, friction, and other basic concepts. Also, Newtonian mechanics can produce complex equations of motion for complicated systems and using these equations for the application can prove to be impractical as it can slow down the application's operation. As a result, developers tend to be flexible with the realism of the physics involved in their program in order to preserve the application's run time.

However, despite the difficulty in reproducing Newtonian physics exactly, there are still many web applications devoted to simulating physics with a large amount of realism. Many of them are even specifically designed to be used as teaching tools, allowing students to interact with the program by changing values for properties such as mass, spring constants, and gravity so the student can see the effect that each property can have on the motion and reaction of the system. Studies have been conducted investigating the effect that these programs may have on a student's learning of physics[5]. These studies have suggested that while effective in some ways, there are also some problems that develop when implementing these types of applications for student use. A case study involving students having to use a modeling program to learn basic physics showed that while students

performed higher on physics-related tasks, problems had presented themselves when it was observed that some students did not always use the program as the designer had originally intended[6]. In addition to this study, other studies have suggested significant learning effects with computer programs, but these studies also stress the crucial rule that the educator plays in a student's capacity for learning. However, while there are many interactive learning tools available for student use, it is very difficult to find one that allows for both the teacher and student to interact with one another with the same application.

3. Methods For Development

For the development of an application intended for teacher-student interaction for the teaching of physics-related subjects, the Eclipse Classic 3.4.2 platform was used as a developing environment in conjunction with the Adobe Flex Builder 3 plug-in. In addition, the scripting language PHP was used for the production of the web page that will run the application.

A conceptual design of the program's layout would provide an area for teacher-student interaction to move, drag, and throw objects; a side panel to house all the available objects designed to be applicable to important physics concepts; and a menu bar to allow students to modify and save their work and to allow teachers to create lectures and to view and grade student work. Simple objects such as balls to model rigid-body motion, inclines to illustrate the effects of collision and movement, and springs to help in the learning of force and energy were all written into the program.

To provide fundamental physics concepts, the program would include basic ideas like the effects of acceleration due to gravity on a falling object and the concept of a bouncing ball and its tendency to return to equilibrium. Within the program, the relationship between gravity, velocity, and displacement was utilized to supply the conditions for movement of the object in the code. Additionally, concepts like the coefficient of restitution, which relates directly to elastic and inelastic collisions, and the effects of spring constants were incorporated into the code. Furthermore, to avoid infinite calculations by the program, small velocities were considered negligible in order to end the program's execution properly and to avoid calculating movement that may not even register visually on the user's screen. Finally, conditions were developed in order to define the resulting velocities and displacements when two objects collided with each other.

4. Results

4.1. Program Design

The design of the program worked to simulate physics by incorporating gravity, inelastic collisions, and friction to help in the aid of learning physical realizations of falling objects. Initially, the program sets up a simple 2-dimensional representation of a ball and an incline, both drawn at initial positions specified by the code (which can be read in the Appendix). The ball is set with an initial velocity, which defines how much the ball moves after a time event. The time event function, named as OnTick(), is initiated to occur 25 milliseconds after the previous time event initiation.

Within this OnTick() function, ball movement and ball collision with any objects are defined. It is important to note that even though gravity has been used, it has been reduced and does not equal the typical value of 9.81 $\frac{m}{sec^2}$ because of the previously mentioned problem of converting from a distance-time space to one that does not recognize the existence of movement. Therefore, the constant for gravitational acceleration was set to an arbitrary value. This still achieves the effect of simulating an object's acceleration as it falls. Similarly, the coefficient of restitution and the constant for frictional force were also set to arbitrary values. Further conditions were applied to prevent infinite looping by the program by stating that if velocities were small enough (so small that any changes would be essentially indistinguishable on the user's screen), the program would restart and set the ball back to its initial position.

The design of the interface itself includes a menu bar to allow for additional code to allow users to bring up additional windows such as a chat window or an objects container window. A vertical container was also added to allow for additional code that would allow users to choose an object to add to the canvas that simulates the simple game physics. For example, a potential container was added to house three buttons that the user could interact with to add another ball, incline, or spring. Furthermore, a window was added to allow each user to see the position of the ball as it is moved. Shown below in Fig. 1 is a potential outlay of the desired final program.

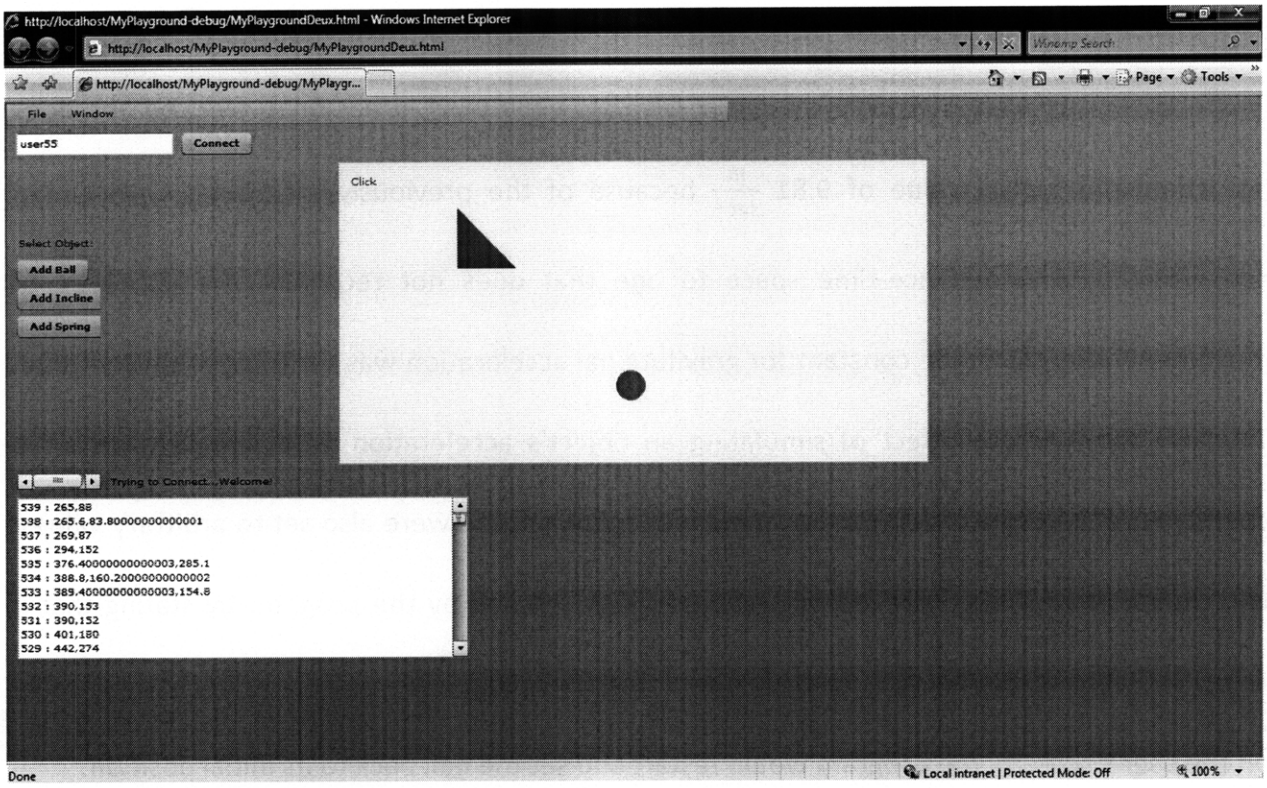

Figure 1. Screenshot of running program in user's browser.

The ability for the user to interact with the program includes the ability to pick up the ball with the user's mouse (triggered by the event of clicking the mouse and holding it down and subsequent movements of that mouse). Once the user releases the mouse, the ball is essentially released back into the movement-portion of the code and continues to fall and bounce as it did before the event. However, problems occur in certain events of atypical mouse movement. Since ball movement was only defined to occur within the canvas of the application, in the event that the ball is moved out of the realm of the canvas, the program is unable to run subsequent ball movements. As a result, the ball essentially freezes outside of the canvas and remains there even in the event of the user releasing the mouse. The ball may continue to even follow the user's mouse movement even after the release of said mouse. The program must restart within a new browser instance in order for the application to run properly again. An example of this occurrence is shown in Fig. 2 below.

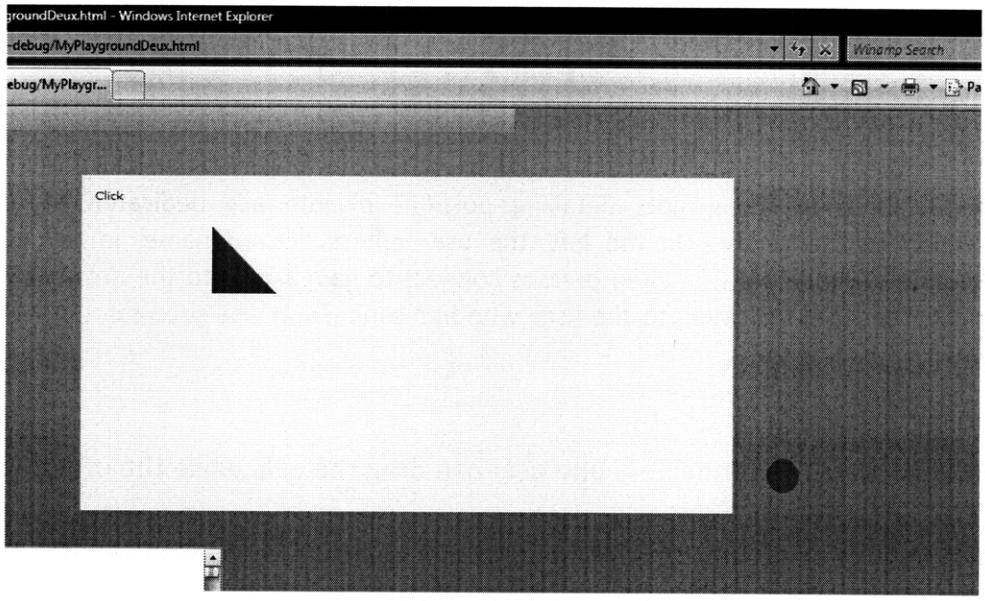

Figure 2. Close-up screen shot of bug within program in the event of ball being moved outside the perimeter of the canvas.

4.2. Network Capabilities

The network architecture used for the implementation for the application's collaborative abilities was client-server architecture. Previous written code by Dr. David L. Brock was integrated into the application and changed according to needs in order to allow for networking.

When accessed through the user's browser, the user is prompted for a name (if the application is not given one, it will randomly assign the user a generic name and number) and then connects to the server by clicking the button "Connect." A label tells the user whether or not a connection was established.

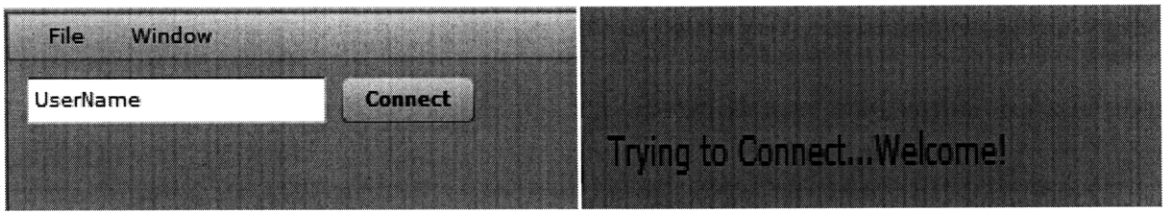

Figure 3. Close-up screenshots detailing portions of interface dedicated to user connection to the server. On the left, the user enters desired name, in this case "UserName", where the user then presses connect to gain access to the network. On the right, the label indicating to the user whether connection was successful or not is shown.

When multiple users are connected, one user can drag the ball, while the others can view the subsequent movements of that ball on their browsers. This occurs by the user that is doing the movement sending the pixel position coordinates to the server. The server then sends that information to the other connected users. When this information is received, the users that are not interacting with the ball stop their simulation and begin to follow the ball's movements according to the received coordinates. When the user drops the ball, the simulation is restarted within that user's browser. A text box was included within the interface that reads off the received information from the server regarding the ball's position. This allows for the user to be aware of the ball's position and become aware of any problems that may arise due to bugs within the program itself, which is detailed in Fig. 4.

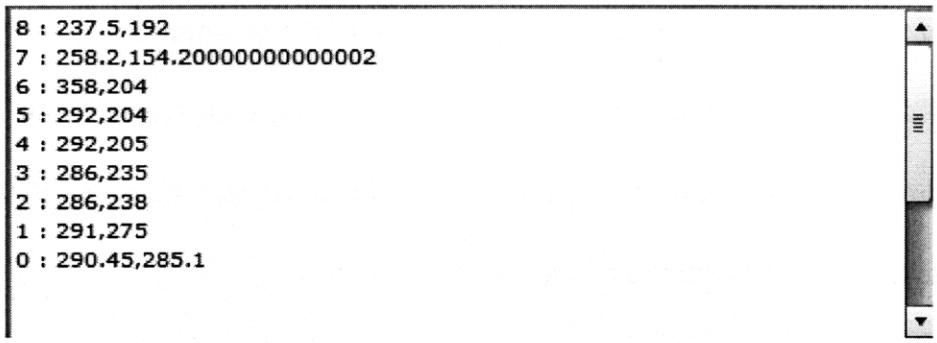

Figure 4. Text box that displays pixel position coordinates of the ball as user moves it.

However, some problems presented themselves upon the running of the program. Since position information is essentially traveling from the initiated user to the server then to the listening user, a time delay between initiation and implementation on the listening user's browser is created. This creates the ball movement to appear jumpy and slow. Furthermore, listening user is unable to process the incoming position information as fast as it is being received. This results in creating problems within the listening user's simulation. As the listening user is receiving information at a very fast rate, the user is unable to process that information as fast and creates such information in becoming jumbled or mixed up. Thusly, the listening user cannot receive all the positions that the active user is sending, which creates more jumping within the simulation itself.

5. Conclusions and Recommendations

Results suggest that the creation of an application with physics-simulated programming and real-time collaborative abilities is entirely possible. Furthermore, these type of applications can be implemented for educational purposes and may aid in teacher-student interaction and may

improve upon student learning. However, the program outlined within this paper is a simple application designed to test the possibilities of implementing such type of web application. There are numerous possibilities for further improvement on this type of RIA and further improvements can be made with the network architecture.

Recommendations for further studies and application programming include integrating more objects for user interaction, including adding mass to objects, varying densities, fluid flow, viscosity, etc. The vertical container can be further programmed to allow the user to use the button to decide to add any other objects in which they can click the button and then drag the object into the canvas and drop it into the desired location. After an object is added, the program can recognize it and initiate code that will allow for interaction with other objects already in the canvas. Also, programming allowing for user-designed changes in constants such as gravity, coefficient of restitution, and frictional force can be easily added and implemented into the current program layout.

Recommendations for improving the network architecture include implementing a new type of architecture that provides for faster information exchange to allow for a smoother real-time collaboration. Also, in order to truly use this program in a realistic educational setting object movement, even during times where there is no user interaction occurring, to simultaneous movement across all browsers that are connected to the network at a given time must be added. Furthermore, improvements in fixing the bugs found when the user stopped dragging the ball are needed, as the browser that is following the ball movement (and thus is not interacting with the program) is not continuing its motion as defined by the program.

Additional recommendations to improve the user interface include the implementation of a chat window that will allow students to enter questions or answers as text that will be seen by the teacher and the rest of the class, which will further allow the teacher to respond to such questions within the interface. Windows that display the current list of users connected and their respective names can also be added to the interface. Additionally, students can upload assignments and access premade lessons through a common folder shared among those connected to the network. This folder can be seen by all users and can be accessed by them. Furthermore, an interface specifically designed for the teacher's use can also be designed and integrated into the current program. For instance, the teacher's interface can differ from that of the student's by housing an area where the teacher can access and change grades, access and view student's work done during assignments when the teacher was not present, and make and upload lessons to allow for student access at later times.

Finally, an important recommendation for further study would include improvements upon problems of contention. That is, when multiple users try to interact with the same object at the same time, this creates opposing information being sent to the server. This results in the server being unable to proceed because of the opposing information, which causes the simulation to stop working entirely. Creating a program that determines a user that has ultimate say in what information is implemented can fix the problem. For instance, naming the teacher as the user that the server will accept information and ignore all opposing information will provide a simple fix. Also, the server can ignore all information in the case of multiple users interacting at the same time with the same object and action and just continue the simulation as is while it waits

for another instance where this is not the case, in which it will implement that information instead.

References

1. The Eclipse Foundation, "About the Eclipse Foundation," Eclipse Foundation, Inc. Available at: http://www.eclipse.org/org/. 2009.
2. Adobe Flash Player, "Products: Adobe Flash Player," Adobe Systems Incorporated. Available at: http://www.adobe.com/products/flashplayer/. 2009.
3. iCampus, "Introducing Natural Interaction (Magic Paper)," iCampus: the MIT-Microsoft Alliance. Available at: http://icampus.mit.edu/MagicPaper/default.aspx. Last Accessed: May 12, 2009.
4. Purho, P., "Kloonigames: Monthly Experimental Games," Available at: http://www.kloonigames.com/blog/. Last Updated: May 8, 2009.
5. Böhm, P., "Computer-Assisted Teaching and Learning of Physics," *WDS '08 Proceedings of Contributed Papers Part III*. Pgs. 19-23. 2008.
6. Böhm, P., "Computer-Assisted Teaching and Learning of Physics," *WDS '08 Proceedings of Contributed Papers Part III*. Pgs. 19-23. 2008.

Appendix

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
      layout="absolute" creationComplete="init()">

            <mx:Script>
            <![CDATA[
                  import mx.formatters.NumberFormatter;
            import mx.core.UIComponent;

            // Constant properties for game objects.
      private var ballRadius:Number=15.0;
      private var ballMass:Number=2.0;
      private var inclineHeight:Number=60;
      private var springConstant:Number=140;
      private var springLengthRelaxed:int=20;

      private var gravity:Number=.2; // This must be different since working
in pixels.
      private var coefficientRestitution:Number=0.8;
      private var ballInitialVelocity:Number=2.0;
      private var dxBall:Number;
      private var dyBall:Number;
      private var dtTime:Number=1;

      // Sprites.
      private var ball:Sprite=new Sprite();
      private var incline:Sprite=new Sprite();
      private var spring:Sprite=new Sprite();
      private var uiComponent:UIComponent=new UIComponent();
      private var myTimer:Timer;
      private var isDragging:Boolean=false;
      private var isFollowing:Boolean=false;

      // Set up initial function.
      private function init():void
      {
            initChat();
      }
      // Set up initial simulation function.
      private function initSimulation():void
      {

            // Draw and place the ball.
            this.ball.graphics.beginFill(0xee00ee);
            this.ball.graphics.drawCircle(0, 0, this.ballRadius);
            this.ball.graphics.endFill();
            this.uiComponent.addChild(this.ball);
            this.drawingCanvas.addChild(this.uiComponent);
            this.resetBall();

            // Draw and place the incline.
            this.incline.graphics.beginFill(0xff0000);
```

```
            this.incline.graphics.moveTo(-this.inclineHeight/2, -
this.inclineHeight/2);
            this.incline.graphics.lineTo(-this.inclineHeight/2,
this.inclineHeight/2);
            this.incline.graphics.lineTo(this.inclineHeight/2,
this.inclineHeight/2);
            this.incline.graphics.lineTo(-this.inclineHeight/2, -
this.inclineHeight/2);
            this.uiComponent.addChild(this.incline);
            this.drawingCanvas.addChild(this.uiComponent);
            this.incline.x=150;
            this.incline.y=75;

            // Draw and place the spring.

            // Create a timer.
            this.myTimer=new Timer(25);
            this.myTimer.addEventListener(TimerEvent.TIMER, onTick);
            this.myTimer.start();
      }

      import mx.messaging.channels.SecureAMFChannel;

            public var netSocket : NetSocket;
            public var host : String = "*****"; // Not included for security
reasons.
            public var portNumber : int = 8000;
            public var userMaximum : int = 100;


            public function initChat() : void {

                  Security.loadPolicyFile("socket://" + this.host +
"/crossdomain.xml");
                  this.netSocket = new NetSocket();
                  this.netSocket.SetOnConnect(this.OnConnect);
                  this.netSocket.SetOnReceive(this.OnReceive);
                  this.netSocket.SetOnError(this.OnError);
            }
            public function Connect() : void {
                  if (this.tbName.text == "") {
                        var id : int = Math.random() * this.userMaximum;
                        this.tbName.text = "user" + id.toString();
                  }
                  this.lbStatus.text="Trying to Connect...";
                  this.netSocket.Initialize(this.host, this.portNumber);
                  if (!this.netSocket.Connect()) {
                        this.lbStatus.text = this.netSocket.statusMessage;
                  }

            }
            public function OnError() : void {
                  this.lbStatus.text = this.netSocket.statusMessage;
            }
            public function OnConnect() : void {
//                  this.lbStatus.text = this.netSocket.statusMessage;
                  this.lbStatus.text+="Welcome!";
```

```
                        initSimulation();
                }
                private function SendPosition():void
                {
//                      var
msg:String=this.ball.x.toString()+","+this.ball.y.toString();
                        var msg:String="<tm><type>position</type>"+
                        "<from>"+this.tbName.text+"</from>"+

        "<info>"+this.ball.x.toString()+","+this.ball.y.toString()+"</info>"+
                        "</tm>";
                        msg+="EOM";
                        Send(msg);
                }
                private function SendStopFollowing():void
                {
//                      var
msg:String=this.ball.x.toString()+","+this.ball.y.toString();
                        var msg:String="<tm><type>stopfollowing</type>"+
                        "<from>"+this.tbName.text+"</from>"+

        "<info>"+this.ball.x.toString()+","+this.ball.y.toString()+"</info>"+
                        "</tm>";
                        msg+="EOM";
                        Send(msg);
                }
                private function Send(text : String) : void {
                        this.netSocket.SendMessage(text);
                }
                private var messageCount:int=0;
                private function OnReceive(text : String) : void {
//                      if(this.isDragging)
//                      return;
//                      this.isFollowing=true;
                        var elements:Array=text.split("EOM");
                        var msg:XML=new XML(elements[0]);
                        var type:String=msg.type;
                        if(type=="position"){
                                ProcessPositionMsg(msg);
                        }
                        else if(type=="stopfollowing"){
                                ProcessStopFollowingMsg(msg);
                        }
                        if (this.tbHistory.text != "")
                                this.tbHistory.text += "\n";
                        this.tbHistory.text = messageCount.toString() + " : "+
this.ball.x.toString()+","+this.ball.y.toString()+"\n"+this.tbHistory.text;
                        messageCount++;

//                      if (this.tbHistory.text != "")
//                              this.tbHistory.text += "\n";
//                      this.tbHistory.text += text;
                }
        private function ProcessPositionMsg(msg:XML):void
        {
                var sender:String=msg.from;
                if(sender==this.tbName.text)
```

```
            return;
            var elements:Array=msg.info.split(",");
                  this.ball.x=parseInt(elements[0]);
                  this.ball.y=parseInt(elements[1]);
                  this.isFollowing=true;
      }
      private function ProcessStopFollowingMsg(msg:XML):void
      {
            var sender:String=msg.from;
            if(sender==this.tbName.text)
            return;
                  this.isFollowing=false;
      }
      private function onTick(evt:TimerEvent):void
      {
            if(this.isDragging){
                  SendPosition();
            return;
            }

            if(this.isFollowing)
            {
                  return;
            }
            // First start with ball movement.
            // Due to gravity, y-direction velocity changes.
            this.dxBall=this.dxBall;
            this.dyBall=this.dyBall+this.gravity*this.dtTime;

            // New position of ball.
            this.ball.x=this.ball.x+this.dxBall*this.dtTime;
            this.ball.y=this.ball.y+this.dyBall*this.dtTime;

            // First, to avoid infinite looping, conditions are applied to
small velocities.
            if(Math.abs(this.dxBall)<=.001)
            {
                  this.dxBall=0;
                  this.resetBall();
            }
            if(Math.abs(this.dyBall)<=.001)
            {
                  this.dyBall=0;
                  this.resetBall();
            }

            // Define what will happen if ball hits sides of the screen.
            if(this.ball.x-this.ballRadius<=0)
            {

      this.dxBall=this.coefficientRestitution*Math.abs(this.dxBall);
            }
            if(this.ball.x+this.ballRadius>=this.drawingCanvas.width)
            {
                  this.dxBall=-
this.coefficientRestitution*Math.abs(this.dxBall);
            }
```

```
            if(this.ball.y-this.ballRadius<=0)
            {

      this.dyBall=this.coefficientRestitution*Math.abs(this.dyBall);
            }
            if(this.ball.y+this.ballRadius>=this.drawingCanvas.height)
            {
                  this.dyBall=-
this.coefficientRestitution*Math.abs(this.dyBall);
            }

            // Add some friction to reduce velocity.
            if(this.ball.x-this.ballRadius<=1)
            this.dxBall=this.dxBall-.3*this.dxBall;
            if(this.ball.x+this.ballRadius>=this.drawingCanvas.width-1)
            this.dxBall=this.dxBall-.3*this.dxBall;
            if(this.ball.y-this.ballRadius<=1)
            this.dyBall=this.dyBall-.3*this.dyBall;
            if(this.ball.y+this.ballRadius>=this.drawingCanvas.height-1)
            this.dyBall=this.dyBall-.3*this.dyBall;

            // Define what will happen if ball collides with an incline.
//          if(ball.hitTestObject(incline)
//          {
//                var ballDistanceX:int=ball.x-incline.x;
//                var ballDistanceY:int=ball.y-incline.y;
//
//
//          }
//

            // Next, define what will happen if ball collides with a spring.

            // Move the ball.
            this.ball.x=this.ball.x+this.dxBall;
            this.ball.y=this.ball.y+this.dyBall;

            var formatter:NumberFormatter=new NumberFormatter();
            formatter.precision=1;

//          this.lbDebug.text=formatter.format(this.ball.x)+",
"+formatter.format(this.ball.y);


      }
      // Define function that resets ball after it stops.
      private function resetBall():void
      {
            this.ball.x=.25*this.drawingCanvas.width;
            this.ball.y=.25*this.drawingCanvas.height;
            this.dxBall=this.ballInitialVelocity;
            this.dyBall=this.ballInitialVelocity;
      }

      private function OnMouseDown(event:MouseEvent):void
      {
            var x:int=event.stageX-this.drawingCanvas.x;
```

```
            var y:int=event.stageY-this.drawingCanvas.y;
            this.lbDebug.text = x.toString() + ", " + y.toString()
            + ": " + this.ball.x.toString() + ", " + this.ball.y.toString();
            if(ball.x-ballRadius<=x&&x<=ball.x+ballRadius&&
            ball.y-ballRadius<=y&&y<=ball.y+ballRadius)
            {
                  this.isDragging=true;
                  this.lbDebug.text="Click";
            }
      }

      private function OnMouseUp(event:MouseEvent):void
      {
            this.isDragging=false;
            SendStopFollowing();
      }

      private function OnMouseOut(event:MouseEvent):void
      {
//          this.isDragging=false;
      }

      private function OnMouseMove(event:MouseEvent):void
      {
            if(!this.isDragging)
            return;
            var x:int=event.stageX-this.drawingCanvas.x;
            var y:int=event.stageY-this.drawingCanvas.y;
            this.ball.x=x;
            this.ball.y=y;
      }
            ]]>
      </mx:Script>


      <mx:MenuBar x="0" y="0" width="736" id="menuMain" labelField="@label">
      <mx:XMLList>
            <menuitem label="File">
                  <menuitem label="New"/>
                  <menuitem label="Open"/>
                  <menuitem label="Save"/>
            </menuitem>
            <menuitem label="Window">
                  <menuitem label="Chat Window"/>
                  <menuitem label="Objects"/>
            </menuitem>
      </mx:XMLList>

      </mx:MenuBar>
      <mx:VBox x="10" y="131" height="253" width="85" id="boxObjects">
            <mx:Label text="Select Object:" width="84" id="boxLabelObject"
visible="true" selectable="false" color="#14387B"/>
            <mx:Button label="Add Ball" id="buttonBall" visible="true"/>
            <mx:Button label="Add Incline" id="buttonIncline" visible="true"
labelPlacement="left"/>
            <mx:Button label="Add Spring" id="buttonSpring" visible="true"/>
      </mx:VBox>
```

```
      <mx:Canvas width="600" height="300"
            borderStyle="solid" backgroundColor="#FAF6F6"
            id="drawingCanvas" horizontalCenter="0" top="60"
mouseDown="OnMouseDown(event)" mouseMove="OnMouseMove(event)"
mouseOut="OnMouseOut(event)" mouseUp="OnMouseUp(event)">
            <mx:Label x="10" y="10" text="Debug" id="lbDebug"
color="#E60D36"/>
      </mx:Canvas>
      <mx:TextInput x="10" y="30" id="tbName"/>
      <mx:Button x="178" y="30" label="Connect" id="bConnect"
click="Connect()"/>
      <mx:Label x="103" y="368" text="Status" id="lbStatus"/>
      <mx:TextArea x="10" y="392" height="160" id="tbHistory" width="460"/>

</mx:Application>
```