An Approach to Testing Software on Networked Transport Robots

Ichiro Satoh

National Institute of Informatics
2-1-2 Hitotsubashi Chiyoda-ku Tokyo 101-8430 Japan
Email: ichiro@nii.ac.jp

Abstract—**Networked transport robots have been widely used to carry products in manufacturing and warehousing spaces. Such robots communicate with servers in the spaces and other robots through wireless local-area networks. Therefore, software running on such robots is executed with the services that the robots are connected to through networks, including multicast protocols. To test such software, we need to execute it within the network domains of the locations that the robots may move and connect to because the correctness of the software depends on the services. To solve this problem, we present a framework for emulating the physical mobility of transport robots by using the logical mobility of software designed to run on computers. It enables such software to run within target network domains so that the software can locally access servers and receive multicast packets limited to the domains.**

Keywords–Software testing; Wireless communication; Protocol; Mobile agent.

I. Introduction

Many manufacturers and warehousers have been using automated vehicles, called *transport robots*, to undertake repetitive transport tasks inside their facilities. Modern transport robots for warehousing and manufacturing spaces have become smart and exchange information on dynamic demands and environmental changes in their target spaces with stationary servers and other robots. They then should adapt themselves according to the received information. Thus, robotics software plays a key role as it is the medium through which their autonomy and adaptation are embodied. One problem is that the complexity of their software is far greater than conventional transport robots. For example, these robots are networked with stationary servers to exchange information with other robots via wireless networking, e.g., Wi-Fi. Furthermore, networking for transport robots in large warehousing and manufacturing spaces results in another serious problem in testing software for transport robots in the sense that these robots frequently connect or disconnect to multiple network domains, which may be smaller than target warehousing and manufacturing spaces, while they move in such spaces.

In addition, not only the hardware of such transport robots but also their software tend to be complicated. In fact, software plays a key role in robotics as it is the medium by which machines are made smart and adaptive. Software testing is a popular methodology for finding information on the quality of a software product or service by executing software intent on finding its own problems, e.g., bugs, errors, or other defects. Test-driven development is an evolutionary approach to development that combines test-first development in which you write a test before you write just enough production code to fulfill that test and refactoring.

The development and testing software for such robots is more difficult than that for conventional systems. This is because, typically, software for robots need to make robots reactive, concurrent, embedded, real-time, and data intensive. Most transport robots tend to communicate with stationary servers. Therefore, they are networked in order to exchange a variety of information with stationary servers and other robots via wireless networking. As a result, when a transport robot moves between locations, it may lose connectivity to a network domain provided on the previous location and then gain connectivity at another network domain provided on the current location. The software for running the robot can no longer connect to the servers provided in only the former domain, only those in the latter domain. To verify the correctness of software for networked transport robots, developers need to test software with all servers in the areas that their robots may visit through the robots' itineraries. However, it is difficult for developers to actually move real robots between locations in facilities that are used for business.

The purpose of this paper is to present a framework for testing software designed to run on transport robots. The framework is based on an early approach presented in one of our past papers [11], in which the approach supported testing software to be running on mobile computers by using the movement of emulators used for mobile computers. Since a manufacturing company asked us to develop a method to test software designed to run on transport robots, we extended the past approach with the ability to test moving robots to solve the company's problems. One reason is that mobile computers, which the past approach focused on, do not move between locations under their own control, but transport robots themselves move between locations. The past approach also assumed that the coverage areas of wireless networks to which mobile computers connected did not overlap, but in small spaces for warehousing and manufacturing, network domains supported through wireless networks may not be separated. The approach was aimed at testing client-side software running on mobile computers but server-side software often runs on transport robots. Therefore, although the framework presented in this paper is constructed on the basis of the basic concept of the past approach, it is extended with several abilities to test software running on transport robots.

We do not intend the framework to be general. The framework is aimed at testing networked software, which should be application-level in the sense that it does not directly access low-level hardware. Conversely, any lower-level software, e.g., OS and device drivers, including software for directly monitoring and controlling sensors and actuators is not within the scope of the framework. The framework proposed in this paper is an extension of our two early frameworks [11][13]. The first enables software designed to run on portable computers to directly connect to network domains in the sense that the software could send and receive packets reachable within the domains, but it does not support the movement of robots. The second was designed for testing software running on robots but lacks any mechanisms for emulating networking, e.g., changing Internet Protocol (IP) addresses, when robots move between two coverage areas of IP-enabled wireless networks.

The remainder of this paper is organized as follows. In Section 2 we discuss an example scenario. Section 3 presents the design and implementation of the proposed framework. Section 4 shows demonstrates the usage of the framework through an example and discuss software testing with the framework. Section 5 surveys related work and Section 6 provides a summary.

II. Example Scenario

As mentioned in the previous section, our framework was inspired by practical problems discussed in our research collaboration with a manufacturing company. The company's factory is shared by the company itself and its subsidiary companies. They use modern transport robots to carry products between the areas managed and operated by them, where each of the areas provides its own wireless local-area network for communicating with transport robots running within it and local services provided only in the network. Transport robots move from area to area in the factory along their itineraries as shown in Fig. 1, where the coverage area of each wireless network access point is smaller than the target manufacturing spaces. Each network area has one or more local servers available. A service discovery mechanism in each area periodically multicasts User Datagram Protocol (UDP) packets within the network domain of the areas to avoid congestion due to the multicasting of packets.

- When a robot arrives at a new area in the factory, it can receive multicasted UDP packets issued from a service discovery mechanism, e.g., Universal Plug and Play (UPnP), in the current area and learn the network address of the mechanism's directory server.
- The robot connects to the server and then informs its own addresses to the server.
- When the robot leaves the area, it can no longer connect to the servers that it connected to in the area and it also cannot receive any UDP packets issued from the area's service discovery mechanism.

Networked software running on transport robots can be classified into two kinds, i.e., client-side and server-side software, independently of the transmission protocol, e.g., Transmission Control Protocol (TCP) and UDP. To test client-side software for the discovery mechanism on a transport robot, the software needs to be executed within each of the network domains of the areas that the target robot may visit because multicasted UDP packets for the mechanism can be reached within the individual domains. When a transport robot discovers available services within its current network domain one the server-side, its software also needs to be executed to multicast UDP packets so that discover other robots or stationary servers within each of the network domains of the areas that the target robot may visit.

Some readers may think that even when the target software runs outside the areas, it can receive multicasted UDP packets via a tunneling technique. That is, we forward these packets from a target area to a computer that runs the software. However, there are firewalls in networks for reasons of security, and the cost of forwarding often affects time constraints in protocols, e.g., timeouts.

III. Design and Implementation

Developers are required to test their target software within each of the areas that their target robots may visit. However, it is difficult for developers to actually move or carry robots between areas and connect them to networks in a running factory. Our proposed testing framework is used to deploy and execute software that is designed to run on transport robots that change their current networks as they move. This framework has two key ideas. The first is to provide a target software with software-level-emulated execution environment in which the software should run. The second is to provide the software with an emulation of the physical mobility of a robot by using the software's logical mobility, which has been designed to run on robots over various networks. Physical mobility entails the movement and reconnection of mobile computing devices between sub-networks, while logical mobility involves software that migrates between hosts on sub-networks. The above emulator enables the target software to be execute within the emulation of a target robot and to directly connect to the external environment, such as the resources and servers provided in the networks that a robot connects to.

- The first is to use host-level virtual machines, e.g., VMWare and Hyper-V, and migrate the target software and operating systems from a virtual machine host to another host by using a technique, called *live migration*. The technique enables virtual machines to migrate to other machines to emulate the disconnection/reconnection of transport robots to networks within which multicast packets for plug-and-play protocols are transmitted to servers, stationary embedded computers, and other mobile or stationary robots.
- The second is to introduce an emulator for testing software with plug-and-play protocols running on language-level virtual machines, e.g., a Java virtual machine called *JVM*. The emulator can carry the target software between hosts by using a mobile agent technology. This is useful for testing application-level or middleware-level software.

The current implementation is based on the latter because the former needs high-speed networked storage systems, e.g., a Storage Area Network (SAN), which are expensive and used in data-centers rather than warehousing and manufacturing spaces. Our target software is also Java-based software to communicate with stationary servers through TCP, UDP, or upper layer protocols. Each emulator provides the target software with not only the internal environment of its own target robot but also the external environment, such as the resources and servers provided in the networks that the robot connects to. Our final goal is to emulate the reconnection of networked robots to networks managed by multicast-based management protocols by using virtual machine migration. In this paper we explain our approach on the basis of the second, i.e., mobile agent-based emulator, because the first and second are common and

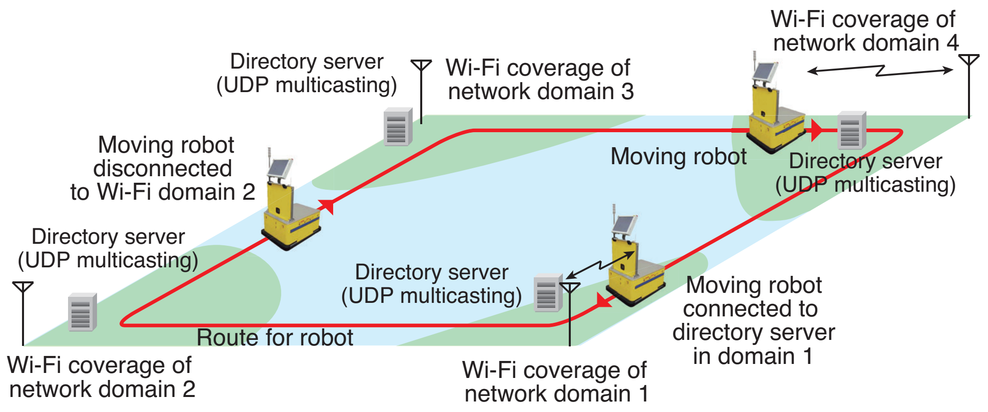

Wi-Fi coverage of network domain 4
Directory server (UDP multicasting)
Wi-Fi coverage of network domain 3
Moving robot disconnected to Wi-Fi domain 2
Moving robot
Directory server (UDP multicasting)
Directory server (UDP multicasting)
Directory server (UDP multicasting)
Moving robot connected to directory server in domain 1
Route for robot
Wi-Fi coverage of network domain 2
Wi-Fi coverage of network domain 1

Figure 1. Transport robot with WiFi in a factory

it is simpler to implement the second than the first. Physical mobility entails the reconnection of a robot to a network, while logical mobility involves a mobile agent-based emulator of a robot.

- Like virtual machines, this framework emulates its target robot.
- Depending on the reconnection of its target robot, the mobile agent-based emulator can carry software that should run on the computer on behalf of the robot to networks that the robot may be moved into and connected to.
- The emulator allows us to test and debug software with computational resources provided through its current network as if the software were being executed on the target robot when dynamically attached to the network.
- Software successfully tested in the emulator can still be run in the same way without being modified or recompiled.

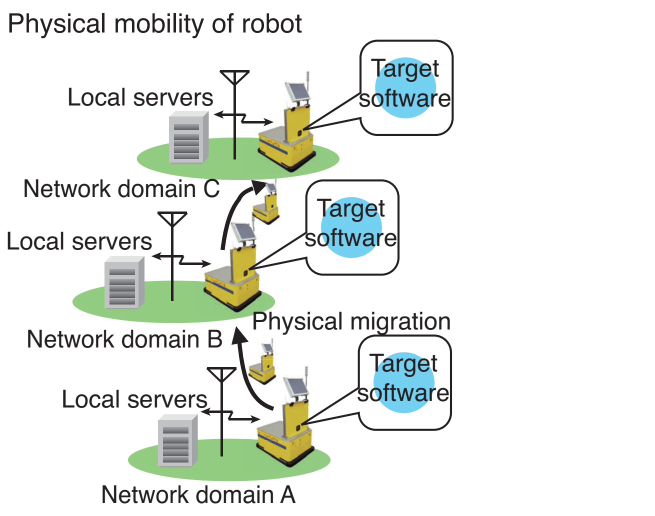
Each mobile agent is just a logical entity and must thus be executed on a computer. Therefore, this framework assumes that each of the sub-networks to which a device may be moved and attached to has more than one special stationary host, called an *access point host*, which offers a runtime system for executing and migrating mobile agent-based emulators. Each access point host is a runtime environment for allowing applications running in a visiting emulator to connect to local servers in its network. That is, the physical movement of a mobile computing device from one network and attachment to another is simulated by the logical mobility of a mobile agent-based emulator that carries the target applications from an access-point computer in the source network to another access-point computer in the destination network. As a result, each emulator is a mobile agent, and can thus basically not only carry the codes but also the states of the applications to the destination, so the carried applications can basically continue their processes after arriving at another host as if they had been moved with the target device.

The emulator delegates instruction-level emulation of target robots to JVM. In fact, each emulator permits its inner software to have access to the standard classes commonly supported by the JVM as long as the target robot offers them. The upper of Fig. 2 shows the physical mobility of robots and the lower of Fig. 2 shows the logical mobility of emulators.

In addition, each emulator offers its inner software as typical resources of the target robots. It can maintain a database to store files. Each file can be stored in the database as a pair consisting of a file/directory path name pattern and a content and provides its target software with basic primitives for file operation, e.g., file creation, reading, writing, and deletion. The framework provides the target software with two states in the lifecycle of the software running on the target robot, *networked running state* and *isolated running state:*. The former enables the target software to run within the target network domains, can link up with servers on the network through TCP and UDP and can send/receive UDP multicast packets. This state emulates that the robot is within the coverage area of one of the network domains provided through wireless networks. The latter runs the software but prohibits the software from communicating with any servers on the network. This state emulates a situation in which a robot is out any coverage areas of the network domains.

The framework provides an original runtime system for emulators by extending our existing mobile agent platform [12]. When an emulator with its target software is transferred over a network, the runtime system transforms the state and code of the agent, including its software, into a bitstream defined by Java's JAR file format, which can support digital signatures for authentication and transmit the bitstream to the destination host. Mobile agent-based implementation of the framework assumes that the target software is constructed as a set of Java bytecode, although its virtual machine-based implementation can support other software. Each emulator allows its target software to access most network resources from the host, e.g., the `java.net` package.

As mentioned in the first section, in an earlier version of this framework the target software must be client-side when communicating through TCP. The current implementation of this framework dynamically inserts a packet forwarding mechanism like Mobile IP [9] into the `java.net` package by using a bytecode level modification technique [1] when classes for TCP servers, e.g., `ServerSocket` and `InetAddress`, of `java.net`, are invoked from the target software. When wireless network domains overlap, robots may have more than one IP address. Our modified classes for IP addresses, e.g.,



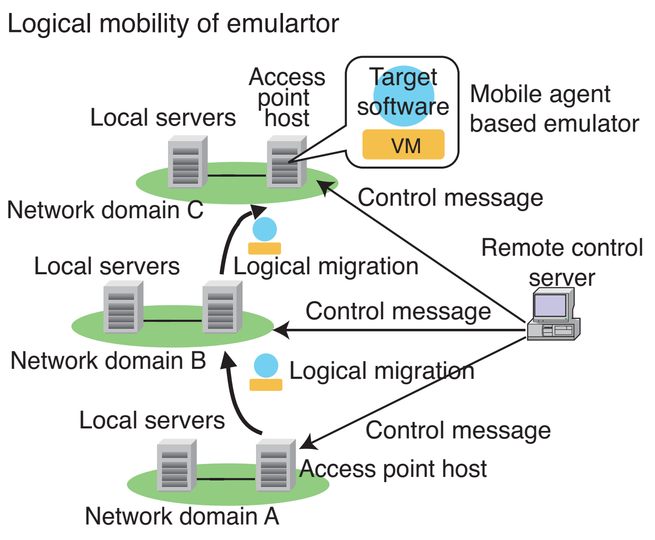


Figure 2. Physical mobility of robot (left) and logical mobility of emulator (right)

InetAddress, can return an IP address explicitly specified from developers.

IV. Experience

To illustrate the utility of the framework, in this section we present our experience with testing two typical kinds of software for networked transport robots.

A. *Testing software for transport robots*

In developing modern transport robots, we need to test transport robots with WiFi interfaces, which tend to be used in factories or warehouses (Fig. 3). We had five requirements:

- Each networked transport robot has an embedded computer (Intel Core i5, 2-GHz) with Linux and a WiFi interface.
- The factory has eight areas, where each area has its own wireless local area network provided through WiFi and provides directory servers available within the coverage space of the WiFi.
- Each robot discovers directory servers by receiving advertisement messages with their network addresses periodically issued from them through a UDP multicast-based original service discovery protocol available within the WiFi area of its current location.
- Each robot periodically updates its location to other robots or stationary servers within its current area through a TCP/IP-based original service discovery protocol.
- The coverage areas of the WiFi access points may overlap, and there are some spaces beyond the coverage areas of the WiFi access points.

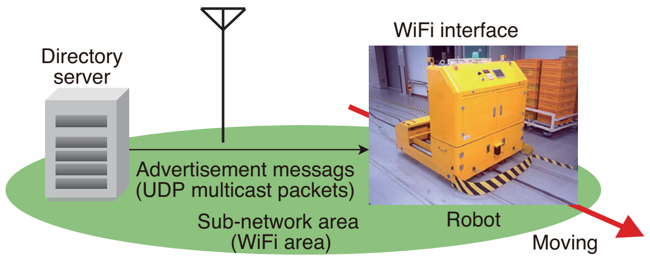

Directory server
WiFi interface
Advertisement messags (UDP multicast packets)
Sub-network area (WiFi area)
Robot
Moving

Figure 3. Communication between transport robot and directory server through WiFi

We tested two protocol stacks for the service discovery protocol through UDP multicast and session protocols between robots and directory servers by using the proposed framework. These protocols were constructed in Java so that we could directly use a mobile agent-based emulator based on JVM. To test the protocol stacks running on the client-side, i.e., robots, we customized a mobile agent-based emulator for the target robots. The emulator provided virtual I/O to control the movement of a robot for its target software, but it carried the software to a host within the target areas and enabled the software to receive UDP multicast packets, which were reachable within the area, and directly connected to the servers.

The developer could instruct the emulator to migrate to access-point hosts on the sub-networks of other areas. Also, since the emulator could define its own itinerary in the areas, it could precisely trace the movement of each robot. It could carry the target software, including the protocol stacks, to access-point hosts in the areas. It could continue to run the software in the local area network and permitted the software to directly receive UDP multicast packets, which servers only transmitted within the domains of the local area networks. We measured the processing overhead of the emulator, but the performance of software running in an emulator on an access-point host was not inferior to that of the same software running on the target robot, as long as the processing capability of the host was equivalent to that of the robot.

B. *Discussion*

While it was impossible to measure the framework's benefits quantitatively, it could eliminate the task of the developer having to carry and connect his/her target robot to local-area networks to verify whether software designed to run on the robot can successfully coordinate with servers or other robots. Let us now compare the framework with the other two existing approaches.

a) Comparison with field testing approach: This approach involves the developer carrying computers running the emulator of a target robot and testing the target software in the emulator within the local-area network at the current location. The developer can stay in front of his/her computer and directly view and operate the graphical user interface of a map viewer application on the computer. Like our framework, this approach permits the target software to receive packets that the location information servers multicast within the current local-area network because the software is running within the domain of the local-area. However, the developer carries the computer between places simply to check whether or not the software runs properly. This task is extremely cumbersome for the developer. Our framework, however, can replace the physical mobility of the developer with the logical mobility of an emulator of the robot and it thus enables the software to run and link up with servers within the local-area network.

b) Comparison with network-enabled emulator approach: A few emulators enable software to run on a local computer and link up with location information servers on target networks that their target devices may connect to through networks. Such existing emulators cannot send and receive packets beyond security mechanisms, e.g., firewalls. The cost of the approach is also inevitable in the sense that it often makes heavy traffics in networks, because packets transmitted only within the local-area networks at the location of a target robot tend to be much. The approach resulted in increased latency and network traffic in communication between the target application and servers, unlike ours, because the application in an emulator had to remotely communicate with the servers via routers and gateways, whereas the target robot could be directly connected to the servers. This is a serious problem in testing applications in gathering a large volume of data from servers, and vice versa.

V. Related Work

There have been many commercial and academic frameworks for simulating the target robots in virtual environments and for testing software for the robots in the environments. As far as we know, there is no paper on enabling software to be tested with networked environments that target robots may connect to.

Nevertheless, we discuss several existing approaches to testing software for robots. SITAF [14] is a framework for testing robot components by simulating environment. It generates test cases on the basis of specifications given by the developer. This test generation combined with simulation allows tests to be repeated. It also discards the need oto reuse tests, since they are generated. Biggs [3] presented testing software by using a repeatable regression testing method for software components that interact with hardware, but his approach focused only on individual components rather than whole robots. Among them, Chung et al. [5] showed experiments on applying International Organization for Standardization (ISO) for software testing (ISO 9126) to components for academic robotics. Laval et al. [7] proposed an approach to enabling the testing of not only isolated components but also whole robots. Their approach assumed standalone robots, so they did not support software for networked robots. Paikan et al. [8] proposed a generic framework for test driven development of robotic systems. It enabled functionalities to be tested but did not support any networking. Chen et al. [4] and Petters et al. [10] inserted an extra step in hybrid tests between simulation and tests based on three levels: component-level tests, online-level test with humans, and offline test (based on logs). Son et al. [15] proposed another three levels of tests: unit testing, state testing and API testing. However, their approaches did not support networked software running on robots. Laval et al. [7] proposed a safe-by-construction architecture based on a formal method instead of any testing approaches.

Reconnection and disconnection resulting from the movement of robots are similar to that when carrying portable computers, e.g., notebook PCs, tablets, and smartphones. There have been several attempts at testing software designed to run on portable computers. [2][6][16]. A typical problem in physical mobility is that the environment of a mobile entity can vary dynamically as the entity moves from one network to another.

VI. Conclusion

In this paper, we presented a framework for testing software running on networked transport robots, e.g., transport robots. The goal of the framework is to enable us to test networked software that reconnects and disconnects to the networks of the robots' destinations according the movement of the robots. It can emulate the physical mobility of target robots and enables software to directly connect to the networks of destinations in addition to the internal execution environment of the robots. Since our emulators were provided as mobile agents, which can travel between computers under their own control, they could carry and test software designed to run on their target robots in the same way as if they had been moved with the robots on which they were executed, and connected to services within their current local area networks. Our early experience with the prototype implementation of this framework strongly suggested that the framework could greatly reduce the time needed to develop and test software for networked industrial computers.

References

[1] Apache Software Foundation: "Byte Code Engineering Library," `http://jakarta.apache.org/bcel/`, October 2001.

[2] K. Beck: "Test Driven Development: By Example," Addison Wesley, November 2003.

[3] G. Biggs: "Applying regression testing to software for robot hardware interaction," In Proceedings of IEEE International Conference on Robotics and Automation (ICRA'2010), pp. 4621-4626, May 2010.

[4] I. Y. Chen, B. A. MacDonald, and B. C. Wunsche: "A flexible mixed reality simulation framework for software development in robotics," Journal of Software Engineering for Robotics, No.2, Vol.1, pp. 40-54, September 2011.

[5] Y. K. Chung and S. M. Hwang: "Software testing for intelligent robots," In Proceedings of International Conference on Control, Automation and Systems, pp. 2344-2349, October 2007.

[6] D. Gelperin and B. Hetzel: "The Growth of Software Testing," Communications of the ACM, Vol. 31, No. 6, pp. 687-695, June 1988.

[7] J. Laval, L. Fabresse, and N. Bouraqadi: "A methodology for testing mobile autonomous robots," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'2013), pp. 1842-1847, November 2013.

[8] A. Paikan, S. Traversaro, F. Nori, and L. Natale: “A Generic Testing Framework for Test Driven Development of Robotic Systems,” International Workshop on Modelling and Simulation for Autonomous Systems (MESAS 2015), pp. 216-225, Lecture Notes in Computer Science, vol. 9055. Springer, April 2015.

[9] C. Perkins, “IP Mobility Support”, Internet Request For Comments RFC 2002, October 1996.

[10] S. Petters, D. Thomas, M. Friedmann, and O. Von Stryk: “Multilevel testing of control software for teams of autonomous mobile robots,” Simulation, Modeling, and Programming for Autonomous Robots, pp. 183-194, November 2008.

[11] I. Satoh: “A Testing Framework for Mobile Computing Software,” IEEE Transaction on Software Engineering, Vol.29, No.12, pp. 1112-1121, December 2003.

[12] I. Satoh: “Mobile Agents,” Handbook of Ambient Intelligence and Smart Environments, pp. 771-791, Springer, October 2010.

[13] I. Satoh: “Testing software for networked industrial systems,” Proceedings of 39th Conference of IEEE Industrial Electronics Society (IECON’2013), IEEE Industrial Electronics Society, October 2013.

[14] H. Seong and J. Seok: “SITAF: simulation-based interface testing automation framework for robot software component,” In Florian Kongoli, editor, Automation. InTech, July 2012.

[15] J. Son, T. Kuc, J. Park, and H. Kim: “Simulation based functional and performance evaluation of robot components and modules,” In proceedings of International Conference on Information Science and Applications (ICISA’2011), pp. 1-7, May 2011.

[16] J. A. Whittaker: “What is Software Testing? And Why Is It So Hard?,” IEEE Software, pp. 70-79, January 2000.