Design of ZeroMQ-Based Cooperative Simulation Framework for Distributed Code and Model Components

Sunghee Lee, Heungjun Park, and Woo Jin Lee

***Abstract*—Recently, the scale of distributed computing environment is growing larger. One of the reasons is development of IoT environment which directly impacts the human life by providing instant access to vast amount of information and services correspond to healthcare, smart home service, etc. Interaction testing among distributed systems such as IoT systems is non-trivial task, also cooperative simulation of both model and code components is more difficult one. Model-based and code-based simulation techniques are widely used to test embedded systems. Usually code-based simulation is possible when all the modules are implemented, after finishing model-based simulation. Both simulations cannot be applied cooperatively due to lacks of cooperation among model and code components. In this paper, we propose a ZeroMQ-based simulation framework for simulating distributed components to mitigate these problems of testing distributed components. This framework can be applicable to even incompletely implemented system, which makes it possible to develop the system incrementally.**

***Index Terms*—Cooperative simulation, distributed simulation, model and code, IoT.**

I. Introduction

Today, many people are interested in IoT (Internet of things) which make anyone connect to anything at any time and place or which make anything connect to anything at any time and place [1]. These availability and connectivity features have led to development of IoT systems for enhancing the convenience of our life. For example, wearable devices such as ‘Galaxy Gear’ and ‘G Watch’ can consume contents at any time and place. Also ‘LG Home Chat’ makes it possible for user to remotely control and monitor home appliance through mobile messenger whenever and wherever he wants.

These IoT systems constantly connect, interact, and cooperate with their surrounding environment and other devices such as PC, smart phone, tablet, and so on [2]. We need to test whether IoT systems work normally or not, for example, when CCTV is connected with user’s smart phone to monitor user’s house. But it is very difficult to test all of these operations of real IoT systems in real environment. Instead, we can simulate model or source code of IoT systems to test many of these operations. Existing simulation techniques can either perform model-based or code-based simulation one at a time. In this approach, already developed model cannot be utilized for code-based simulation and vice-versa. Also most of the existing simulation techniques work in central isolated environment.

In this paper, we propose a ZeroMQ-based [3] framework for simulating distributed components to mitigate the aforementioned problems. The framework enables a component of a system to communicate with other components of the system to simulate through ZeroMQ-based message broker. The proposed structure is able to perform cooperative simulation among model and code components, which makes it possible to incrementally implement the system.

This paper is structured as follows. Section II provides related work and Section III provides the structure of framework to cooperatively simulate model and code. For providing feasibility of our approach, in Section IV, case study is presented and Section V provides conclusion.

II. Related Work

A. Host-Based Testing

Unlike general software which is developed in target machine environment, embedded software is developed in host environment and is usually tested in the same environment [4]. Simulation is usually performed in the host environment that is not a target machine but PC or VP (Virtual Prototype) [5]. IoT system [1] possess the same problems faced by embedded distributed environment. Some parts of the system must be tested in target environment whereas other parts can be tested in the host environment because the host environment is clearly different from the target machine environment. In this paper, the proposed framework performs simulation in host environment in which independent simulation of each components represents unit testing whereas simulation of multiple components represents integration testing.

B. Model-Based Simulation

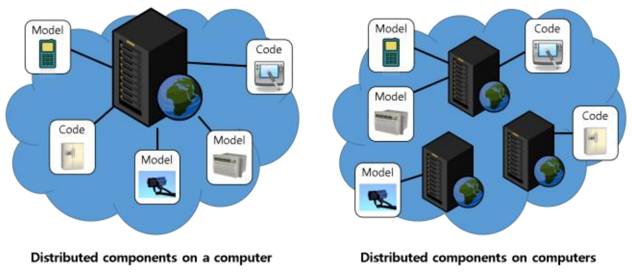
It is not difficult to simulate a model which does not interact with other components. But in IoT environment, most systems have not only several abstract levels but also several distributed applications which interact among themselves [6].

These distributed models are defined as sub-models and these sub-models may be parts of one model which impose the object-oriented concept. Model components communicate with each other only through specified message. In this

Manuscript received November 10, 2014, revised June 25, 2015. This work was supported by the IT R&D program of MSIP/KEIT [10041145, Self-Organized Software platform (SoSp) for Welfare Devices] and Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(2014R1A1A2058733).

The authors are with the Department of Computer Science and Engineering, Kyungpook National University, Republic of Korea (e-mail: lee3229910@gmail.ac.kr, bbakeung@naver.com, woojin@knu.ac.kr).

perspective, shown in Fig. 1, it is possible to simulate a model irrespective of how each sub-model is distributed on computers and how many sub-models exist on a computer [7].


Model
Code
Code
Model
Model
Model
Code
Model
Code
Model
Distributed components on a computer
Distributed components on computers
Fig. 1. Component of IoT system distribution.

Modeling is usually represented by state chart and each state has information of system's behavior. States (or models) communicate via message with each other. In this paper, the unit of modeling is a component which is composed of one or more model components. And a combination of one or more components is able to simulate single or multiple applications [7].

C. *Code-Based Simulation*

After model-based simulation, verified model is implemented by source code. In IoT system, all models can be implemented concurrently. The primary goal of code-based simulation is verifying the designed models with respect to source code, testing whether the models perform normally after model-based simulation and whether the models can be implemented correctly. Also, well-written code may be reused when real system is implemented [8].

To simulate code, virtual component is implemented by using dummy and stub. Dummy is a simple interface which simulates the real functions and stub can substitute a part of functions which system actually performs. Scale and precision of code-based simulation are decided by how these virtual components are implemented. The more detail they are implemented, the more precise code-based simulation is; otherwise code-based simulation may be poor.

Code will not be able to simulate either if all components that compose an application are not implemented or if any components that compose the application has been partially implemented. Precisely, this problem we are solving using our proposed framework.

D. *ZeroMQ*

ZeroMQ [3], LGPL open source, is socket library which supports concurrency framework. It can be used to connect codes in any language such as C, C++, JAVA, .NET, Python and on any platform such as Linux, Windows. It is known as Ø MQ, 0MQ, zmq and provides sockets that transport atomic messages across various transport layers such as in-process, inter-process, TCP, and multicast. It is able to connect sockets with N-to-M and uses asynchronous I/O which provides scalable multicore applications.

ZeroMQ is appropriate in IoT because it can be used in centralized, distributed, small, or large system. In this paper as shown in Fig. 2, we use REQ and REP sockets of ZeroMQ. REQ-REP communication is similar to TCP communication. For example, as in TCP communication, here also only client can initiate communication by using REQ socket. Similar to TCP, a server responds using REP socket but it can't initiate communication. Distributed components use this REQ-REP combination to communicate with each other through message broker.

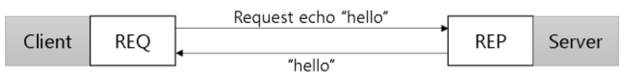
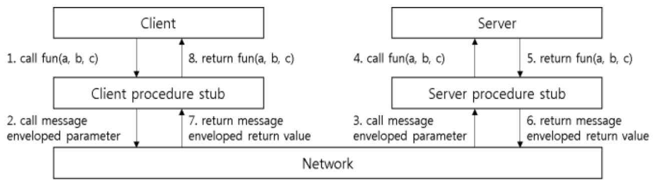


Fig. 2. REQ-REP combination.

E. *Remote Procedure Call (RPC)*

RPC is based on the semantics of a local procedure call, but actually it acts across the network [9]. It is used to call a procedure which doesn't exist in local system; instead it exists in remote system which can be accessed through the network. Fig. 3 represents basic RPC mechanism. For example, as shown in Fig. 3, client calls a function which has parameters a, b, and c. Since the function does not exist in local system, client needs client procedure stub. Then, to call the real function which exists in server, the stub generates a message by encapsulating parameters to be sent to the function located at server and then the message is delivered through network. Server also needs server procedure stub to de-capsulate the received message and call the function with received parameters. The stub calls function in server which in turn returns value of the function from itself to the stub. The stub then sends back the returned value to client by following the same encapsulation and de-capsulation procedure. In this way, client can call a function in server.


Client
1. call fun(a, b, c)
8. return fun(a, b, c)
Client procedure stub
2. call message enveloped parameter
7. return message enveloped return value
Server
4. call fun(a, b, c)
5. return fun(a, b, c)
Server procedure stub
3. call message enveloped parameter
6. return message enveloped return value
Network
Fig. 3. RPC mechanism.

In this paper, we apply RPC mechanism to framework for model and code cooperative simulation. Distributed components can be both client-side and server-side at the same time, so that they also include stubs.

III. Design of ZeroMQ-Based Framework for Model and Code Cooperative Simulation

In this paper, to perform model and code cooperative simulation, both model and code should be made up unit of component which contains ZMQ, stub and simulator that is based on either model or code. In this section, we propose the structure of ZeroMQ-based framework and component for model code cooperative simulation.

A. *ZeroMQ-Based Framework for Distributed Simulation*

We apply aforementioned RPC mechanism to framework in order to simulate distributed components which are either model-based or code-based as shown in Fig. 4. The message broker acts as connector (network in RPC) among components (client or server in RPC) so that components can interact with others through message broker. A component is composed of three different entities: ZMQ, stub and simulator. The simulator may be either model-based or code-based.

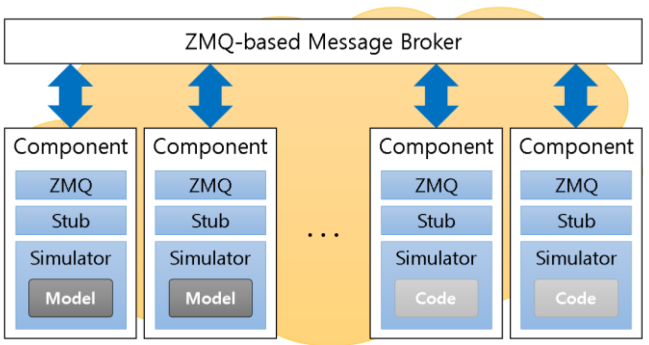


Fig. 4. ZeroMQ-based framework for distributed components.

B. *Cooperative Simulation of Model and Code Components*

For cooperative simulation of model and code components, we assume that simulator entity can communicate with component message translate stub by using only component API call and All interface among distributed component of a system is defined. Fig. 5 shows structure of component for cooperative simulation.

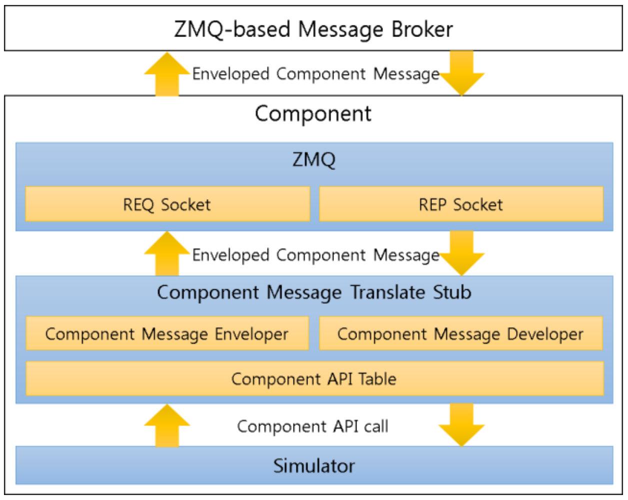

ZMQ-based Message Broker
Enveloped Component Message
Component
ZMQ
REQ Socket
REP Socket
Enveloped Component Message
Component Message Translate Stub
Component Message Enveloper
Component Message Developer
Component API Table
Component API call
Simulator

Fig. 5. Structure of component for cooperative simulation.

ZMQ entity has REQ and REP socket. The REQ socket makes component to act as client-side network in RPC whereas the REP socket makes component to act as server-side network in RPC. Component message translate stub has component message enveloper which act as client procedure stub in RPC and component message developer which act as server procedure stub in RPC. Enveloper makes component message enveloped API information whereas developer gets API information from enveloped component message. Component API Table contains API information that system defined previously. Simulator calls a component API, which is called by other component.

For example, component A based on *model* has two API which are defined as "*int add(int a, int b)*" and "*int subtract(a, b)*" in each component API table. The add API exists in its local but the subtract API doesn't. Component B based on *code* also has two API which are the same as component A. The subtract API exists in component B but the add API doesn't. Therefore, both components A, B should be perform cooperative simulation by interacting each other.

If simulator of component A calls the *subtract* API, Component A envelopes component message with parameters on the basis of component API table and then it delivers the message to component B through REQ socket. In component B which receives the message, stub gets information API from delivered message through REP socket and then the stub call real *subtract* API in component B. Then the returned value from component B is sent back to component A by following the same aforementioned procedure. In the same way, component B calls *add* API and gets return value.

IV. Case Study

In this section, shown in Fig. 6, our framework can be apply to simple smart home environment which has three components: model-based *Smart Home Model*, code-based *Emergency Control System* and model-based *Lamp Model*. This environment already has two defined API: EmergencyModeAPI() to broadcast a "emergency" event, LampOffAPI() to generate a "lampOff" event.

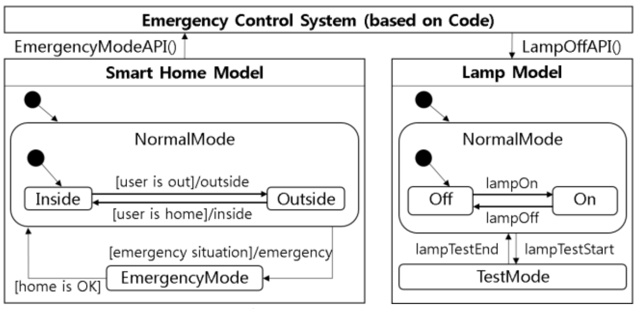


Fig. 6. Example of simple smart home environment.

For example, first we have implemented the entire system and finish Emergency *Control System* after completing model-based simulation. Then, we need to test the system which interact with the others based on model. When home is under emergency situation, "emergency" event in *Smart Home Model* is broadcasted by EmergencyModeAPI() which is API called from model and exists in *Emergency Control System*. This is process of API call from model to code. Next, because home is under emergency situation, *Emergency Control System* must turn off some devices. As one of them the system must turn off a lamp in *LampModel*, so that the system calls LampOffAPI() to generates a "lampOff" event. *Lamp Model* is "Off" state if it is "On" state. In this way, our framework makes it possible to simulate among different components.

V. Conclusion

In this paper, we proposed the structure of ZeroMQ-based framework for distributed simulation. In the framework, components interact with each other through message broker by using REQ-REP sockets combination. Our framework, which there are three different entities such as ZMQ, message translation stub, and code or model simulator, supports cooperative simulation among model and code components. For presenting feasibility of our approach as a case study, we applied it to a simple example of smart home environment. As using our framework, each distributed component of system can be developed incrementally on a computer and on computers. Also the framework is applicable in any language and on any platform because it is ZeroMQ-based. In future we study concurrent simulation of distributed model and code for the same component.

REFERENCES

[1] R. Khan, S. U. Khan, R. Zaheer, and S. Khan, "Future Internet: The Internet of things architecture, possible applications and key challenges," in *Proc. 10th International Conference on Frontiers of Information Technology (FIT)*, 2012, pp. 257-260.

[2] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-based Internet of things: Discovery, query, selection, and on-demand provisioning of web services," *IEEE Transactions on Services Computing*, vol. 3, issue 3, pp. 223-235, 2010.

[3] Information. [Online]. Available: http://zeromq.org

[4] H.-M. Qian and C. Zheng, "A embedded software testing process model," in *Proc. International Conference on Computational Intelligence and Software Engineering*, 2009, pp. 1-5.

[5] S. Y. Jang, H. Ryu, and W. J. Lee, "Development environment of SWF based virtual prototype for embedded software simulation," *Journal of KIISE: Computing Practices and Letters*, vol. 19, no. 12, pp. 643-647, 2013.

[6] T.-G. Kim, "Modeling simulation engineering," *Communications of the Korea Information Science Society*, vol. 25, no. 11, pp. 5-15, 2007.

[7] F. Junqueira, E. Villani, and P. E. Miyagi, "A platform for distributed modeling and simulation of productive systems based on Petri nets and object-oriented paradigm," in *Proc. 10th IEEE Conference on Emerging Technologies and Factory Automation*, vol. 1, 2005, pp. 907-914.

[8] S. Demers, P. Gopalakrishnan, and L. Kant, "A generic solution to software-in-the-loop," *IEEE Military Communications Conference*, pp. 1-6, 2007.

[9] C.-S. Lee, K.-H. Lee, J.-K. Lee, "A group RPC protocol for distributed systems," in *Proc. International Conference on Information, Communications and Signal Processing*, vol. 2, pp. 805-809, 1997.

Sunghee Lee is a student in the School of Computer Science and Engineering at Kyungpook National University, Korea. He received a bachelor's degree and a master's degree from Kyungpook National University. He is currently a candidate for the Ph.D. degree in the Department of Computer Science and Engineering. He has been a researcher at Embedded Software Engineering Laboratory, Kyungpook National University. His reserch interests include embedded sofeware testing and distributed embedded software testing.

Heungjun Park is a student in the School of Computer Science and Engineering at Kyungpook National University, Korea. He received a bachelor's degree from Kumoh National Institute of Technology. He is currently studying the master's degree in the Department of Computer Science and Engineering. He has been a researcher at Embedded Software Engineering Laboratory, Kyungpook National University. His reserch interest is embedded sofeware testing.

Woo Jin Lee is a professor in the School of Computer Science and Engineering at Kyungpook National University, Korea. He received the Ph.D. degree in computer science from Korea Advanced Institute of Science and Technology in 1999. His main research interests include software testing, embedded systems, software simulation, and formal methods.