Network Simulation Architecture for Smartspace

Morihiko Tamai, Naoki Shibata[†], Keiichi Yasumoto and Minoru Ito

Graduate School of Information Science, Nara Institute of Science and Technology,
Ikoma, Nara 630-0192, Japan e-mail: {morihi-t,yasumoto,ito}@is.naist.jp
[†] Department of Information Processing and Management, Shiga University,
Hikone, Shiga 522-8522, Japan e-mail: shibata@biwako.shiga-u.ac.jp

Abstract. In this paper, we propose a network simulator for smartspace. Our simulator provides the following functions. (1) Realistic simulation of wireless communication; (2) Software compatibility for allowing code for real device to run on the simulator without modifying it; (3) Simulating network consisting of virtual and real devices. To realize the above (1), the simulator simulates radio propagation taking into account the influence of obstacles on a smartspace. For (2), the simulator provides APIs for simulation compatible with APIs for real devices. For (3), the simulator bridges communication between virtual and real devices using NAT. In this paper, we describe the above functions and results of a performance evaluation.

1 Introduction

Testing applications which run on smartspace environment is difficult and expensive, since test examiners have to install many appliances and sensors in a testbed, and have participants perform a huge set of actions on it. Testing these applications on a simulator is one of the promising solutions to this problem. The following functions are desired for smartspace simulators. (1) Arrangement of virtual devices and visualization of device states in a 3D space; (2) Simulation of wireless communication taking into account influences of obstacles such as walls and furniture; (3) Simulation of changes in physical quantities such as room temperature changed by air conditioning; (4) Mechanisms to execute software for real devices as virtual devices without large modifications. Users of simulator would sometimes want to test operation of some of the devices through human perception. For such a purpose, we need (5) mechanisms to allow virtual and real devices communicate with each other. In order to automate some part of testing applications, we need (6) systematic generation of test sequences.

To realize the functions (1) to (6) mentioned above, we have designed and implemented a smartspace simulator called UbiREAL[1]. The functionalities of UbiREAL about (1), (3) and (6) are described in [1]. In this paper, we focus on the functionalities of (2), (4), and (5). Many of existing studies use ns-2 or QualNet as network simulator for smartspace. These simulators are capable of reproducing wireless and wired communication between devices. However, the functions (3) to (6) are not considered.

The following Section 2 describes an overview of UbiREAL. Section 3 presents the functions of our network simulator. Section 4 evaluates the performance of the proposed simulator, and Section 5 concludes the paper.

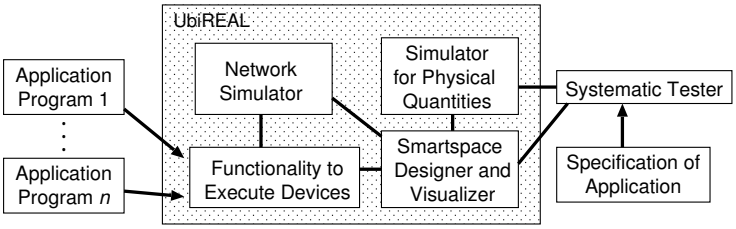

Fig. 1. Modular structure of UbiREAL

2 Overview of UbiREAL

The modular structure of UbiREAL is shown in Fig. 1. In order to make users easily construct a virtual smartspace, UbiREAL provides a graphical tool (called *smartspace designer*) for designing smartspace with which users can intuitively specify the locus of room arrangement in the smartspace, types and positions of devices, and behavior of virtual inhabitants. In order to simulate behavior of devices which give/receive influence to/from physical quantities such as temperature or brightness, UbiREAL provides functions to simulate changes of these physical quantities in consequence of operations of devices like air-conditioner. Testing ubiquitous applications is hard since they have to be tested under a huge number of arrangement patterns of devices, initial values of physical quantities and behavior of inhabitants. UbiREAL provides systematic testing function which generates various test patterns automatically (see [1] for details). Communications between devices are simulated by the network simulator.

3 Architecture of Network Simulation

UbiREAL network simulator simulates protocol stacks from the physical layer to the presentation layer. In the physical and MAC layers, both wired (IEEE802.3) and wireless (IEEE802.11, ZigBee, and Bluetooth) protocols are simulated. As for the network layer, it supports AODV to simulate wireless ad-hoc networks. As for the transport layer, it supports TCP and UDP.

A physical network topology to be simulated can be specified using the smartspace designer (see Fig. 1). Configurations of virtual switching hubs and wireless access points can be also specified by the designer. Virtual devices can be selected from a pulldown menu, and placed by drag-and-drop. To specify a wired network topology, Ethernet cables are specified by drawing lines between virtual devices. Network configurations of each device such as IP and MAC addresses, and ESSID of wireless LAN network can be also specified using the designer.

The network simulator provides the following functions: (1) **Realistic simulation of wireless communication** As for the simulation of indoor wireless communications, free space radio propagation model which does not take obstacles into account is insufficient in terms of accuracy. The proposed simulator implements radio propagation models more suited to indoor simulation. (2) **Software compatibility** The simulator provides source level software compatibility which allows code for real device to run on the simulator without large modification. (3) **Bridging communications between real and virtual devices** To simulate smartspace, appliances which change operations based on physical quantities such as sound and temperature have to be simulated. It

would be also useful to test operations of some appliances by human perception using real devices. For these purposes, the proposed simulator has the function to bridge communications between virtual and real devices.

In the following subsections, we explain main functions of the proposed simulator.

3.1 Simulation of Wireless Communication

Radio propagation models are used to reproduce the reachability of wireless packets between devices. The supported models include free space model, line-of-sight model, and ray-tracing model[2]. In free space model, the received signal power $P_r(d)$ at the distance d from the transmitter is represented by the following equation: $P_r(d) = P_t\lambda^2/(4\pi)^2d^2$. Here, P_t and λ are the transmitted signal power and the wavelength, respectively. In the free space model, the obstacles in the smartspace are not considered. In the line-of-sight model, when there is no obstacles on the line-of-sight between the transmitter and the receiver, the same propagation model as free space model is used. Otherwise, all communications are blocked. In ray-tracing model, radio wave from transmitter is represented as multiple rays, and the rays which reach to a receiver directly and indirectly by reflection against walls/floor are traced. The received signal power at receiver is represented as the sum of the signal power of those rays. The parameters required by the propagation model are given by the configuration file.

If there is no device which moves around the smartspace, radio propagation is calculated beforehand at the start-up time of the simulator, and the simulation with line-of-sight and ray-tracing models can run as fast as that with free space model. If there are some movable devices, radio propagation is calculated periodically during simulation. The period should be determined considering tradeoff between calculation overhead and simulation accuracy.

3.2 Source Code Compatibility

UPnP is becoming a de facto standard for cooperating appliances and other devices for ubiquitous applications. Our simulator enables application programs developed using UPnP library (such as Libupnp[3] and CyberLink for Java[4]) to run on the simulator without large modification.

In order to keep software compatibility between application programs which run on a real device and on the simulator, the simulator provides a library which has the same APIs as the communication library used in the UPnP library. Ordinary BSD socket library and java.net package can be used as the communication libraries. Application code is linked with the simulator's library (called *socket stub*) at compile-time or run-time. A socket stub is located between an application program and the network simulator, and it forwards method calls (e.g., *bind*, *connect*, etc.) of the BSD socket (or java.net) library from the application program to the network simulator. The forwarded calls are transmitted through TCP connections between socket stub and the simulator. When a method name and its arguments are received by a socket stub, the simulator adds an event corresponding to the method into the event queue. Each event in the queue is processed at a predetermined time. As a result of event processing, when a return value to the caller arises, the data is passed to the application program through the socket stub.

3.3 Communication between Virtual Device and Real Device

To realize communication between virtual devices and real devices, the network simulator executes network address translation (NAT) process. When a packet transmitted from a virtual device is transmitted towards a real device, the NAT process rewrites the source address of the packet to the address of the host running the simulator (called host A). Similarly, the source port number will be rewritten to one of the unused port number of the host A. When a packet is transmitted from a real device to a virtual device, the packet is rewritten similarly. In UPnP, since a network address and a port number are embedded in SSDP and SOAP messages, the NAT process also rewrites them.

4 Performance

In this section, we describe the experimental results of performance evaluation. In the experiment, we investigated whether the simulator can carry out the simulations of communications between devices in practical time.

The setup of the experiment is as follows. A temperature sensor and an air-conditioner are installed in a virtual smartspace. The air-conditioner subscribes the event of changes in sensor's value using UPnP protocol. The sensor updates its value every second, and the air-conditioner acquires the value of temperature sensor every second. A *response time* is defined as the period from the time when the sensor updates its value, to the time when the air-conditioner receives the value. We increased the number of pairs of sensor and air-conditioner from 1 to 30, and observed response time. In the experiment, we assumed all devices are connected to a network via an Ethernet hub, and devices communicate with each other by standard protocols in UPnP framework. In the experiment, the network simulator only emulated protocol layers above session layer, which includes the set of UPnP protocols. Response times were measured after all air-conditioners finish event subscription. The machine used for the experiment is as follows: CPU: AMD Athlon64 3400+, memory: 1GB, OS: Linux 2.6.8, Java: J2SE 5.0.

From the experiment, we confirmed that the average response time is 5 ms when the number of pairs of a sensor and an air-conditioner is 1, and that the average response time is 8ms or less even if the number of pairs is 30. Since each device just acquires a value approximately once every second, the proposed simulator performed the simulation with sufficient speed.

5 Conclusion

In this paper, we described the functions of proposed network simulator and results of its performance.

References

1. H. Nishikawa, S. Yamamoto, M. Tamai, K. Nishigaki, T. Kitani, N. Shibata, K. Yasumoto, and M. Ito, UbiREAL: Realistic Smartspace Simulator for Systematic Testing, to appear in the 8th Int'l Conf. on Ubiquitous Computing (UbiComp2006), 2006.
2. S. Fortune, Algorithms for prediction of indoor radio propagation, Technical Report, Bell Laboratories, 1996.
3. libupnp, http://upnp.sourceforge.net/.
4. CyberLink for Java, http://www.cybergarage.org/net/upnp/java/index.html.