PeerSim Cooker : A GUI IDE for PeerSim

Eric Jui-Lin Lu
Department of Management Information Systems, National Chung Hsing University
250, Kuo Kuang Rd., Taichung 402, Taiwan R.O.C.
jllu@nchu.edu.tw

Yau-Tsan Jang
Department of Management Information Systems, National Chung Hsing University
250, Kuo Kuang Rd., Taichung 402, Taiwan R.O.C.
g9629005@mail.nchu.edu.tw

Guan-Wei Hwang
Department of Management Information Systems, National Chung Hsing University
250, Kuo Kuang Rd., Taichung 402, Taiwan R.O.C.
earows @gmail.com

Abstract*—Nowadays, P2P technology has attracted a great attention in both academia and industries. In general, simulators are used to study the performance of P2P protocols. However, it is found that it is very difficult for beginners to get acquainted with any simulator. To overcome this problem, we developed a GUI-based integrated development environment, called PeerSim Cooker for PeerSim. PeerSim Cooker also provides a wizard mode that can guide developers to accomplish experiments in a step-by-step manner. Additionally, a Unified Message Passing Framework is proposed and embedded in PeerSim Cooker so that the tasks required for developing experiments can be further simplified.*

Keywords: Peer-to-Peer, Simulator, IDE, PeerSim.

1. Introduction

Since the wide spread usage of Napster[12] and KazAa[5], people started to realize the power of resource sharing in a Peer-to-Peer (P2P) manner. As a result, many P2P protocols had been proposed. In general, to study the performance of a P2P protocol, a P2P simulator such as Narses[13], 3LS[17], NeuroGrid[4], P2PSim[15], PeerSim[1], etc. is employed. Unfortunately, P2P simulators shared common limitations: poor scalability, little or no documentation, and steep learning curve. Naichen et al. [10, 11] surveyed various P2P simulators and reported that PeerSim is one of the best P2P simulators. For example, PeerSim can simulate both structured and unstructured networks, allows nodes to dynamically leave or join a network, can simulate up to millions of nodes, and provides many reusable components (e.g. a Node represents a node and a Linkable represents a routing table). Also, PeerSim was developed in Java which made it platform independent. Therefore, many P2P research projects [2, 6, 7, 8, 9, 16] employed PeerSim as their simulation platform. Although PeerSim is functional rich, it is still difficult for beginners to correctly select and utilize existing components to complete experiments. To reduce steep learning curve, it is believed that a graphical user interface (GUI) should be provided by simulators. In addition, a wizard mode, that can step-by-step guide developers to complete experiments, is a big plus for simulators [14, 18, 19]. In this paper, we developed a GUI-based integrated development environment (IDE) for PeerSim which is called PeerSim Cooker. PeerSim Cooker also supports wizard mode to relieve difficulty in learning PeerSim.

In PeerSim, two models of simulations are supported – cycle-based and event-based models [1]. To design an experiment using PeerSim, developers have to determine which model is used. Unfortunately, once a model is selected, the program codes developed for the model cannot be reused for the other model. In other words, developers have to design one version of codes for each model. To reduce the development burden, a Unified Message Passing Framework (UMPF) is proposed and embedded in PeerSim Cooker. For simulation programs conforming to UMPF, they can be run in either cycle-based or event-based models.

The rest of the paper is organized as follows: PeerSim is briefly described in Section 2. In Section 3, UMPF is discussed in details. In Section 4, we study the maximum numbers of nodes in various simulation environments when PeerSim Cooker is used. The experimental results are also presented. The implementation

This research was partially supported by the National Science Council, Taiwan, R.O.C., under contract no.: NSC 95-2221-E-005 -050 -MY2.

and screenshots of PeerSim Cooker is illustrated in Section 5. Finally, the conclusions and possible future works of PeerSim Cooker are presented.

2. PeerSim

PeerSim[1] is a Java-based P2P simulator for overlay networks. In a PeerSim simulated network, each node is represented by an object of type Node. Objects of type Control are used to control nodes joining and leaving the network. The behaviors (or protocols) of each node are implemented by objects of type Protocol.

In general, the tasks of a typical experiment using PeerSim include the following steps [3]: (1) determine the number of nodes in a network, (2) design Protocol objects and initialize them, (3) design Control objects to control and possibly monitor the network, and (4) execute the simulation.

PeerSim supports two models of simulations – cycle-based and event-based models. In cycle-based simulations, PeerSim will sequentially execute all node protocols in one cycle. Between any two cycles, developers are allowed to add Control objects. These Control objects can be used to add or remove nodes, or monitor the values of specified variables. For example, as shown in Fig. 1, each node has two protocols (P1 and P2). In the i^{th} cycle, PeerSim executes both P1 and P2 of all nodes, and C1 will be executed before the end of the cycle.

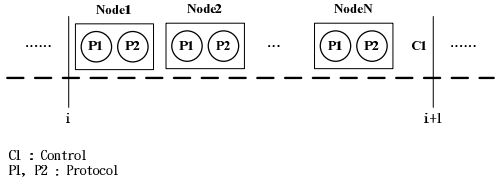

Figure 1. An example cycle-driven simulation

In event-based simulations, events are defined along a time axis. For each time tick, there may be zero or more events. For each event, a protocol is defined. For example, as shown in Fig. 2, there are two events (E_n and E_m) defined in time *i*. E_n defines the execution of P2 in node A, while E_m defines the execution of P1 in node B.

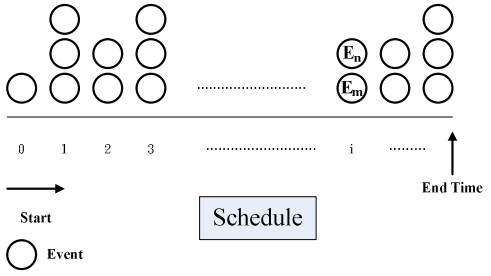

Figure 2. An example event-driven simulation

In PeerSim, developers have to determine which model of simulations should be used. If developers chose to experiment using cycle-driven engine, program codes have to implement the method nextCycle() as shown in Fig. 3. On the contrary, if developers chose to experiment using event-driven engine, program codes developed earlier cannot be reused. This is because, as shown in Fig. 3, program codes have to implement the method processEvent(). To overcome this problem, a Unified Message Passing Framework (UMPF) is proposed.

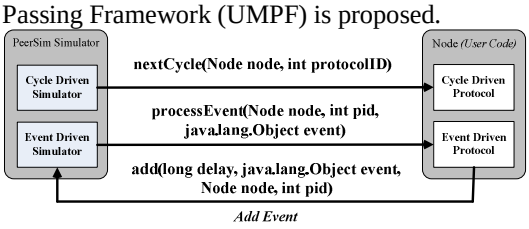

Figure 3. PeerSim calling processes

3. Unified Message Passing Framework – UMPF

As shown in Fig. 4, UMPF mainly include a set of peersimcooker.message.handler.MessageHandlers (abbreviated as MessageHandler) and two classes, Cycle Driven Message Manager (CDM) and Event Driven Message Manager (EDM) which are called by cycle-driven and event-driven simulators respectively. All messages communicated among CDM/EDM and MessageHandlers are objects of subclasses of peersimcooker.message.Message (abbreviated as Message). Each Message has a corresponding MessageHandler.

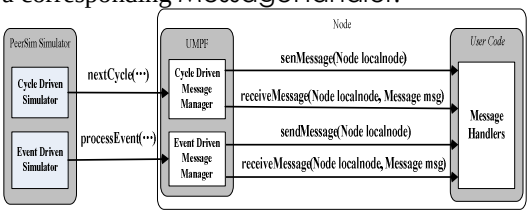

Figure 4. Unified Message Passing Framework

For clarity, the example simulation experiment in [3] is used for explanation. In the experiment, each node has an integer value which is randomly generated between 0 and 100. After randomly selecting a neighbor node from its routing table, each node will send its local value to the neighbor node. After receiving a value, one node will calculate the average of its local value and the received value; set its local value to the average; and send the average back to the source node. As shown in Fig. 5, node A has a local

value 70, and node B has a local value 30. Node A sends its local value to its neighbor node B. Once the value is received, node B computes the average which is 50, sets its local value to 50, and then sends 50 back to node A.

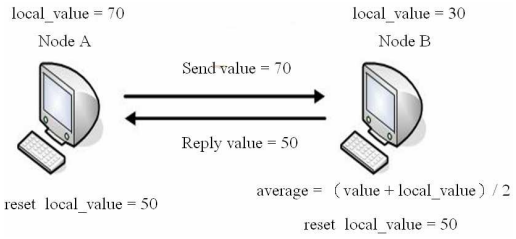

Figure 5. An example experiment

For all subclasses of Message, developers have to define all required information and a method called getMsgType(). getMsgType() returns a string which is used to select an appropriate object of type MessageHandler. For the example experiment, a SendMessage class, which is a subclass of Message, is defined. In the SendMessage as shown in Fig. 6, a value is declared to hold the value 70, and a node is declared to hold the source node which is node A. MessageHandler, which is a subclass of Protocol, is a class which contains two abstract methods – sendMessage() and receiveMessage() which are needed to be implemented by developers. The main purposes of sendMessage() are to create a message and notify a object to do something. When the receiveMessage() of a node is invoked, the node will execute whatever is needed to do when receiving a message. There is one important method send() in MessageHandler that worth to be noted. send() is a general purpose method, and there is no needs for developers to write codes for it. By assigning appropriate arguments, send() will send specified messages to a specified object.

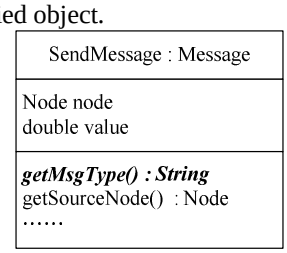

Figure 6. UML Diagram of SendMessage

By using the example, the simulation procedure is described as follows if cycle-based engine is used: When it is time for node A to send a message, cycle-driven simulator executes nextCycle() of a CDM object in node A. Then, sendMessage() of node A's SendMessageHandler will be executed. In the sendMessage(), a SendMessage object (SM_A), which contains node A and the value 70, will be created and SM_A will be sent to a target object by using send(). Because cycle-driven was selected, and because the target object is node B, SM_A will be sent to node B. When SM_A is received, node B calculates the average, resets its local value to 50, and invokes SM_A.getMsgType() which returns a string "SendMessage". Based on the configuration file as shown in Fig. 7, node B creates a SendMessageHandler object (SMH_B), and SMH_B.receiveMessage() will be executed. In the receiveMessage(), a ResponseMessage object (RM_B), which is similar to SM_A, will be created, and RM_B will be sent back to node A. When RM_B is received, node A resets its local value to 50 and invokes RM_B.getMsgType() which returns a string "ResponseMessage". Based on the configuration file as shown in Fig. 7, node A executes receiveMessage() of the ResponseMessageHandler object in node A.

```
protocol.handler1 SendMseeageHandler
{ ……
  type SendMessage
}
protocol.handler2 ResponseMseeageHandler
{ ……
  type ResponseMessage
}
```

Figure 7. PeerSim Configuration

If event-based engine is used for the experiment, the simulation procedure is described as follows: When it is time for node A to send a message, event-driven simulator will execute processEvent() of a EDM object in node A. The EDM object may receive two types of objects from the simulator. One is Event objects, and the other is Message objects. In the example experiment, an Event object will first be received. The EDM will execute sendMessage() of a SendMessageHandler object in node A. In the sendMessage(), a SendMessage object (SM_A), which contains node A and the value 70, will be created and SM_A will be sent to a target object by using send(). Because event-based model was selected, the send() method will send SM_A as an event to the event-driven simulator. When it is time SM_A should be executed, node B calculates the average, resets its local value to 50, and invokes SM_A.getMsgType() which returns a string "SendMessage". Based on the configuration file as shown in Fig. 7, node B creates a

SendMessageHandler object (SMH_B), and SMH_B.receiveMessage() will be executed. In the receiveMessage(), a ResponseMessage object (RM_B), which is similar to SM_A, will be created, and RM_B will be sent as an event to the event-driven simulator. When RM_B should be executed, node A resets its local value to 50 and invokes RM_B.getMsgType() which returns a string "ResponseMessage". Based on the configuration file as shown in Fig. 7, node A creates a ResponseMessageHandler object (RMH_A), and RMH_A.receiveMessage() will be executed.

From the above discussions, it is clear that the program codes for sendMessage() and receiveMessage() are identical no matter which models of simulation is selected. In other words, developers only have to write one version of codes, but can select either model at will.

4. Experiments

It is known that GUI components consume more system resources and thus lower the number of nodes that PeerSim could simulate. In this section, we study the maximum numbers of nodes in various simulation environments when PeerSim Cooker is used. The example shown in Fig. 5 was used for the following experiments. Both cycle-based and event-based models were studied. For each model, the example simulation was run in three different cases. In case 1, simulations were run in pure PeerSim; in case 2, simulations were run in PeerSim Cooker which read the configuration file used in case 1; in case 3, simulations were run in PeerSim Cooker's wizard mode. All experiments were executed on a PC with an Intel Core 2 Duo CPU (1.86GHz), 2GB RAM, and JDK 1.6.0_07. Fig. 8 shows the experimental results.

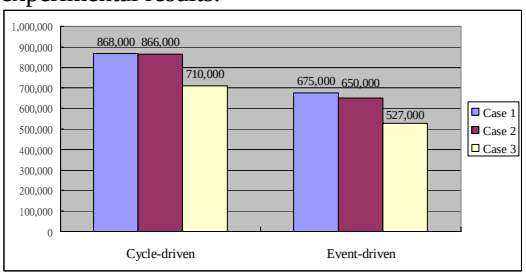

Figure 8. Comparison of the number of nodes in different simulation environments

As shown in Fig. 8, in cycle-based simulations, case 1, 2, and 3 can simulate up to 868,000, 866,000, and 710,000 nodes; respectively. In event-based simulations, case 1, 2 and 3 can simulate up to 675,000, 650,000 and 527,000; respectively.

Gnutella was also used to study PeerSim Cooker. All Gnutella simulations were run in three different cases. As shown in Fig. 9, in cycle-based simulations, case 1, 2, and 3 can simulate up to 472,000, 458,000, and 246,000 nodes; respectively. On the other hand, in event-based simulations, case 1, 2, and 3 can simulate up to 280,000, 279,000, and 250,000 nodes; respectively.

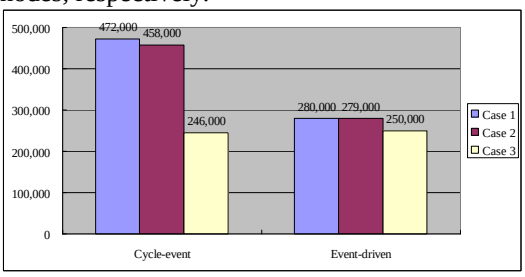

Figure 9. Comparison of the number of nodes using Gnutella

5. PeerSim Cooker

PeerSim Cooker was developed in Java. Both expert and wizard modes are supported by PeerSim Cooker. In the expert mode, developers can design components at their own will by clicking on GUI components. In the wizard mode, developers are guided to complete required component step by step. In the interest of space, only wizard mode is described below.

To simulate the example experiment, a developer can set up basic configurations such as the number of nodes and the simulation engine by using the GUI components shown at the left of Fig. 10. For the example, the number of nodes was 1,000, cycle-based engine was selected, and the number of cycles was set to 50. Then, "+" button was clicked and a wizard window was shown. After clicking on "Quick start for Beginner", one can create required components guided by the wizard.

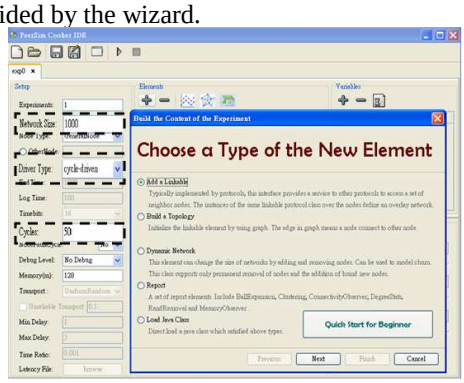

Figure 10. Main screen

As shown in Fig. 11, the first step was to build up a topology for the simulated network. PeerSim provides several topologies. Thus, a developer can simply pick one from the drop-down list. If a

required topology does not exist, one can create one by clicking "Add a Linkable". In the example experiment, "Random Connection" was selected.

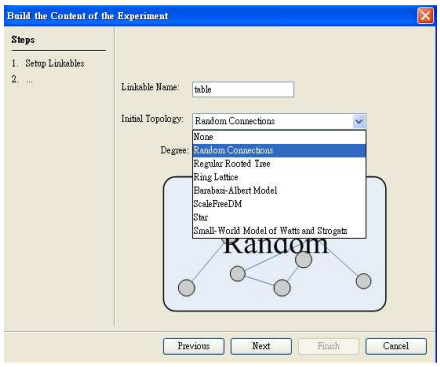

Figure 11. Setup Linkables

The second step is to set up resources for each node. In the example experiment, a variable called *local_value* was created. Then, after filling in the range of *local_value* and selecting "Uniform Distribution", the developer clicked on the "Setting" button in Fig. 12. This action results in randomly generated integers between 0 and 100 will be uniformly distributed to all nodes. Three types of distributions are provided by PeerSim.

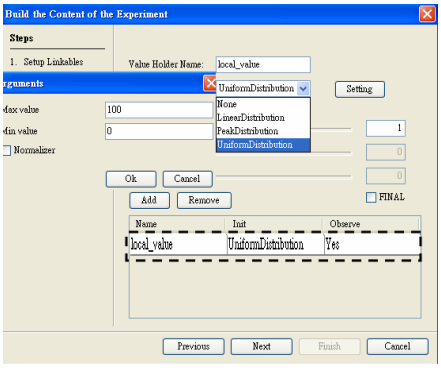

Figure 12. Setup Value Holders

The content of Message is defined in step 3. As shown in Fig. 13, SendMessage and ResponseMessage were defined. For each message class, content such as source node and value can be added.

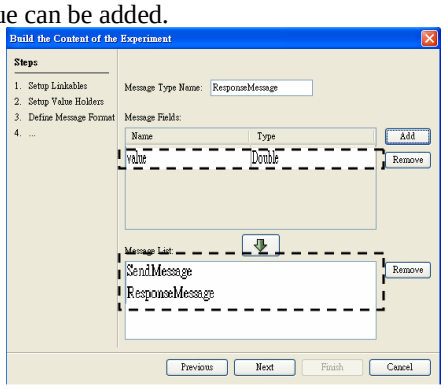

Figure 13. Define Message Format

In step 4, MessageHandlers are defined. For each Message defined in the previous step, a corresponding MessageHandler has to be defined. Thus, for the example experiment, the developer defined two handlers – SendMessageHandler and ResponseMessageHandler. For each MessageHandler, the developer also have to implement both sendMessage() and receiveMessage(). The program codes for the sendMessage() and receiveMessage() of SendMessageHandler are shown in Fig. 14 and Fig. 15. After finishing these program codes, the developer can click on  in Fig. 10.

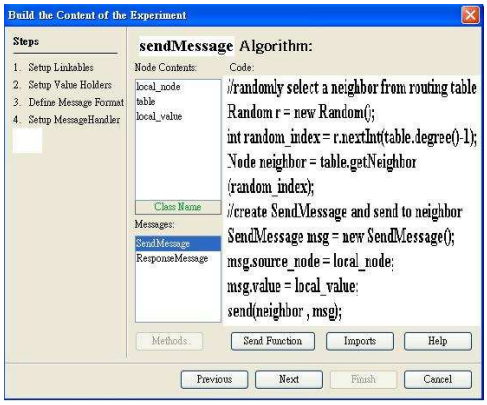

Figure 14. SendMessageHandler — sendMessage

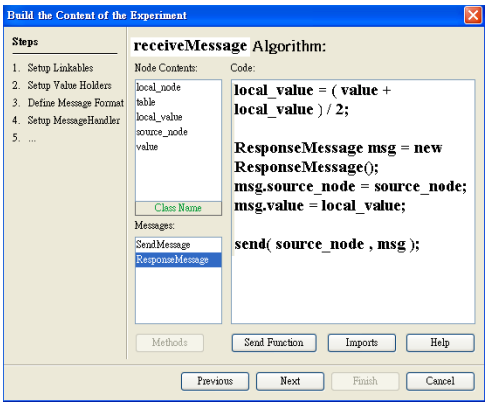

Figure 15. SendMessageHandler — receiveMessage

Additionally, if the developer wishes to see graphical experimental results, she can simply click on  and selects to-be-monitored variables from a drop-down list. For the example experiment, *local_value* was selected and the result was shown in Fig. 16.

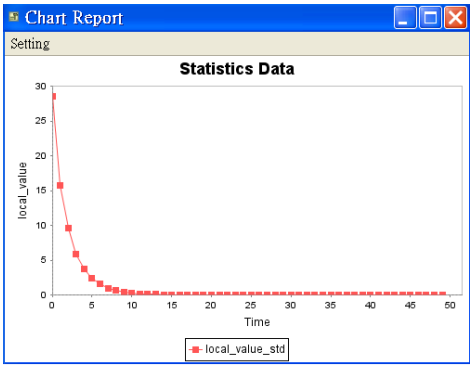

Figure 16. Experiment Result

6. Conclusion

In this paper, we developed a GUI integrated development environment, called PeerSim Cooker, for PeerSim. By using PeerSim Cooker, developers can easily create required components to complete simulated experiments. PeerSim Cooker also provides a function that generates graphical experiment results. The wizard mode supported in PeerSim Cooker can reduce the learning curve of PeerSim. Furthermore, by utilizing UMPF, there is no need to design codes for each model of simulations. Currently, PeerSim Cooker is available for download at http://xml.nchu.edu.tw. In the future, we wish to provide a user-friendly editor for PeerSim Cooker and to develop a general model to adapt the changes of the underlying PeerSim. Also, we wish to further enhance UMPF so that less memory is required.

References

[1] M. Jelasity, A. Montresor and G. P. Jesi, "PeerSim P2P Simulator," [Online] Available: http://peersim.sourceforge.net/.

[2] M. Jelasity, A. Montresor and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," ACM Transactions on Computer Systems, vol. 23, no. 3, pp. 219–252, August 2005.

[3] G. P. Jesi, "PeerSim HOWTO: Build a new protocol for the PeerSim 1.0 simulator," December 2005. [Online] Available: http://peersim.sourceforge.net/tutorial2/tutorial2.html.

[4] S. Joseph, "NeuroGrid: Semantically Routing Queries in Peer-to-Peer Networks," Lecture Notes in Computer Science, vol. 2376, pp. 202 - 214, 2002.

[5] "KazAa" [Online] Available: http://www.kazaa.com/us/index.htm.

[6] E. Lu, Y. Huang, and S. Lu, "ML-Chord: A Multi-Layered P2P Resource Sharing Model", to appear in the Journal of Network and Computer Applications.

[7] A. Marcozzi and D. Hales, "Emergent Social Rationality in a Peer-to-Peer System," Technical Report UBLCS-2006-23, University of Bologna, Department of Computer Science, October 2006.

[8] A. Montresor, "A Robust Protocol for Building Superpeer Overlay Topologies," Proceedings of the Fourth International Conference on Peer-to-Peer Computing, pp. 202 – 209, August 2004.

[9] A. Montresor, M. Jelasity, O. Babaoglu, "Chord on Demand," Fifth IEEE International Conference on Peer-to-Peer Computing, pp. 87- 94, August 2005.

[10] S. Naicken, A. Basu, B. Livingston and S. Rodhetbhai, "A Survey of Peer-to-Peer Network Simulators," Proceedings of the Seventh Annual Postgraduate Symposium, June 2006.

[11] S. Naicken, B. Livingston, A. Basu, S. Rodhetbhai, I. Wakeman and D. Chalmers, "The State of Peer to Peer Simulators and Simulations," ACM SIGCOMM Computer Communication Review, vol. 37, no. 2, 2007.

[12] "Napster" [Online] Available: http://free.napster.com/.

[13] "Narses" [Online] Available: http://arxiv.org/abs/cs/0211024.

[14] T. J. Overbye, P. W. Sauer, C. M. Marzinzik and G. Gross, "A user-friendly simulation program for teaching power systemoperations," IEEE Transactions on Power Systems, vol. 10, no. 4, November 1995.

[15] "P2PSim" [Online] Available: http://pdos.csail.mit.edu/p2psim/.

[16] G. Rossi, S. Arteconi and D. Hales, "Evolving Networks for Social Optima in the "Weakest Link Game"," Technical Report UBLCS-2006-21, University of Bologna, Department of Computer Science, July 2006.

[17] N. S. Ting and R. Deters, "3LS - A Peer-to-Peer Network Simulator," Proceedings of the Third International Conference on Peer-to-Peer Computing, pp.212–213, 2003.

[18] S. Wei, "A Discussion on User- Friendliness and Application Design," Computer Applications and Software, no.9, 2002.

[19] J. Wu, S. T. Chanson and Q. Gao, "Formal Methods for Protocol Engineering and Distributed Systems," Proceedings of International Federation for Information Processing, October 1999.