Developing and Testing Networked Software for Moving Robots

Ichiro Satoh
National Institute of Informatics, 2-1-2 Hitotsubashi Chiyoda-ku Tokyo, 101-8430, Japan

Keywords: Transport Robot, Software Testing, Networed Software, Live Migration.

Abstract: Autonomous transport robots have been widely used to carry products in manufacturing and warehousing spaces. Such robots are smart and networked in the sense that they exchange information with stationary servers and other robots in their visiting spaces through wireless local-area networks. Therefore, when 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 autonomous transport robots by the logical mobility of software designed for running on computers by using a mobile agent technology. It enables such software to run within target network domains so that the software can locally access servers and receive multicast packets limited within the domains. It was evaluated with a practical example in a real factory.

1 INTRODUCTION

Automated vehicles, called *transport robots*, have been used to transport items in warehousing and manufacturing spaces. Most of vehicles tend to be rigid and static in the sense that they carry products among certain locations. Therefore, it is difficult for such robots to adapt demand-driven and dynamic environments. To solve this, modern transport robots become smarter in the sense that they are adaptive to changes in their environments and have the ability to exchange information on dynamic demands and environmental changes in their target spaces with stationary servers and other robots. In such robotics, their software plays a key role as it is the medium through which their autonomy and adaptation are embodied, because not only the hardware of such transport robots but also their software tend to be complicated. 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 are moving in such spaces. Furthermore, such smart transport robots often use service discovery mechanisms to find servers available in their current networks or notify their own presences or network addresses to servers or other robots through multicasts UDP packets, which are limited within within the network domain of the area to avoid congestion result from multicasting packets.

The purpose of this paper is to present a framework for testing software for designing to run on networked transport robots. It is constructed based on an early approach presented in our past paper (Satoh, 2015; Satoh, 2019). The goal of the previous approach enabled software designed to run on robots to connect to or disconnect from networks, which the robots may connect to or disconnect from as they move. The approach deployed and executed software at computers within the networks that target robots may connect to like the framework proposed in this paper. The framework presented in this paper is an extended system of the approach with the ability for its target software to receive and issue multicasting packets.

We here describe the motivation of the framework. A manufacturing company asked us about a method to test software designed for running on transport robots.[1] As mentioned in Section 2, they are using transport robots to carry products in their facto-

[1]We cannot specify the name of company due to their request.

ries. The approach aimed at testing client-side software running on mobile computers but server-side software are often running on transport robots. Therefore, although the framework presented in this paper is constructed based on the basic concept of the past approach, it is extended with several abilities to test software running on transport robots. Nevertheless, we do not intend the framework to be general. The framework aims 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 are not the scope of the framework.

The remainder of this paper is organized as follows. Section 2 describes an example scenario. Section 3 presents a framework for emulating the physical mobility of autonomous transport robots by the logical mobility of software designed for running on computers. 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.

2 REQUIREMENTS AND EXAMPLE SCENARIO

As mentioned in the previous section, our framework was inspired by practical problems 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 its local services provided only in the network. Transport robots moves 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 in the area. A service discovery mechanism in each area periodically multicasts UDP packets within the network domain of the area to avoid congestion due to multicasting packets.

- When a transport robot arrives at a new area in the factory, it can receive UDP-multicasting packets issued from a service discovery mechanism, *e.g.*, UPnP, and then it knows the network address of the mechanism's directory server.
- The robot connects to the server and then inform the addresses of its services to the server.
- When a robot leaves from an area, it can no longer disconnect to the servers that it connected to in the area and receive any UDP packets issued from the area's service discovery mechanism.

Transport robots run on only the routes that have floor-mounted induction loops for navigation. Networked software running on transport robots can be classified into two kinds of software, i.e., client-side and server-side software, independently of transmission protocols, *e.g.*, TCP and UDP. To test client-side software for the mechanism on the transport robot, it needs to be executed within each of the network domains of the areas that the target robot may visit because UDP multicast packets for the mechanism can be reached within the individual domains. When a transport robot discovers available services within its current network domain as a server-side, its software needs also to be executed and multicast packets for discovery mechanisms 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, the target software can receive UDP multicast packets by via a tunneling technique. That is, we forward these packets from the target area to the 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.

3 DESIGN

When testing networked software designed to run on transport robots, developers need to run 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 networks in a running factory. We propose a testing framework for deploying and running software, which is designed for running on transport robots that change their current networks according to their movement. This framework has two key ideas.

- The first is to provide the target software with an software-level emulation of the execution environment that 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 the robot over various networks.

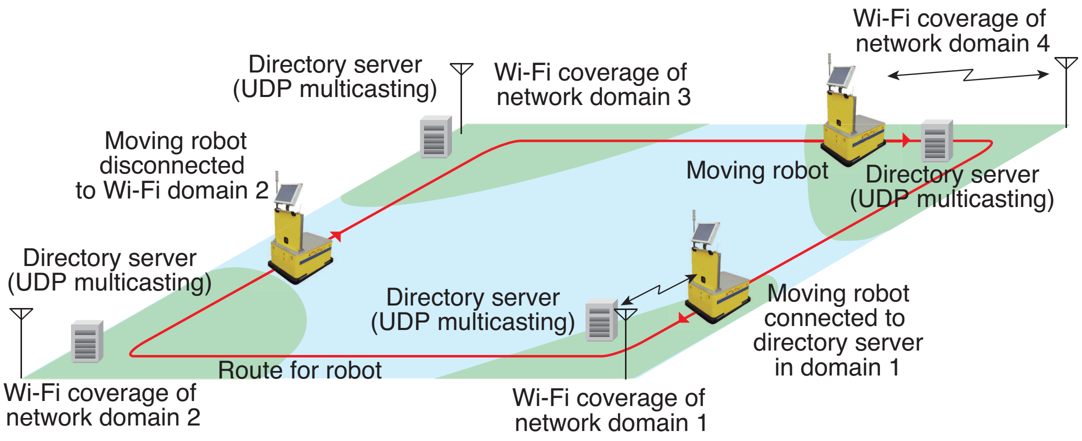
Wi-Fi coverage of network domain 4
Directory server (UDP multicasting)
Wi-Fi coverage of network domain 3
Moving robot disconnected to Wi-Fi domain 2
Moving robot
Directory server (UDP multicasting)
Directory server (UDP multicasting)
Directory server (UDP multicasting)
Moving robot connected to directory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

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 emulator enables the target software to be execute within an emulation of the target robot and to directly connect to the external environment, such as the resources and servers provided in the networks that the 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 virtual machine host to another host by using a technique, called *live migration*, which was often used in enterprise data centers. The technique enabled virtual machines to migrate to other machines to emulate the disconnection/reconnection of transport robots to networks within which multicast packets for service discovery mechanisms, e.g., plug-and-play protocols, are transmitted to servers, stationary embedded computers, and other transport robots.

The second is to introduce an emulator for testing software with plug-and-play protocols running on language-level virtual machines, *e.g.*, Java virtual machine called JVM. The emulator can carry the target software between hosts by using a mobile agent technology. It is useful to test application-level or middleware-level software.

The current implementation is based on the second, because our target software is Java-based software to communicate with stationary servers and other robots through TCP/IP or upper layer protocols. Therefore, the software can be tested with the second. The first also needs high-speed networked storage, storage area network (SAN), which are expensive and used in data-centers instead of warehousing and manufacturing spaces.

3.1 Emulator for Transport Robots

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. This paper explains our approach based on the second, i.e., mobile agent-based emulators, because the first and second are common and 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 the robot.

- Like virtual machines, this framework performs an emulation of 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.
- The 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 the 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 with 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 its 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 its targeted device.

The emulator delegates instruction-level emulation of target robots to Java VM. In fact, each emulator permits its inner software to have access to the standard classes commonly supported by the Java virtual machine as long as the target robot offers them. Figure 2 shows the physical mobility of robots and 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 its file/directory path name pattern and its 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 and 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 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 that the robot is out any coverage areas of the network domains.

3.2 Emulation of Mobility

The framework provides its original runtime system for emulators by extending our existing mobile agent platform (Satoh, 2006). When an emulator with its target software is transferred over a network, the runtime system transforms the state and codes of the agent, including its software, into a bitstream defined by Java's JAR file format, which can support digital signatures for authentication and then 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.*, `java.net` package.

As mentioned in the first section, in an early 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 (Perkins, 2002) into `java.net` package by using a bytecode level modification technique, when classes for TCP servers, *e.g.*, `ServerSocket` and `InetAddress`, of `java.net`, are invoked from the target software. When wireless network domains are overlapped, robots may have more than one IP address. Our modified classes for IP address, *e.g.*, `InetAddress`, can return the IP address explicitly specified from developers.

To operate the framework easily, we provide a control and monitor system. It has a graphical front-end to the framework. It allows us to monitor and operate the moving emulator and its target application by remotely displaying its graphical user-interfaces on its screen.

4 EXPERIMENT

Our experience presented in this section about testing of typical software for networked transport robots to illustrate the utility of the framework. In developing next generation automated guided vehicles for transport, i.e., transport robot, we need to test transport robots with WiFi interfaces, which tend to be used in factories or warehouses (Fig. 4). We had the five requirements:

- Each networked transport robot has an embedded computer (Intel Core i5 2.4GHz) with Linux and a WiFi interface (Fig. 3).
- The factory has eight areas, where each area has its wireless local area network through WiFi and provides a directory servers available within the coverage space of its 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

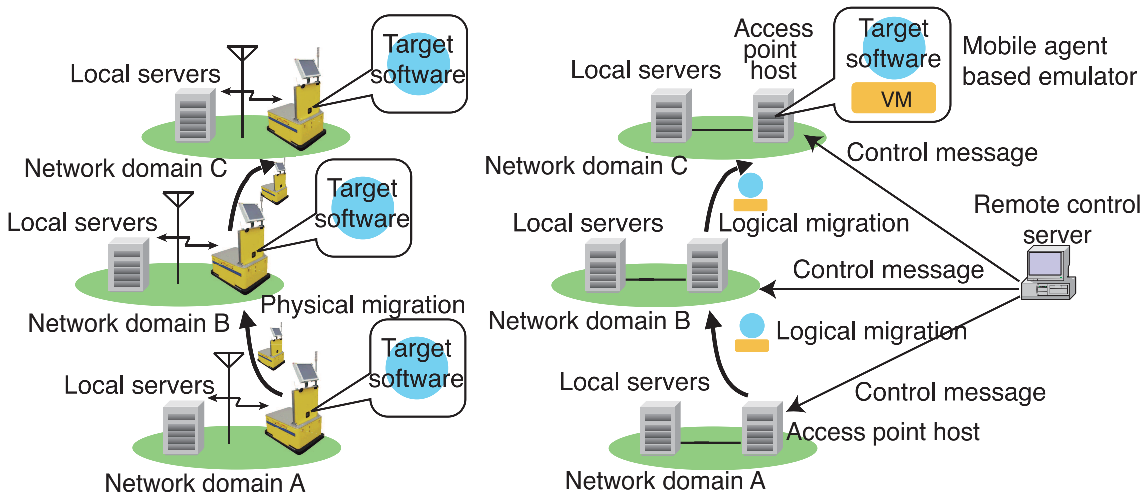

Figure 2: Physical mobility of robot (left) and logical mobility of emulator (right).

protocol available within the WiFi area of its current location.

- Each robot periodically updates its location to. The server that it connects to the server issues the locations of other robots within the servers' area to other robots through a TCP/IP-based original protocol.
- The coverage areas of the WiFi access points of areas may overlap and there are some spaces beyond the coverage areas of the WiFi access points.

Figure 3: Networked transport robot.

We tested two protocols stacks for the service discovery protocol through UDP multicast and TCP session protocols between robots and directory servers by using the proposed framework. These protocols were constructed in Java programs so that we could directly use a mobile agent-based emulator based on Java VM. 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

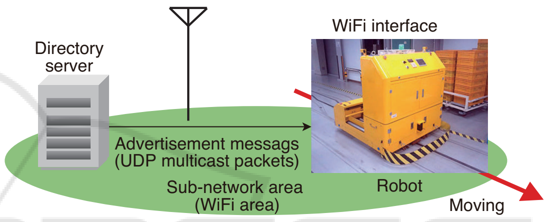
WiFi interface
Directory server
Advertisement messags (UDP multicast packets)
Sub-network area (WiFi area)
Robot
Moving

Figure 4: Communication between transport robot and directory server through WiFi.

of a robot for its target software, but carried the software to a host within the target areas and enabled the software to receive UDP multicast packets, which reached 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 areas, it could precisely trace the movement of each robot. The emulator 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. The cost of migrating the emulator with its target software between two access point hosts, which were connected through a 1 Gbps Ethernet, was 80 msec, where the access point hosts have Intel Xeon 2.8 GHz with 16 GB memory. 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.

5 RELATED WORK

Testing network protocols for non-robots is typically a manual process in which developers test the protocol behavior against various network conditions and configurations so that several researchers have studied approaches to test network protocols with focus on specific aspects of protocol behavior. To reason about the correctness of network protocols, prior work has employed a variety of program analysis techniques, such as model checking (Musuvathi and Engler, 2004; Sistla et al., 2000), static analysis (Feamster, 2004; Udrea et al., 2008), theorem proving (Wang et al., 2009) and refinement checking (Alur and Wang, 2001). However, they have not been designed for moving robots.

There have been many commercial and academic frameworks to simulate the target robots in virtual environments and to test software for the robots in the environments. As long as our knowledge, there is no paper on enabling the software to be tested with the networked environments that the target robots may connect to.

Nevertheless, we discuss on several existing approaches to test software for robots. SITAF (Park and Seok Kang, 2012) is a framework to test robot components by simulating environment. It generates test cases based on a specification given by the developer. This test generation combined with simulations allows repeatability of tests. It also discards the need of test reuse, since they are generated. Biggs (Biggs, 2010) presented on testing software for by using a repeatable regression testing method for software components that interact with its hardware, but his approach focused on only individual components rather than the whole robot. Among them, Chung et al.(Chung and Hwang, 2007) shows their experiments in their applying ISO standard for software testing (ISO 9126) to components for academic robotics. Laval et al. (Laval et al., 2013) proposed an approach to enabling to test not only isolated components, but also the whole robot. Their approach assumed standalone robots so that they did not support software for networked robots. Chen et al. (Chen et al., 2011) and Petters et al. (Petters et al., 2008) presented to insert an extra step for 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. () 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. (Laval et al., 2013) proposed a safe-by-construction architecture based on a formal method instead of any testing approaches.

The reconnection and disconnection result from the movement of robots are similar to those from the carrying of portable computers, *e.g.*, notebook PCs, tablets, and smartphones. There have been several attempts to test software designed to run on portable computers. (Beck, 2002; Gelperin and Hetzel, 1988; Whittaker, 2000). A typical problem in physical mobility is that the environment of a mobile entity can vary dynamically as it moves from one network to another. A lot of research has been proposed to either transparently mask variations in mobility at the network or system level or adapt this to the current environment at the application level (Noble et al., 1997; Perkins, 2002). Nevertheless, current work on these approaches has focused on a location-transparent infrastructure for the applications and location-aware applications themselves. As a result, the task of building and testing software has attracted only limited attention.

There have been a few attempts to test software designed to run on portable computers instead of robots. Several researchers have explored approaches to enabling software to run on local computers and link up with remote servers through networks to access particular resources and services provided by remote networks; *e.g.*, the InfoPad project at Berkeley (Le et al., 1994) and the network emulator of Lancaster University (Davies et al., 1995). However, accomplishing this in responsively and reliably is difficult, and the emulators cannot remotely access all the services and resources that are only available within the network domains because of security, such as firewalls. Moreover, the approach is inappropriate for testing software using service discovery protocols.

6 CONCLUSION

This paper presented a framework for developing and testing software running on autonomous transport robots, which were often used in warehousing and manufacturing spaces. Its goal was to enable us to test networked software that reconnects and disconnects to the networks of their destinations according the movement of transport robots. It could emulate the physical mobility of the target robots and enabled the software to directly connect to the networks of their destinations in addition to the internal execution environment of the robots by using the logical mobility of emulators corresponding to the the target robots. We designed and implemented an emulator based on mobile agents and a virtual machine. Each emulator could emulate its target robot. Since they were provided as mobile agents, which can travel between computers, 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 executed them, 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

Alur, R. and Wang, B.-Y. (2001). Verifying network protocol implementations by symbolic refinement checking. In *Proceedings of the 13th International Conference on Computer Aided Verification*, CAV '01, pages 169–181. Springer-Verlag.

Beck, K. (2002). *Test Driven Development. By Example (Addison-Wesley Signature)*. Addison-Wesley.

Biggs, G. (2010). Applying regression testing to software for robot hardware interaction. In *ICRA*, pages 4621–4626. IEEE.

Chen, I. Y.-H., MacDonald, B. A., and Wunsche, B. C. (2011). A flexible mixed reality simulation framework for software development in robotics. *Journal of Software Engineering for Robotics*, 2(1):40–54.

Chung, Y. K. and Hwang, S.-M. (2007). Software testing for intelligent robots. In *2007 International Conference on Control, Automation and Systems*, pages 40–54. IEEE.

Davies, N., Blair, G. S., Cheverst, K., and Friday, A. (1995). A network emulator to support the development of adaptive applications. In *Proceedings of the 2nd Symposium on Mobile and Location-Independent Computing*, MLICS '95, pages 47–56, Berkeley, CA, USA. USENIX Association.

Feamster, N. (2004). Practical verification techniques for wide-area routing. *SIGCOMM Comput. Commun. Rev.*, 34(1):87–92.

Gelperin, D. and Hetzel, B. (1988). The growth of software testing. *Commun. ACM*, 31(6):687–695.

Laval, J., Fabresse, L., and Bouraqadi, N. (2013). A methodology for testing mobile autonomous robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1842–1847.

Le, T., Seshan, S., and Burghardt, F. (1994). Software architecture of the infopad system. In *In Mobidata Workshop*, page 93.

Musuvathi, M. and Engler, D. R. (2004). Model checking large network protocol implementations. In *Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation - Volume 1*, NSDI'04, pages 12–12. USENIX Association.

Noble, B. D., Satyanarayanan, M., Narayanan, D., Tilton, J. E., Flinn, J., and Walker, K. R. (1997). Agile application-aware adaptation for mobility. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, SOSP '97, pages 276–287. ACM.

Park, H. and Seok Kang, J. (2012). Sitaf: Simulation-based interface testing automation framework for robot software component.

Perkins, C. (2002). Ip mobility support for ipv4. Technical report, RFC, United States.

Petters, S., Thomas, D., Friedmann, M., and von Stryk, O. (2008). Multilevel testing of control software for teams of autonomous mobile robots. In Carpin, S., Noda, I., Pagello, E., Reggiani, M., and von Stryk, O., editors, *Simulation, Modeling, and Programming for Autonomous Robots*, pages 183–194, Berlin, Heidelberg. Springer Berlin Heidelberg.

Satoh, I. (2006). *Mobile Agents*, pages 231–254. Springer US, Boston, MA.

Satoh, I. (2015). An approach for developing software on robots (short paper). In *International Workshop on Factory Control Systems, 2015*, pages 1–2.

Satoh, I. (2019). An approach to testing software on networked transport robots. In *5th International Conference on Advances and Trends in Software Engineering (SOFTENG 2019), to appear*.

Sistla, A. P., Gyuris, V., and Emerson, E. A. (2000). Smc: A symmetry-based model checker for verification of safety and liveness properties. *ACM Trans. Softw. Eng. Methodol.*, 9(2):133–166.

Udrea, O., Lumezanu, C., and Foster, J. S. (2008). Rule-based static analysis of network protocol implementations. *Information and Computation*, 206(2):130 – 157. Joint Workshop on Foundations of Computer Security and Automated Reasoning for Security Protocol Analysis (FCS-ARSPA '06).

Wang, A., Basu, P., Loo, B. T., and Sokolsky, O. (2009). Declarative network verification. In *Proceedings of the 11th International Symposium on Practical Aspects of Declarative Languages*, PADL '09, pages 61–75. Springer-Verlag.

Whittaker, J. A. (2000). What is software testing? and why is it so hard? *IEEE Software*, 17(1):70–79.