Mobile Simulation with applications for Serious Gaming

Andrew Jeffery **Jonathon Panke** **Nick Eaket** **Gabriel Wainer**

Department of Systems and Computer Engineering
Carleton University, Ottawa, ON Canada
jeffery.andrew@gmail.com, jonathonjpanke@gmail.com, neaket@gmail.com, gwainer@sce.carleton.ca

Keywords: Discrete event simulation, DEVS, Smart Phones, serious games

Abstract

We discuss the design and development of an infrastructure for serious gaming applications using a formal Modeling and Simulation environment based on the DEVS formalism. The research uses the RISE simulation middleware services to build a serious game application that can be deployed on a mobile device using a properly developed mobile client. We propose a mobile client that leverages the cloud services provide by a RISE server to obtain simulation data to be used in a serious gaming application. In particular, our prototype provides a stock trading game. The simulation model is based on a Brownian motion economic model that has been analyzed comparatively to real data.

1. INTRODUCTION

With the growth of network connectivity in past years there has been a dramatic increase in the applications and services that are provided for mobile devices [1]. Additionally, in order to overcome the hardware processing limitations of current technology new methodologies and design architectures have emerged. Such technologies, which include distributed systems and cloud services, make it possible to bring complex, process heavy simulation to mobile devices. As further developments in mobile communications arise new tools are being deployed on mobile platforms. When taking into the consideration the increasing popularity of such mobile tools and applications in younger generations, there is a large demand to try to incorporate mobile devices in teaching practices. It has been suggested that the best way to do this is using MSGs (Mobile Serious Games) as explained in [2].

The idea of serious gaming is to provide an environment analogous to reality. In such an environment a problem is presented to the participants, which they must solve using similar techniques and knowledge that would be required if the event occurring was real. This allows for the development and practice of the required skills before they are necessary in the field.

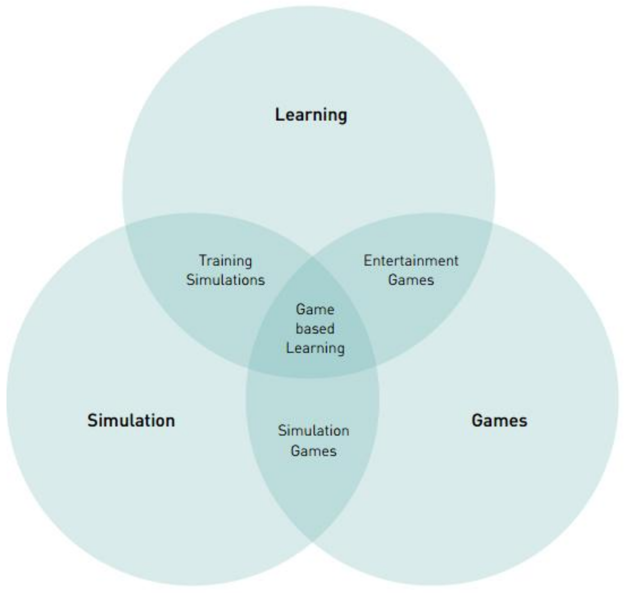


Figure 1 Relationships between Games, Simulation and Learning. [3]

In order to design a serious game successfully, many aspects are required. One of the largest aspects of serious games we are interested in is simulation. Many existing serious games are based on game engines that do not provide real simulation capabilities (instead, they provide basic training scenarios). Instead the goal of incorporating simulation in the serious game is to emulate a real life environment that follows specific rules that related to the real world. The mixture of simulation and learning within the format of a game is shown in Figure 1, the *game based learning* region is the area that incorporates a serious game. Outside of a game format many of the simulation aspects and requirements that serious games implement are found in training simulations, these training simulations normally focus more on their realism over their fun factor, thus not being classified as games.

Applying the idea of serious gaming to a video game normally requires some level of simulation from the application in order to obtain the proper data for events that are occurring in the virtual system. One technique for this is Web-based simulation which has become popular in conducting online simulations. The idea is to use a Web server to implement the networked architecture using a particular architecture (client-server, High Level Architecture, CORBA, RPC, etc) [4]. Many advanced implementations have been built on SOAP Web Services to communicate [5] [6]. These simulation middlewares are complex to interoperate, and their composition scalability is limited. Instead, the Representational State Transefer style (REST) can solve these issues by imitating the Web interoperability style. RESTful Web Services focus on the resources more than on the operations, solving the interoperability limitation and making easy the development of Mash-Ups [7] (which reuse and combine existing services to build a new web application).

Since this type of simulation can be computationally heavy, these types of games traditionally have been limited to hardware not found in mobile devices. Yet with the developments in cloud computing and services, there now exists systems that can provide simulation to a remote user as a cloud or web service. The RISE simulation environment provides an API to access its cloud services using RESTful commands to run simulation models remotely and retrieve the simulation results. In this article, we present a mobile Smartphone client that uses simulation of a Stock Market environment and the inclusion of player driven missions to learn about the basics of stock trading. The client titled Stock Market Tycoon was originally developed with the desire to bring simulation data done on a distributed server to mobile devices. This later turned into a MSG with the goal of having missions for the user to achieve. The successful development of a client that meets the goals of allowing a user to have access to a simulation model remote and the results based on dynamically varying input values was achieved.

2. BACKGROUND

Serious gaming has evolved from what it used to be with war games, and board games. It has entered into the digital world, which has led to techniques in designing serious games. Today serious games for the financial stock market are very scarce. A few examples of modern games with the stock market are: Wall Street Survivor, MarketWatch, and UpDown [8] [9] [10].All three of these web browser based games use real stock values in an attempt to teach their users about investing. These differ from our application greatly due to their lack of simulation (and they also need connection to the Internet at all times). Another example of similar work is found in [11] which demonstrates a simulation engine for a stock options game. Ours differs greatly due to ours being a serious game, and the patented engine is for a normal game which not based on the actual economic behavior of real stocks. Another difference is our focus on presenting the simulation on a mobile device, simulation on a mobile device, mobile simulation in this way is discussed in [12]. Our client focuses more on using the application as a serious game, allowing the use of dynamic data for the simulations.

The goal of any Distributed Simulation middleware is to interlace different simulation environments, allowing synchronization for a simulation that is ran across a distributed network [13]. RISE is the first RESTful distributed simulation environment, which was used in this project to allow for cloud computing services, and allowing multiple users able to run simulations at the same time.

As discussed in [12] and [14], RISE allows plug-and-play interoperability for simulation tools by decoupling the services from the formalism. RISE provides a general purpose API to interface different simulators, allowing different clients and enabling varied experimental frameworks with a number of instances with different settings. RISE exposes the tools through a set of URIs, and when a client makes a request, it uses a specific URI. RISE looks for the resource, and sends the request to it, collecting and forwarding the response to the client. The clients connect to the URIs via HTTP channels. They use GET operations to read resource data, PUT operations to create new resources or update them, and DELETE to remove a resource. For example, a simple session could include the following sequence [12]:

1. `PUT http://.../cdpp/workspaces/bob/DCDpp/model`, to create a model in the workspace "*bob*";
2. `POST http://.../cdpp/workspaces/bob/DCDpp/model?zdir=files`, to submit the model files;
3. `PUT http://.../cdpp/workspaces/bob/DCDpp/model/simulation`, to start the simulation;
4. `GET http://.../cdpp/workspaces/bob/DCDpp/model/results`, to download the results

To use the RISE server an API is provided which uses REST commands. Table 1 includes a summary of some of the services available, which can be used for simulation purposes.

Table 1. REST API with Messaging [14]

Action	Channel	HTTP Success Code	Message
Create Framework	PUT	201 (Created)	XML
Configure Framework	PUT	200 (OK)	XML
Submit Model	POST	200 (OK)	.zip file
Delete Framework	DELETE	200 (OK)	None
Start Simulation	PUT	202 (Accepted)	None
Stop/Abort Simulation	DELETE	200 (OK)	None
Check Simulation	GET	200 (OK)	XML
Download Results	GET	200 (OK)	.zip file

RISE can be instantiated with multiple simulators; and in this case, we have used the default container: Discrete-event Systems Specification (DEVS) models created to be run in the CD++ simulation toolkit [18]. DEVS [15] is a formalism used in the modeling of both continuous and discrete worlds. Basic DEVS models (called **atomic**) are specified as black boxes, and several DEVS models can be integrated together forming a structural model (called **coupled**).

A DEVS atomic model can be defined as:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

where **X** represents a set of input events, **S** a set of states, and **Y** is the set of output events. Four functions manage the model behavior: δ_{int} the internal transitions, δ_{ext} the external transitions, λ the outputs, and **D** the duration of a state. Each model uses its input and output ports to communicate with other models. External events are received via the input ports, and the model defines its behavior upon the reception of such inputs. The internal events produce state changes, and results are communicated to its influences using the output ports.

A DEVS coupled model can be defined as:

$$CM = < X, Y, D, \{Mi\}, \{I_i\}, \{Z_{ij}\} >.$$

where **X** is the set of input events, **Y** is the set of output events, **D** is an index for the components of the coupled model, and for every i in D, $\mathbf{M_i}$ is a basic DEVS component (*i.e.*, an atomic or coupled model). The list of influencees $\mathbf{I_i}$ of a given model is used to determine the models to which outputs must be sent. The function $\mathbf{Z_{ij}}$ translates outputs of a model into inputs for the other models. An index of influencees is created for each model ($\mathbf{I_i}$). For every j in the index, outputs of model $\mathbf{M_i}$ are connected to inputs in model $\mathbf{M_j}$.

As we can see, DEVS can be used to model a system whose states change based on the expiration of a time delay, or due to an input event, which with a stock market environment encompasses both these. An input event for a stock market would be a random event that affects how a stock changes. DEVS models are a simplified structure of what it represents in reality. The models are built upon the experimental expectations of the system; these constraints being the working conditions and its application domain. These constraints are a composition of atomic or coupled components. Coupled models are just a collection of atomic models or coupled sub models. This allows for the creation of complex models.

CD++ [18] is a toolkit, which was developed for the use of modeling simulation environments using DEVS formalisms. CD++ has support for creating two types of models, behavioral and structural models. CD++ is built as a class hierarchy of models related with simulation processing entities. DEVS Atomic models can be programmed and incorporated onto the *Model* basic class hierarchy using C++. A new atomic model is created as a new class that inherits from the *Atomic* base class. The state of a model is defined in the *AtomicState* class. When creating a new atomic model, a new class derived from *Atomic* has to be created.

```
class Atomic : public Model  {
public:
virtual ~Atomic();     // Destructor

protected:
//Kernel services
Time nextChange();
Time lastChange();
holdIn(AtomicState::State &, Time &);
passivate();
ModelState* getCurrentState() ;
sendOutput(Time &time, Port &port, Value value);

//User defined functions.
initFunction();
externalFunction(ExternalMessage & );
internalFunction(InternalMessage & );
outputFunction(CollectMessage & );
string className() const
}; // class Atomic
```
Figure 2. The Atomic Class

Atomic is an abstract class that declares a model's API and defines some service functions the user can use to write the model. The *Atomic* class provides a set of services and requires a set of functions to be redefined:

- nextChange()/lastChange(): return the time until the next internal transition/since the last state change.

- holdIn(state, Time): tells the simulator that the model remains in a *state* during a given *Time.* It corresponds to the ta(s) function of DEVS.

- passivate(): sets the next internal transition time to infinity. The model will only be activated again if an external event is received.

- getCurrentState(): returns current model's phase

- sendOutput(Time, port, value): sends an output message through the specified *port*.

The new class should override the following functions:

- initFunction(): method invoked by the simulator at the beginning the simulation.

- externalFunction(ExternalMessage &): method invoked when an external event arrives to a port. It corresponds to the δ_{ext} function of the DEVS formalism.

- internalFunction(InternalMessage &): method defining the δ_{int} function of the DEVS formalism.

- outputFunction(const CollectMessage&): in charge of transmitting the output events of the model. It corresponds to the λ function of the DEVS formalism.

Once an atomic model is defined, it can be combined with others into a multicomponent model using a specification language specially defined with this purpose. The coupled model at the higher level is always named [top]. Four properties must be configured: components, output ports, input ports and links between models. The following syntax is used:

```
Components: name1[@atomicClass1] name2 ...
```

This sentence lists the components of the coupled model (atomic or coupled). For atomic models, an instance and a class name must be specified, allowing a coupled model to use more than one instance of a given atomic class. For coupled models, only the model name must be given, and it must be defined as another group in the same file.

```
In: portname1 portname2 ...
Out: portname1 portname2 ...
```

These sentences enumerate the model's input/output ports (optional clause).

```
Link:source[@model] destination[@model].
```

This clause describes the internal and external coupling scheme. If the name of the model is not included, the default will be the coupled model currently being defined.

Since a system is decomposed into some number of modular sub-systems, this makes interconnecting models easy and provides communication of data from model to model. This also allows models to be developed in iterations since additional behavior can be added in the form of an additional module (i.e. another atomic model). An advantage of using RISE is that the user spaces on the servers are broken up into domains and frameworks, where there is no interference or shortage of resources between them. This is useful because it allows for multiple users to have their own unique data which can't be tampered with by other users. If many users are requesting simulations to be executed at the same time, each of them doesn't have to wait for someone to free up resources.

A complete description of RISE can be found in [16] and [17]. The most common way to use a simulator remotely is by designing a web-based tool. Byrne, Heaveya and Byrne in [19] provide an extensive review of web-based simulators, discussing the advantages and disadvantages of the client/server pattern applied to the simulation, and analyzing how and with which technologies it can be implemented. In the light of their paper, our work uses a hybrid simulation and visualization approach to drive a remote managed simulation (the client grabs the raw data and visualizes them).

3. SIMULATING BROWNIAN MOTION

Simulating a stock market environment can be achieved in many different ways. One of the most common approaches is using Brownian motion [20], an introductory method to calculating trends in assets. The mobile application prototype we have built uses Brownian motion, and more specifically deterministic Brownian. The deterministic approach allows for the prediction of the assets in a manner that doesn't allow for risk. The formula used for our model is [20]:

$$\frac{\Delta S}{\Delta t} = rS_o$$
$$S_T = S_o + rS_o\Delta t$$

where the variables are $\Delta S = S_0 - S_T$, the change in stock price, S_T is the new Stock Price. S_0 is defined as the initial Stock Price and r is the Random Normalized Variable. Note that Δt is the difference in time between S_T and S_0 as a fraction of a year.

The model receives the initial stock price at the current time, and it then calculates the predicted stock value for the next 24 hours at one hour intervals by default. The time interval between stock prices can be changed in the model to allow for finer or broader intervals. In order to track the data of multiple stocks, each stock has an integer stock index that is unique so that data sets may be properly associated with their stocks.

In order to have dynamic data, the client running on the Smartphone creates an event file that lists the events and the time of occurrence that the model utilizes when it runs the simulation. Each event must specify the time it will occur at during the simulation, the port that the event will be sent to in the model and the value of the event. An example of an event (.ev) file is given in figure 3. In this example there are 3 events that will occur during the simulation. The events occur at hour 01:00:00, 02:00:00, and 03:00:00 as stated in the event file. Each event occurs on the stated port, in this case all the events occur on the same port (`InStockIndex`), note that the port must be name and exist so that events can be entered correctly into the simulation. The final entry is the value that is passed to the requested port. This value is then used by the simulation when the event occurs as specified by the model used. The creation of event files is the core mechanic that is used to provide information to the simulation server from the mobile client.

```
00:01:00:00 InStockIndex 004567
00:02:00:00 InStockIndex 003678
00:03:00:00 InStockIndex 234568
```

Figure 3 Example of the Event File

On the successful creation of the .ev file, the file is compressed and uploaded to the RISE server via the POST command [12]. After the file is uploaded to the required simulation framework, the simulation can be initialed at any time by using the REST command PUT [12]. Once the simulation completes the simulation results are available as a zip file that may be retrieved until either the simulation is executed again or the framework is deleted. An example of these commands follows as:

`POST http://.../cdpp/workspaces/andrew/DCDpp/Brownian?zdir=Brownian`, to submit the model files to the Brownian model framework;

`PUT http://.../cdpp/workspaces/andrew/DCDpp/Brownian/simulation`, to start the simulation;

`GET http://.../cdpp/workspaces/bob/DCDpp/Brownian/results`, to download the results after simulation is completed.

The status of the simulation can be checked using the command:

`GET http://.../cdpp/workspaces/andrew/DCDpp/Brownian/status`, to submit the model files;

To show the realism in a Brownian motion simulated stock environment, simulations where compared to real world data. The real world stock example chosen for comparison was the well-known company BlackBerry (BBRY). The graph in figure 4 is the daily stock graph for March 20th, 2013. While looking at the stock graph note that the market opens 9:30 EST and closes at 4:00 EST, and the graph plots the points in 5 minute intervals. The graph shows the BBRY stock opening with a price of \$16.25 a stock and closing with a price of \$16.53. The letters on the stock graph are news events that correlate with the BBRY stock for that day.

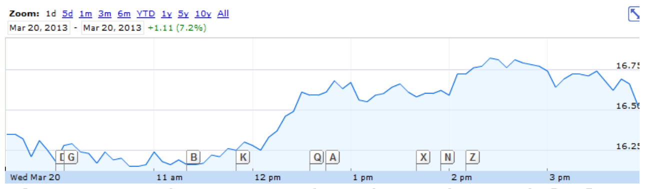


Figure 4 BlackBerry March 20th Stock Graph [21]

To test and validate the effectiveness of the deterministic Brownian motion DEVS model that was implemented two variables had to be calculated – the change in time and the number of data points for a given day. For this we got a value of Δt to be ~0.0042 which equates to a 5 minute interval of time in accordance to a year, and 84 data points to mimic a normal stock day. Using these numbers to generate 5 trials, the simulation results can be seen in figure 5.

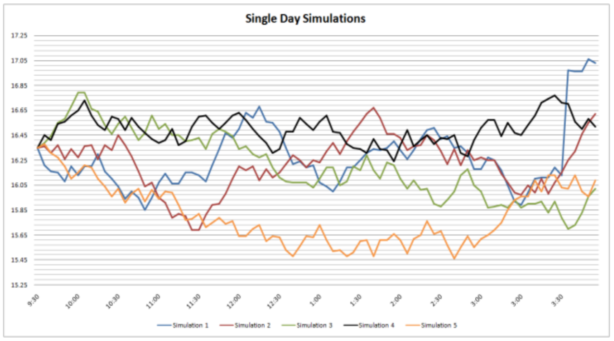


Figure 5 Multiple simulation runs mocking Blackberry's daily stock graph

Based on a visual inspection of the results, it can be seen that some trials produce similar behaviour, while a few others deviate slightly. But the closing stock prices are within the 10% margin of the realistic value. Further data analysis is summarized in table 2.

Table 2 Simulation Trial Statistics

Trial	Average Mean (μ)	Standard Deviation (σ)	10% C.I.	5% C.I.
1	16.27	0.25	±0.054	±0.071
2	16.23	0.21	±0.046	±0.061
3	16.21	0.27	±0.057	±0.076
4	16.49	0.11	±0.024	±0.032
5	15.82	0.24	±0.051	±0.063

Based on the result in table 2, it can be seem that the random data produced falls in the region of the realistic data as can be seen by the trial averages and the range of their 10% confidence intervals. This behaviour displayed is reminiscent of a realistic stock price.

4. IMPLEMENTATION

The application is based on a client-server architecture. A main piece of any client-server architecture is the communication protocol used to handles all the messaging and communication between the two entities. The communication module implemented in this project is the communication link for the mobile client and the RISE server, which handles all messaging between the two entities. The communication module has a number of responsibilities that assist that client aside from communicating with the RISE server. The system was organized into 3 subsystems during development, the GUI, the Phone Client engine, and the RISE server. An overview of the entire system is given in the deployment diagram in figure 6.

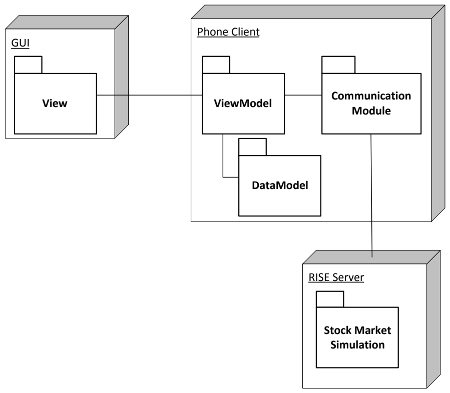


Figure 6 System deployment diagram

The ViewModel governs the interactions between the GUI and the communications from the RISE server on the locally stored data. The data model is used to store and retrieve data in a thread safe manner and the Communication Module tracks the state of the server and responds to asynchronous messages from the ViewModel and the RISE server. Due to this asynchronous nature and that each stock update requires a number of stages to complete in sequence, the Communication Module implemented a state machine to track the process of each update as shown in figure 7. Each state of the communication state machine represents a stage in the communication protocol that the client and server execute in order to obtain new data. The data returned is dependent on the chosen model of simulation, and the client determines which model it is using before communication begins so that it can signal the RISE server to setup the appropriate framework.

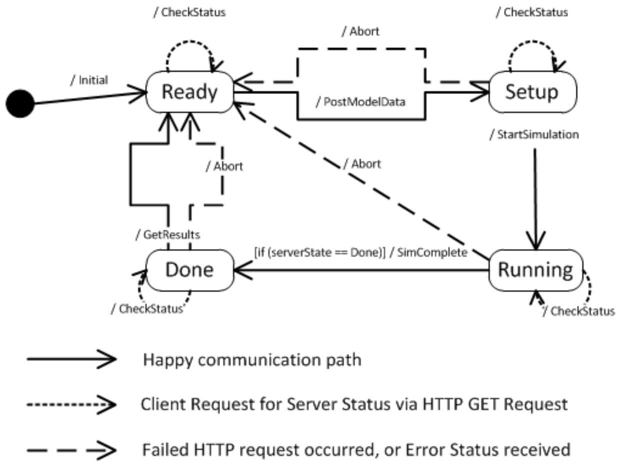


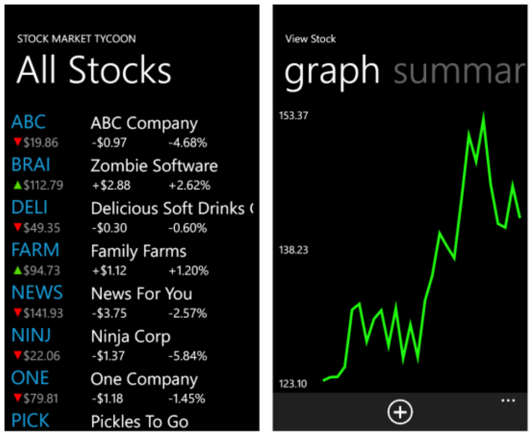
Figure 7 Application State Diagram

There are four states to the communication state machine. The **Ready** state is the initial state when the state machine is instantiated. In the **Ready** state there is no activity occurring on either the client or server side; it is from this state that any communication begins. The **Setup** state indicates the framework has been initialized with the selected model and has pending data for a simulation. The **Running** state indicates no other data requests can be sent, since a simulation is currently running. The **Done** state indicates that there is pending simulation data that needs to be collected by the client. The transitions refer to the actions that the communication goes through and reflect a change in the state of the RISE server. The `PostModelData` transition occurs when the mobile client creates a model framework and instantiates the correct model with an event file. The `StartSimulation` transition sends a PUT command to the RISE server to start the simulation and on the mobile client a new thread is created to monitor the simulation status. The `SimComplete` transition has the guard that the server must have completed the simulation, if true the data retrieval and parsing objects are loaded so that on the retrieval of the results they can be updated in the DataModel. The `GetResults` transition fetches the data from the server and uses the loaded parser to read and sort the data, which afterward is updated in the DataModel. This method is used for communication to the server by the client to obtain new simulation data.

5. APPLICATION AS A SERIOUS GAME

The stock simulations are executed remotely on the RISE server in the cloud. This allows a Smartphone to run complex, process and CPU intense simulations due to the use of the cloud services provided by the RISE API. The mobile client was implemented as a Windows Phone Client application which goal was to distribute the more CPU intense task of simulation to the RISE server for the purposes of simulation a stock market environment. The initial data was sent using the REST commands that are specified by the RISE server documentation via compressed zip files. The Mobile client was then able to retrieve the simulation data to use as the basis for a stock market trading simulator called “Stock Market Tycoon” or SMT.

The two iconic screens of any stock market profile are the recent history and the summary of current statistics view. These where both included in the application to help improve the emersion of the player in the game. Since the format of these views is commonly the same, the phone client attempted to reuse the most common elements as seen in figure 8. The two most common elements where the line graph of recent history and the color coded gain or loss arrows on the market screen for at a glance analysis.


STOCK MARKET TYCOON

All Stocks

ABC	ABC Company		
▼$19.86	-$0.97	-4.68%	
BRAI	Zombie Software		
▲$112.79	+$2.88	+2.62%	
DELI	Delicious Soft Drinks (		
▼$49.35	-$0.30	-0.60%	
FARM	Family Farms		
▲$94.73	+$1.12	+1.20%	
NEWS	News For You		
▼$141.93	-$3.75	-2.57%	
NINJ	Ninja Corp		
▼$22.06	-$1.37	-5.84%	
ONE	One Company		
▼$79.81	-$1.18	-1.45%	
PICK	Pickles To Go		

Figure 8 (Left) Market available in SMT client (Right) View of recent stock behavior

Stock Market Tycoon uses the simulated data to present to the user the options of trading stock for profit and browsing a small selection of stocks that were present in a traditional stock market format. This format and the use of the simulation data provides a recreation of the same environment that is present in a real stock market except for the real-time aspect since time progression of the market on the phone client is at the user’s discretion. Therefore the client application implements the target attributes of a serious game due to it’s the recreation of a real system based on a simulation model.

6. CONCLUSION AND FUTURE WORK

For the recreation of a system in the format of a serious game, there is the possibility for game based learning. But despite the educational content of a game, if the recreation of the real system is valid then the game constitutes a serious game. Serious games require a set of rule or model in order to recreate the environment and events that a system is composed of. In order to accomplish this in a video game the most common method is via simulation.

The hardware limitations of mobile devices present a barrier to heavy computational simulations. What we have presented is an implementation that leverages web services provided by the RISE API to run simulation models on a cloud server architecture, this frees the mobile device to focus on user interaction. By separating the implementation of a serious game into two areas, simulation and user interaction, the client server design for the mobile application aims to fit the appropriate hardware for the appropriate task. The more hardware limited smart phone runs it application with its focus on the user experience and communication. While the more hardware sophisticated RISE server can run complex models developed to accurately represent the stock market system or other environments of the choice. Some areas for future work are the creation of a more complex stock market environment by expanding on the existing Brownian model. Furthermore this technique can be expanded to different systems and a more generalized client or communication engine can be pursued.

REFERENCES

[1] B. M. Cunningham, P. J. Alexander and A. Candeub, "Network Growth: Theory and Evidence from the Mobile Telephone Industry," *Information Economics and Policy,* vol. 22, no. 1, pp. 91-102, 2010.

[2] J. Sanchez, C. Mendoza and A. Salinas, "Mobile Serious Games for Collaborative Problem Solving," IOS Press, Chile, 2009.

[3] M. Ulicsak, "Games in Education: Serious Games," Futurelab, Slough, 2010.

[4] G. Wainer and K. Al-Zoubi, "An Introduction to distributed simulation," in *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*, C.Banks, J. Sokolowski, Eds., Wiley, 2010.

[5] S. Mittal, J. L. Risco-Martín and B. P. Zeigler, *DEVS/SOA: A Cross-Platform Framework for Net-centric Modeling and Simulation in DEVS Unified Process*, *SIMULATION,* vol. 85, no. 7, pp. 419-450, July 2009.

[6] A. Boukerche, F. M. Iwasaki, R. B. Araujo and E. B. Pizzolato, "Web-Based Distributed Simulations Visualization and Control with HLA and Web Services," in *Proc. of the 2008 12th IEEE/ACM DS-RT*, Washington, DC, USA, 2008.

[7] Y. Harzallah, V. Michel, Q. Liu and G. Wainer, "Distributed Simulation and Web Map Mash-Up for Forest Fire Spread," in *Proceedings of the 2008 IEEE Congress on Services*, Washington, DC, USA, 2008.

[8] "Wall Street Survivor," [Online]. Available: www.wallstreetsurvivor.com/dashboard. [Accessed 16 April 2013].

[9] "UpDown," [Online]. Available: www.updown.com. [Accessed 16 April 2013].

[10] "Market Watch," The Wall Street Journal, [Online]. Available: www.marketwatch.com/game/fet. [Accessed 16 April 2013].

[11] K. e. al., "Stock Simulation Engine For An Options Trading Game". US Patent 6709330, 24 March 2004.

[12] E. Mancini, G. Wainer, K. Al-Zoubi and O. Dalle, "Simulation in the Cloud Using Handheld Devices," in *MSGC' 12 Workshop*, CCGRID, 2012.

[13] G. Wainer and K. Al-Zoubi, "Using REST Web-Services Architecture for Distributed Simulation," in *ACM/IEEE/SCS 23rd Workshop on Principles of Advanced and Distributed Simulation*, Lake Placid, 2009.

[14] G. W. Khaldoon Al-Zoubi, "RISE: REST-ing Heterogenious Simulations Interoperability," in *Winter Simulations Conference*, Ottawa, 2010.

[15] B.P. Zeigler, H. Praehofer, T.G.Kim. *Theory of Modeling and Simulation*. 2nd Edition. Acadamic Press. 2000.

[16] K. Al-Zoubi and G. Wainer, "Rise: Rest-ing heterogeneous simulations interoperability," in *Proceedings of the 2010 Winter Simulation Conference (WSC)*, Baltimore, MD, 2010.

[17] K. Al-Zoubi and G. Wainer, "Distributed Simulation Using RESTful Interoperability Simulation Environment (RISE) Middleware," in *Intelligence-Based Systems Engineering*, Springer Berlin Heidelberg, 2011, pp. 129-157.

[18] G. Wainer, *Discrete-Event Modeling and Simulation: A Practitioner's Approach*, P. Mosterman, Ed., Taylor and Francis, 2009.

[19] J. Byrne, C. Heaveya and P. Byrne, "A review of Web-based simulation and supporting tools," *Simulation Modelling Practice and Theory,* vol. 18, no. 3, pp. 253-276, 2010.

[20] S. R. Starja, *Stochastic Modeling of Stock Prices*, Montgomery Investment Technology Inc., New Jersey.

[21] "Research in Motion Ltd.," Google Finance, [Online]. Available: https://www.google.com/finance?q=BBRY&hl=en&ei=b15vUbirI8THqAHHswE. [Accessed 15 April 2013].