An XML-based Microworld Simulator for Business Modeling Education

Yoshiharu Kato
Faculty of Economics, Musashi University
kato@cc.musashi.ac.jp

Abstract

This work addresses the education of business modeling for the college students, and is intended to teach the basic knowledge and skills necessary for the business modeling by combing introductory programming, UML modeling, XML marking-up, and computer simulation using a graphical modeling simulator. The students can design and build their own worlds using an XML-based modeling language which allows the visualization of the world as well as the description of the behavior of the objects. The language consists of two layers; the basic layer provides the general scripting functions such as variables, if, and do-while, and the world layer provides the specialized scripting functions such as say, move, and route. The visualization is provided by a simple game-like animation in an isometric view. By constructing and simulating their own worlds, the students can learn the basics of business modeling. This paper presents the overview of the work as well as the design and implementation of the simulator.

1. Introduction

The understanding of the modeling methodology is gaining importance even for the students majoring in economics or business administration. The obvious reason is the deep penetration of Information and Communications Technology in our business environment. Another reason is the increasing popularity of both XML and UML. Therefore, the students are aware of the importance of business modeling. However, the modeling methodology cannot effectively be learned without exercising actual modeling.

In this paper, we address the problem of teaching business modeling to the students who have little or no prior programming experience. The method consists of four phases. The first phase is for the students to learn visual programming using Squeak[1]. The purpose of this phase is threefold: to familiarize with programming, to stimulate the imagination, and to learn how Squeak's visual world simulates the real world. The second phase is to learn about UML and understand how it can be used to describe business models[2][3]. The third phase is to build an own tiny world using a simulator developed specifically for this purpose. The forth phase is to enhance the world so that more details can be simulated.

The simulator is an XML-based game-like environment with which the students can build own tiny worlds and observe the results presented in an isometric view. The modeling language is named Virtual World Modeling Language (VWML), and can define the appearance as well as the behavior of the objects. A typical example is to create a tiny world in which several agents are living, working, and shopping. The objects can contain other objects, for example, to model the inside of a house. The agents can leave one place to visit other place.

To create their own models, the students need to observe the real world, extract the representative features, develop simplified world plans, draw necessary graphics, write VWML scripts, run the simulator, observe the behavior of the blocks, debug the scripts, and refine the world. From the series of these experiences they can acquire the basics of the modeling, learn the use of XML, and improve the skill of logical/algorithmic thinking.

The simulator is written entirely with Java, and is intended to be a small program providing only minimum functionality so that the interested students can learn from its code.

In this paper the overview of the work as well as the design and implementation of the simulator will be presented.

2. Background

2.1. Difficulties in learning business modeling

Computers are no longer the machines to work only on numbers. Instead, they are our agents, for example,

Proceedings of the Fourth International Conference on Creating, Connecting and Collaborating through Computing (C5'06)
0-7695-2563-6/06 $20.00 © 2006 IEEE

to perform complex business processes over the Internet, 24 hours a day, 7 days a week. Therefore, the business modeling is gaining much importance. Also, the importance of the business modeling is well recognized by major organizations due to the emergence of Enterprise Architecture[4]. Therefore, learning the business modeling is becoming important even for the students majoring in economics or business administration. The learning program may include the concept and framework of business modeling as well as the use of UML and XML for documentation, visualization, and construction of business models.

Modeling needs much skills and experiences to learn because it is based on the extraction and abstraction. The students experienced in the object-oriented programming may not have difficulties in learning the business modeling. However, the students majoring in economics or business administration usually have little or no programming experiences or computer science background. Besides, it is sometimes difficult to motivate those students to learn even an introductory programming because they think programming is a difficult task and is not a fun. Even after they began to learn programming, they might loose their interest in programming because there is a large gap between the example programs to be learned during the course and the sophisticated commercial programs available on their desktops. For these reasons the education of the business modeling is getting importance but is rater difficult.

2.2. Model verification

When analyzing a certain business model and illustrating it in UML diagrams, the students often encounter various variations because modeling is not a straightforward process and the solution is not unique. Although there are many UML tools available, those tools mainly verify the consistency among diagrams and have limited execution capability. Therefore, it will be worthwhile to provide some simulation environment in which the models can be constructed and executed by manually translating UML diagrams into some simulation code. The simulator may provide visual interface for ease of modeling as seen in many simulation games. Or, it may provide a modeling language so that the students can describe the internal of the models and acquire the basic programming skills through the simulation activities. This work has taken the latter approach.

2.3. Related work

The computer simulation has extensively been used to study the social and economic issues by building computational models[5, for example]. This social simulation takes many different approaches including a multi-agent approach. Our approach is based on the agent technology, but is not intended to simulate a large society.

There are a large number of commercial and free game programs which simulate the construction and management of various kinds of worlds. The primary purpose of those programs is the growth of the simulated world. A game player has a greater degree of freedom in model construction, but usually has little or no knowledge of the internal mechanism because game characteristics are usually set by various parameters which will be interpreted by the game engines. In contrast, our approach is to program the behaviors of all the objects so that students can understand all the behaviors with no black boxes and can verify the validity of their business models.

Scripting languages have been used for the development of games[6][7]. The languages are usually unique to particular games or toolkits, but there are also some common scripting languages such as Lua[8]. Our approach is to use an XML-based scripting language so that a specialized language for the business modeling can be designed and, at the same time, the students can learn about XML. From the implementation viewpoint the use of XML has an advantage of simplifying the parser and the script interpreter.

3. Modeling education

The presented method focuses on the integration of modeling and simulation. Modeling is not a simple process and the models cannot easily be verified. Modeling tools are effective in making the diagrams, and some of the tools can generate code skeleton. However, the generated skeleton needs some amount of coding before it can be executed. Simulation is a powerful tool in analyzing complex systems. On the other hand, the social simulation, for example, is based on complex mathematical model, and is not well-suited to our needs. Therefore, the proposed method combines both modeling and simulation to provide a specialized environment so that the students can design simple models using UML and then create objects for simulation and verification. This combination provides a bridge between modeling and simulation. The visualization allows the model verification and provides a fun. Students will be motivated in learning modeling, simulation, UML, and XML. They will also

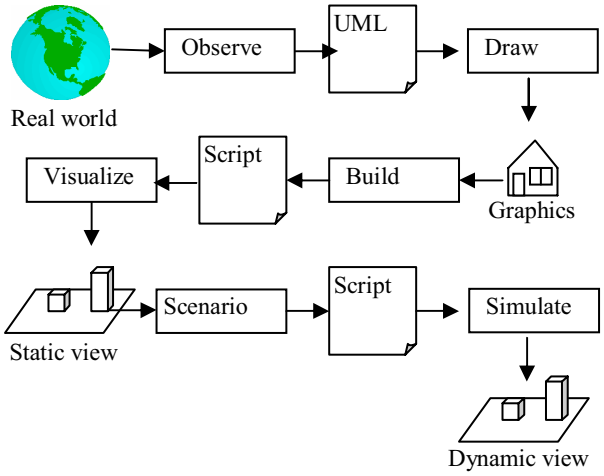

Figure 1. The modeling sequence.
Modeling begins with the observation of the real world, and UML diagrams are prepared to illustrate the static relationships among objects as well as the dynamic relationships. Then, drawing the exterior and interior of each object creates a set of bitmap graphics. Building the microworld needs a set of scripts which define the relationship among objects. Visualization is done by the simulator which presents the static view of the world. Scripting defines the behavior of the necessary objects. Finally, simulation is performed by the simulator which interprets the scripts in a step-by-step fashion to present the dynamic view of the world.

be interested in learning programming, for example, to understand the internal of the simulator.

At the completion of the course, it is expected that they will be able to:

1) develop a simple business model by observing the real world,
2) draw the UML diagrams such as class diagrams, statechart diagrams, and sequence diagrams to illustrate the business model,
3) draw the simple graphics for the construction of the business model,
4) develop a scenario by writing and debugging simple scripts, and
5) verify the simulation result and revise the scenario.

The proposed method is based on the sequence shown in Figure 1.

However, because the students have no prior programming experiences, it is necessary to include an introductory programming at the beginning of the course. Therefore, the presented method consists of four phases; the first phase is to learn the visual programming with Squeak, the second phase is to learn UML and its usage in business modeling[3], the third phase is to create an own world using VWML and

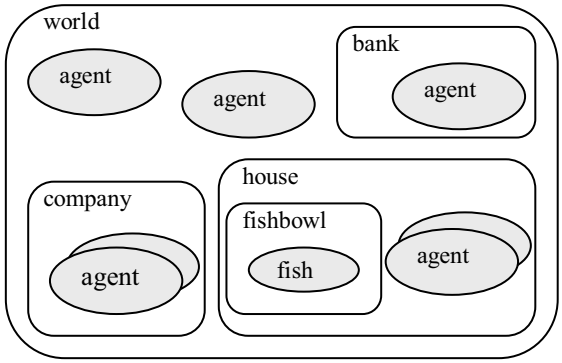

Figure 2. An example of the world.
The top-level world is composed of multiple objects called blocks, and each block may, in turn, be composed of multiple blocks. Each block's behavior is defined by a script. There is no distinction between a movable block (e.g., an agent) and a non-movable block (e.g., a house) from the simulation viewpoint.

simulate the model, and the fourth phase is to enhance the world. This method focuses on the modeling skills as well as the introductory programming, graphics, simulation, Squeak, UML, and XML.

The world to be built here is not a large, complex model. Instead, it should be a simple, familiar model which demonstrates the student's understanding of the real world based on the knowledge in economics as well as business management. For example, models may illustrate the basics of economy[9] so that even children can understand the business model.

The advantages of the presented method are:

1) a consistent approach to teach business modeling,
2) a combination of modeling and simulation,
3) an incremental approach from a simple model to complex one, and
4) a game-like approach for the retention of motivation.

The disadvantages are:

1) a relatively long course will be required which may span three or four semesters, and
2) a lot of effort will be required to construct a complex model.

However, building up a library of graphics and scripts will improve the productivity in the future.

4. Modeling and simulation

4.1. VWML overview

The specialized modeling language, VWML, is based on XML, and is defined in a DTD named "vwml.dtd". A VWML file defines both the world structure and the script for an object called "block".

The world structure is defined by the <*world*> tag while the script is defined by the <*script*> tag. A block can be a world, agent, bank, company, house, or any other object. Similar to the nested projects in Squeak, a block can contain other blocks, for example, to define a world with a house, a house with a fishbowl, and a fishbowl with a goldfish as shown in Figure 1. A block must have a unique name for the identification by name, and must belong to one of the categories defined in the DTD for the enumeration by category. A VWML file to be read in by the simulator defines the top-level world block whose name is assumed to be "world". The following is a fragment of the definition for the world shown in Figure 2.

```
<world author="any" title="An example world">
  <ground width="6" height="6" offset="2"
    image="boxel.png"/>
  <atlas>
    <line>....B.</line>
    <line>.X....</line>
    <line>......</line>
    <line>.C....</line>
    <line>......</line>
    <line>..Y..H</line>
  </atlas>
  <block name="bank1" symbol="B"
    category="bank" image="bank1.png"
    define="bank1.xml"/>
  <block name="company1" symbol="C"
    category="company" image="company1.png"
    define="company1.xml"/>
  <block name="house1" symbol="H"
    category="house" image="house1.png"
    define="house1.xml"/>
  <block name="agent1" symbol="X"
    category="agent" image="agent1.png"
    define="agent1.xml"/>
  <block name="agent2" symbol="Y"
    category="agent" image="agent2.png"
  define="agent2.xml"/>
</world>
```

The scripting language consists of two layers; one is the basic layer which provides the general scripting functions, and the other is the world layer which provides the modeling primitives. As the project further progresses, the latter layer will be enhanced to support more primitives.

The basic layer provides the following constructs:

- string, integer
- arithmetic operators (+, -, *, /, %)
- relation operators (=, !=, >, >=, <, <=)
- string operator (?)
- logical operators (!, &, |)
- associative array
- *assign*, *dispose*
- *if*, *if-true*, *if-false*, *else*
- *select*, *when*, *otherwise*
- *do-while*, *repeat-until*
- *push*, *pull*
- *push-element*, *pop-element*
- *function*, *call*, *return.*

There are only two types of literal values; string and integer. A string literal is enclosed by a pair of single-quote characters. The Boolean value is represented as a string. The arithmetic operation will be performed on a string assuming that its value is an integer. The variables are type-less, and there is no need to define a variable prior to its usage. When a variable is referenced with no prior assignment of a value, a "null" value will be returned.

A block of code, whose equivalent in C and Java is {…}, is written as an XML element which contains nested XML elements as in <*if* cond="x=1"> <*assign* var="a" expr="b*c-d"/> </*if*>. Therefore, the *else* clause can be written as in <*if* cond="x=1"> … <*else*> … </*else*> </*if*> instead of <*if* cond="x=1"> … <*else*/> … </*if*> which is the style adopted by VoiceXML[10].

The world layer provides the following primitives:

- *for-each*
- *request*, *on-request*
- *query*, *on-query*
- say, *on-say*
- *ask*
- *move*, *step*
- *route*
- *visit*, *leave*
- *play*
- *refresh*
- *wait, halt.*

The <for-each> primitive selects the blocks by name or by category. The <request> primitive sends a message to the selected blocks. When a message is sent to a particular block, the interpreter will search the XML tree to find the <on-request> primitive with a matching regular expression. The <say> primitive sends a message to the selected blocks, and the message will be displayed above the block. The <ask> primitive communicates with the user. The <move> primitive moves the block in a specified direction, and the <step> primitive performs the walking animation. The <route> primitive calculates the optimum path from one location to other location. The <visit> primitive enters the inside of other block, and the <leave> primitive exits the current place to return to

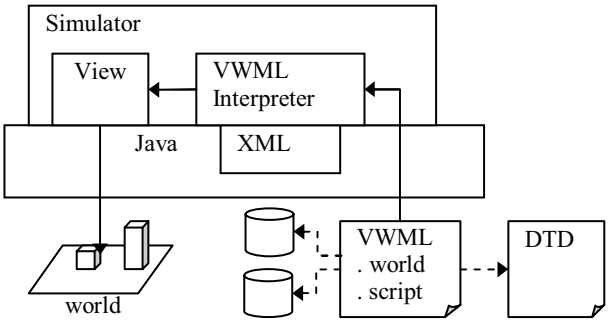

Figure 3. A structural overview.
The simulator is written in Java. The script written in VMWL will be interpreted to simulate the world.

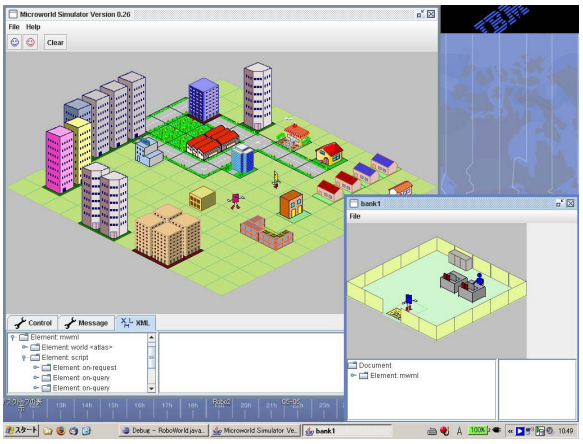

Figure 4. A modeling example.
The simulation is in progress, and the inside of a bank is shown in a separate window as one of the agents is visiting the bank.

the previous place. The <play> primitive plays a specified sound file. The <refresh> primitive updates the screen. The <wait> primitive adjusts the simulation timing, and the <halt> primitive stops the simulation.

For each block there are several predefined system variables whose values are automatically initialized and maintained by the simulator, such as @name, @x, @y, @mode, @step, @capacity, @#agents, and @parent. Also, there are some global system variables such as @@world, @@season, @@year, @@month, @@day, and @@time.

4.2. Functional distribution

The functional distribution between the primitives and the user-written scripts has been determined by two factors. The primary factor is to minimize the number of the primitives. In other words, the functions which cannot be realized by the script alone will be provided by primitives. Examples are the primitives such as <*move*> and <*play*>. The secondary factor is to improve the ease of scripting. In other words, the function which can be very complex or inefficient if implemented as a script will be provided by a new primitive. An example is the search of an optimum route from one location to other location in the world. This search algorithm may be implemented by writing a complicated script, but the script will not be very easy to write and very efficient to execute. Therefore, the <*route*> primitive is introduced to search the path between two locations for ease of programming and better performance.

4.3. Implementation

The structure of the simulator is shown in Figure 3. The user interface allows the selection of a top-level VWML file which will be parsed by the XML parser to construct an XML tree. The VWML interpreter runs as a thread, and interprets the XML tree to construct the top-level world. Examples of the blocks at this level are agents, companies, houses, and trees. During the construction of the top-level world, the interpreter may read in second-level VWML files which will be parsed by the XML parser to construct the second-level XML trees. During the construction of the second-level worlds, the interpreter may read in further XML files. This process will be repeated until all the details are completed.

When the simulation is started by the user, the top-level script will be invoked as if a message, <*request* message="run">, was sent to the top-level script. Therefore, the interpreter will search for the XML node whose value is <*on-request* message="run">. Then, the interpreter will interpret the child nodes in sequence. During the interpretation a symbol table will be maintained for each block. Also, a stack is implemented as an entry in the symbol table. A request or a query can be sent from a block to its child or from a block to its parent, and the data can be passed between two blocks via stacks.

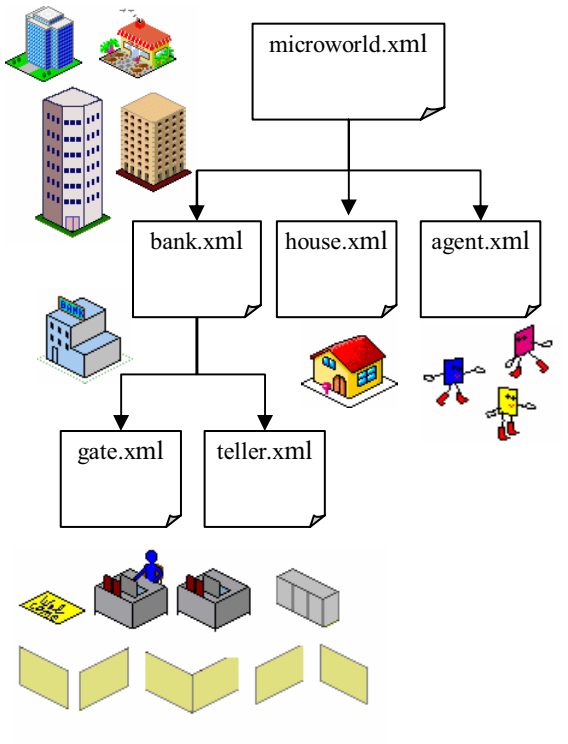

Figure 5. The relationship among the scripts.
The top-level script defines the world structure and invokes the scripts for a bank, a house, and three agents. In turn, the script for the bank defines its internal structure and invokes the scripts for further detail. It can be seen that the bank consists of several blocks including a teller, a cabinet, and walls.

The entire simulation is performed in a single thread. There is no particular scheduling mechanism inside the simulator, and the simulation is performed in sequence by the nested command execution loops. The interaction between two blocks is performed directly without the use of message queuing. The simulation timing varies depending on the complexity of the simulated world such as the number of the blocks, the depth of the nested blocks, the complexity of the logic, the iterations of the loops, etc. Therefore, the timing to update the screen can be controlled by the script by executing the <*refresh*> primitive.

5. Modeling examples

An example of the modeling is shown in Figure 4. In this world there are a bank, a house, and three agents as well other blocks including some factories and buildings. The top-level script defines the world structure and controls the entire simulation. The bank and the house have their own script to define the internal structure and behavior. On the other hand, all

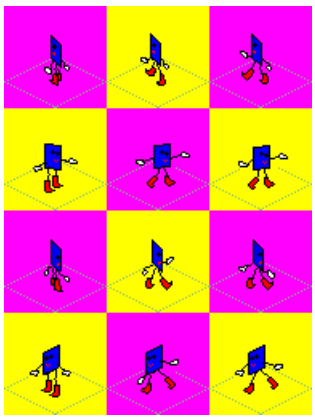

Figure 6. A graphic file.
A graphic file can contain 12 frames for a simple animation.

of the three agents are sharing the same script, meaning that the agents have a different appearance but have the same behavior. The hierarchy of the scripts is shown in Figure 5.

This example started with a simple structure and was later enhanced to the current structure with 7 scripts, 1 common script, and 35 graphics by sharing some graphics with another example to be discussed next. Each graphic file can consist of 12 frames, i.e., 3 poses in 4 directions, as shown in Figure 6 so that an animation can be produced by selecting the frames in a particular sequence.

A more complex world with two agent families shown in Figure 7 was created by two of the author's students in a step-by-step fashion. The initial stage was

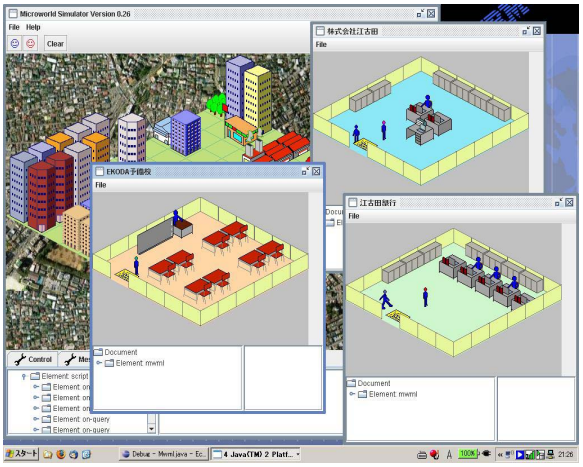

Figure 7. Another modeling example.
There are two families with three agents each. The company, bank, school, and shop are involved in the simulation. The shop is currently not shown because no agent is visiting there.

to have the agent withdraw some money at the bank. First, the students defined a minimum number of blocks; an agent block and a bank block. The top-level block (i.e., the world block) defined the world structure, but had no script. Then, the initial scripts were added to both the world block and the agent block so that the agent could move to the bank by executing the predefined sequence of the moves. By observing the behavior of the agent, the students understood that the agent script was depending on the world and was not flexible because the agent would fail if it was placed at a different position.

The students, therefore, tried to enhance the script so that the agent can "goto" the bank. However, they found that no <*goto-bank*> or equivalent primitive was defined in VWML. After having discussions with the author, they understood that "goto" was a high-level behavior which was composed of several low-level behaviors such as the initiation of the action, the selection of a particular bank, the identification of its location, the search of an optimum route, and the series of the steps along the route. By studying those situations they understood that a VWML function could be written to provide a generic "goto" function. Once the "goto" function became available, it was used to "goto" any places including the bank, school, house, shop, gate, and shelf. Lastly, the internal of the bank was defined so that the agent could "visit" the bank. At the bank the agent could "goto" the teller, make a deposit, "goto" the gate, and "leave" the bank.

The second stage was to add a company block so that the agent could work there for a salary. This was similar to the bank scenario. The third stage was to enhance the world by introducing a family; the father goes to work, the mother goes to bank and shop, and the child goes to school. During the simulation the students found that some error recovery mechanism was required because of the possible interferences during the walking of the agents. The forth stage was to add one more family. The last stage was to define the details of the world.

The scenario of this example is relatively complicated. The fathers go to work in the morning, enter the company building, do some work, submit a report, receive a salary, leave the building, and return to home in the evening. The mothers go to bank, enter the bank, go to the teller, withdraw some money, leave the bank, go to shop, enter the shop, go to the shelf, go to the cashier, pay for the purchase, leave the shop, and return to home. The children go to school in the morning, enter the school, go to own desk, do some study, leave the school, and return to home in the evening. Also, there are tellers, cashiers, and teachers.

It took about one semester to complete this example. The final version consists of 15 scripts, 1 common script, and about 50 graphics.

6. Discussion

Although the presented models were highly simplified version of our daily life, the students were able to acquire the basic knowledge and skills necessary for business modeling by observing the real world, extracting the key features, designing the world, drawing the graphics, writing the scripts, debugging the scripts, and enhancing the world. Currently, those students have been working on an application based on these examples, and other students have been working on the construction of models.

Through the modeling experiences, they have learned that there are many high-level activities in our daily lives such as "goto", "buy", and "sell" which need to be built on a series of smaller activities. The modeling and simulation will draw our attention to the fact that we unconsciously make strategic decisions at every moment[11][12].

The use of the simulation was successful in filling the gap between the entry-level programs and the commercial programs as the students could create and simulate their world with fun and without complex programming. They were able to understand the fundamental of the modeling, the concept of the object-oriented analysis, and the hypothesis and test cycle through the problem-based learning.

However, writing and debugging an even simple script needs a trial-and-error as well as algorithmic thinking. The syntactical errors often discourage the students, but unexpected behavior of the blocks resulted from logical errors could sometimes motivate them to find out the cause of the problems. Usually, a syntactical error can be corrected easily, but a logical error cannot. Therefore, it is important to provide a set of easy-to-understand samples in sequence as in the case of the creation of the example discussed above. In addition, providing a debug facility will be a greater help to investigate the logical errors because there is currently no debugger available.

VWML is a simple scripting language and is not intended for the sophisticated knowledge sharing such as provided by KQML[13]. The communication between two agents is achieved by the use of <*say*> primitive. This is a simple message passing from one block to another block provided that both blocks are adjacent when no distance parameter is specified. The message is parsed by the interpreter and the first <*on-say*> primitive whose value expressed in a regular expression matches the message will be invoked.

There are some candidate primitives for future improvement. An example is the pair of <*receive*> and <*release*> primitives which are similar to the pair of <*visit*> and <*leave*> primitives, but can be used, for example, to implement a shopping cart. Another example is the <*sort*> primitive which will be used to select the best candidate from a list. To assist the debugging process some debug primitives such as <*assert*> will be required. Also, it may be necessary to have an error handling mechanism similar to the try-catch model in Java in addition to the currently available <*if-true*> and <*if-false*> primitives.

All the transactions such as the withdrawal at the bank and the payment at the shop are currently implemented by performing simple arithmetic on local variables. However, in the future, all the financial data will be maintained and reported by implementing a tiny subset of XBRL (Extensible Business Reporting Language) [14] so that the status of all the blocks can be represented by a common model.

7. Conclusion

This work has addressed the education of business modeling using an XML-based modeling simulator. The presented method enables the students to learn the basic modeling knowledge and skills through the experiences starting from the introductory programming to the actual script-based simulation. This is particularly important for the students having no programming experiences. The world is described with VWML which is an XML-based modeling language, and is intended to model our economic activities; all the activities are to be described by the scripts, not by the parameters often used by the simulation games.

It can be seen that the modeling and simulation in a step-by-step fashion will help the students to start with a simple model and to learn from the visual feedback. Although the simulation needs both scripts and graphics, collecting sample scripts can assist the writing of the scripts, and building up a graphics library can promote the reuse of graphics. Establishing a set of modest rules in drawing graphics may also be helpful.

This simulator may be used to create other kinds of applications such as education materials, games, presentations, and demonstrations. Currently, the simulator is a standalone program, but will be enhanced to enable the use of networking so that an agent can move from one desktop to another. Also, adding more primitives to VWML will allow more complex, realistic modeling.

Acknowledgement

The author would like to express his thanks to Takashi Yamakawa for the creation of some graphics. The author also would like to express his thanks to Daisuke Oishi and Satoshi Fujimoto for their example model as well as their feedback on the modeling language and the simulator.

References

[1] Squeak, http://www.squeak.org/.

[2] H.-E. Eriksson, M. Penker, *Business Modeling with UML*, John Wiley & Sons Inc., 2000.

[3] A. Takemasa, S. Sagawa, *An Introduction to UML for Businessperson*, Mainichi Communications, 2004 (In Japanese).

[4] J.A. Zachman, "A Framework for Information Systems Architecture", *IBM Systems Journal*, vol. 26, no. 3, pp. 276-292, 1987.

[5] N. Gilbert, K.G. Troitzsch, *Simulation for the Social Scientist*, Open University Press, 2005.

[6] D.M. Bourg, G. Seemann, *AI for Game Developers*, O'Reilly Media, Inc., 2004.

[7] M.Overmars, "Teaching Computer Science through Game Design", *IEEE Computer*, vol. 3, no. 4, pp.81-83, 2004.

[8] Lua, http://www.lua.org/.

[9] L. Armstrong, *How to Turn Lemons into Money: A Child's Guide to Economics*, Harcourt Children Books, 1976.

[10] VoiceXML, http://www.voicexml.org/.

[11] D.A. Norman, *Emotional Design: Why We Love (or Hate) Everyday Things*, Basic Books, 2004.

[12] Lord of The Sims, http://www.playcenter.com/PC_Games/interviews/will_wright_the_sims.htm.

[13] KQML, http://www.cs.umbc.edu/kqml/.

[14] XBRL, http://www.xbrl.org/Home/.