UbiREAL: Realistic Smartspace Simulator for Systematic Testing

Hiroshi Nishikawa[1], Shinya Yamamoto[1], Morihiko Tamai[1], Kouji Nishigaki[1], Tomoya Kitani[1], Naoki Shibata[2], Keiichi Yasumoto[1], and Minoru Ito[1]

[1] Graduate School of Information Science, Nara Institute of Science and Technology
8916-5, Takayama, Ikoma, Nara 630-0192, Japan
{hirosh-n, shiny-ya, morihi-t, koji-ni,
t-kitani, yasumoto, ito}@is.naist.jp
[2] Department of Information Processing and Management, Shiga University
Hikone, Shiga 522-8522, Japan
shibata@biwako.shiga-u.ac.jp

Abstract. In this paper, we propose a simulator for facilitating reliable and inexpensive development of ubiquitous applications where each application software controls a lot of information appliances based on the state of external environment, user's contexts and preferences. The proposed simulator realistically reproduces behavior of application software on virtual devices in a virtual 3D space. For this purpose, the simulator provides functions to facilitate deployment of virtual devices in a 3D space, simulates communication among the devices from MAC level to application level, and reproduces the change of physical quantities (e.g., temperature) caused by devices (e.g., air conditioners). Also, we keep software portability between virtual devices and real devices. As the most prominent function of the simulator, we provide a systematic and visual testing method for testing whether a given application software satisfies specified requirements.

1 Introduction

It is one of the most important challenges to realize Smartspace environments which provide people useful services by making embedded devices cooperate based on contexts. In order to realize Smartspace, we need ubiquitous application softwares which control many information appliances based on contexts and user preferences. We also need ubiquitous sensor networks as infrastructure for these applications. There are many studies on realizing middleware for facilitating development of ubiquitous applications and testbeds for testing if those applications work expectedly [1–4].

Since applications running on Smartspace have influence on convenience and even safety of our daily life, those applications must be developed carefully so that they run expectedly and safely. However, it is difficult and expensive to test them thoroughly in real world environments, since test examiners have to assemble testbeds using various types of sensors and information appliances and generate

a quite large number of contexts for tests where each context consists of user locations/behavior, time, and so on. Also, there are so many possible deployment patterns of sensors and appliances. In addition, users of Smartspace may want to know how devices work based on typical contexts and their preferences in advance and change deployment of devices, control policies (rules) and/or preferences to find the best configurations to fit their life styles. However, this kind of trial and error burdens users too much if it is conducted in real world environments.

For evaluating protocols in large scale wired and/or wireless networks, network simulators such as ns-2 and QualNet are widely used. Those network simulators can be used to evaluate only aspects of communication between devices of Smartspace. So, in order to cope with the above problems, we need a simulator for realistically simulating Smartspace environments. For this purpose, the following criteria should be satisfied with a Smartspace simulator: (1) *Support to design Smartspace* which allows test examiner to easily deploy networked devices in a freely designed 3D virtual space; (2) *Realistic context generation* which generates various contexts based on user behavior and device actions/communications in the virtual space; (3) *Graceful visualization* which intuitively informs test examiner of how devices work based on contexts through visual animations; (4) *Software compatibility* which allows software and protocols (such as device drivers) to run on the virtual space as well as in real world environments; and (5) *Systematic testing* which systematically generates possible contexts and checks whether the system runs expectedly.

There are several studies to realize a Smartspace simulator such as UBIWISE [5] and TATUS [6]. These existing simulators partly achieve the above criteria (1), (3) and (4). However, the important criteria (2) and (5) are not realized.

In this paper, we propose a Smartspace simulator called *UbiREAL* (Ubiquitous Application Simulator with REAListic Environments) which provides a virtual testbed for ubiquitous applications in a 3D space.

For the criteria (1)-(5), UbiREAL provides the following functions: (i) arrangement of devices in a 3D space by GUI; (ii) simulation of wired and wireless communication between devices (i.e., sensors and information appliances) from MAC layer to application layer; (iii) emulation of temporal transition of physical quantities in a space; (iv) visualization of device states by 3D animations; and (v) systematic tests for a given application under test and deployment of devices in a space. The above function (ii) realizes the software compatibility (criterion (4)), and (iii) realizes the realistic context generation (criterion (2)). These functions cooperate to achieve the systematic tests (criterion (5)).

The rest of the paper is composed as follows. Section 2 briefly describes the related work. In Section 3, we explain overall structure of UbiREAL. In Sections 4, 5, 6 and 7, we present the details of UbiREAL, that is, device arrangement and visualization in a virtual Smartspace, simulation of communication, simulation of physical quantities, and systematic tests, respectively. Experimental results for UbiREAL performance are shown in Section 8. Finally, Section 9 concludes the paper.

2 Related Work

There are many research efforts which aim to facilitate development of ubiquitous applications and/or improve reliability of those applications. The existing studies are largely classified to three groups treating middleware, testbeds and simulators.

Firstly, as middleware and/or framework for efficient development of ubiquitous applications, many studies such as Refs. [1, 7, 8] have been researched.

Biegel et al. have proposed middleware which controls sensors, actuators and application components in an event-driven manner, in order to facilitate development of mobile and context-aware applications [1]. Niemelä et al. addressed that it is important to achieve inter-operability, adaptability and scalability among components for agile pervasive computing, and they proposed an architecture achieving them [7]. Roman et al. have proposed middleware called "Gaia", which provides various services in Smartspace [8]. In addition, Chan et al. have proposed a J2ME-based middleware with micro-server proxy for thin clients to use services of Gaia [9]. Messer et al. have developed middleware which integrates different CE (Customer Edge) devices, where it allows users to only choose what they want to do through comprehensive pseudo-English interface when using the middleware [10]. Nishigaki et al. have proposed a language called "CADEL" which facilitates rule description by defining complex conditions/contexts as simple phrases in natural language and developed a framework for context-aware applications with information appliances which aim to allow ordinary home users to easily configure device controls [2]. These researches for middleware mainly focus on facilitating application development and increasing availability of various devices and usability of systems. Thus goals of these middleware overlap with our goal. Since these existing middleware can be applications of our UbiREAL simulator, we can say that the middleware researches and UbiREAL research are mutually complementary.

Secondly, there are several existing studies concerning testbeds of ubiquitous applications [11, 12]. Consolvo et al. have evaluated advantages and drawbacks of existing methods by experiments designing a Smartspace called Labscape [11]. Nakata et al. have noticed that a simulation to make testbeds is very hard, and thus proposed a simulation method which virtually executes not only virtual devices but also real devices in the simulation [12].

These researches on testbeds are important for ubiquitous applications to be developed efficiently and improve reliability, but it would be hard and costly to construct various configurations of devices in a real environment for tests. For this problem, Ref. [12] adopts an approach similar to UbiREAL, that is, to execute real devices and virtual devices cooperatively with interactions. However, the project is in early phase, and the paper describes only rough ideas without detailed implementation or method.

Thirdly, as studies concerning simulations of ubiquitous environments, Refs. [5, 6, 13, 14] have been proposed. Hewlett-Packard Laboratories have proposed a simulator called *UBIWISE* [5]. UBIWISE is designed for development of prototypes and for tests of new hardware device and its software in the virtual ubiq-

uitous computing environment. UBIWISE consists of two simulators; UbiSim, which generates 3D virtual space, and WISE, which displays how devices are running through 2D visual graphics. In addition, the research team in Trinity College has proposed the simulator called *TATUS* [6]. TATUS can simulate a ubiquitous application in a 3D virtual space. TATUS has a wireless network simulator in it. However, TATUS is developed with 3D-game-engine, and so the tester has to operate and move a user-character in the virtual space or let multiple characters move based on simple AI. TATUS also simulates packet loss ratio in wireless communication when characters frequently cross line-of-sight between devices. However, it is not mentioned in detail how to simulate other physical quantities in a virtual space in response to human behavior and/or device states. Sanmugalingam et al. have proposed an event simulator which aims to visualize and test scalability of given location-aware applications developed based on the proposed event-driven middleware [13]. Roy et al. showed that it is very important to track location of inhabitants in order to generate various contexts in smarthome, but they also showed that an algorithm for tracking locations of several inhabitants in an optimal way is NP-hard [14]. As a result, they have proposed a framework which stochastically generates likely interaction patterns between the inhabitants based on the game theory.

Those existing simulators simulate only restricted cases of inhabitants' behavior because they achieve the behavior manually or based on simple automation. Thus, they are not enough to confirm that a given ubiquitous application program runs as expected for all possible context patterns. They neither simulate it accurately that inhabitant and device behavior changes physical quantities such as temperature in the target space. So, it would be hard to guarantee validity of the given application implementation.

3 UbiREAL Overview

In this section, we briefly outline overview of proposed UbiREAL simulator.

3.1 Objectives and Applications

We first describe a typical ubiquitous application example which automatically controls home appliances based on contexts. We assume a house consisting of a living room and several other rooms, where three inhabitants Alice and her parents are living. We also assume that lamps, TV, video recorder, stereo and air conditioner as well as various sensors are deployed in the living room, and they can be controlled through network installed at the house. We suppose to control those appliances as follows.

– When Alice enters living room, illumination of the room is turned down, and the stereo begins playing jazz music quietly. The air conditioner is automatically configured at 25 degrees Celsius of temperature and 50% of humidity.

- Then, her father and mother come into the room. This time, the illumination is set bright, and the air conditioner is configured at 27 degrees Celsius and 60% of humidity. The stereo begins playing classic music.
- When baseball broadcast begins, TV is automatically turned on with the baseball channel, if father is in the living room. If father is not present, the video recorder starts recording the program.

When home appliances are controlled in the way described above, we have to consider the cases where several inhabitants (application user) have different preferences of the settings of air conditioner, TV programs or so on. The sound produced by stereo and TV can conflict with each other. Lamps should not be controlled so that it continuously repeat turning on and off. Air conditioner and heater in a room should not be turned on simultaneously. Situations like these may occur unintentionally when brightness or temperature is within a specific range.

As seen above, we have to test if home appliances are controlled as application users intended. But, finding all possible glitches is difficult by only observing the application behavior for several patterns. Testing rules under many possible contexts using real appliances in a real house requires tremendous labor and time.

Our UbiREAL simulator has been designed and developed to simulate and test ubiquitous applications with various contexts. The objectives of developing UbiREAL include the followings in addition to resolving the problem described above.

(1) To offer a way to inhabitants (application users) to intuitively check how devices work

If home appliances are controlled by rules like the case described above, the rules are sometimes specified from the scratch and/or modified by application users themselves. In order to allow ordinary people to manipulate rules, the system should offer a 3D view of a virtual space with which users can check places and operations of virtual devices. When the system finds situations (contexts) that some rules do not work as expected, those situations and wrong operations should be shown to users through 3D view.

(2) To enable cooperation between virtual devices and real devices

Even if there is no software problem, devices can operate incorrectly due to hardware failures or other problems. In such a case, it would be convenient if parts of virtual devices are replaced with real devices. This function also allows users to perceive how the real devices work based on the context in accordance with the (virtual) surrounding environment. For example, by applying this function to air conditioners or lamps, users can perceive the effective temperature or brightness. It greatly helps users to specify rules so that services are given as expected.

(3) To make device drivers (software) for real devices executable as virtual devices without large modification

When devices are simulated on existing network simulators like ns-2, the device drivers have to be written in event-driven way, and thus drivers for real devices, which are usually written in flow-driven way, cannot be executed "as-is". Writing driver software for various virtual devices from scratch costs huge amount

of labor. Moreover, since drivers for virtual devices and real devices are different, even if virtual devices are confirmed to work correctly, we cannot guarantee that real devices should also work correctly.

(4) To allow testers to systematically test the whole application system by automatically generating many possible contexts

Even using simulation, it would be hard task to generate many possible contexts by hand and check whether devices work as expected by simulating the given application for those contexts.

3.2 UbiREAL Structure

We compose UbiREAL simulator of the following four parts as shown in Fig.1: (1) Visualizer and GUI for designing virtual Smartspace; (2) Network simulator; (3) Simulators for physical quantities; and (4) Systematic tester.

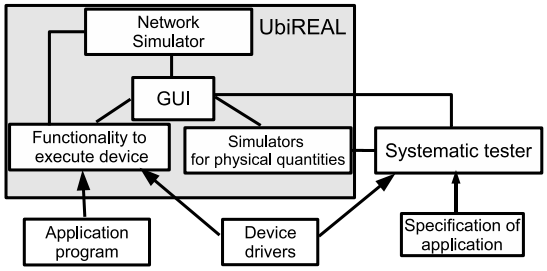


Fig. 1. UbiREAL Architecture

Basically, we allow UbiREAL to execute any device driver software [1] developed for real devices and application program developed for devices (or in a home server) to be executed as virtual devices without large modification. For this purpose, UbiREAL is designed so that general communication protocols and APIs for information appliances within UPnP framework such as SOAP and SSDP can be used on it. However, for visualization purpose, some devices have to send all states to the simulator when requested. So, software drivers for such devices have to be modified slightly. Devices with switches manually manipulated by users also have to send a signal to the simulator when states of these switches are changed. The states obtained from devices are used for visualization and tests.

Hereafter, we describe the four main functions of UbiREAL.

[1] We suppose that the device driver software is executed at user mode of operating system. It commits to OS operations for controlling its locally attached actuators or obtaining values from its sensors using, e.g., UPnP library, and exchanges messages with the device driver software running on other devices.

Fig. 2. Specifying Route of Avatar

Visualizer and GUI for designing Smartspace This function helps user to create virtual home and rooms in which virtual devices are placed. Also, this function helps designing routes which (virtual) application users and other movable devices trace. Details are described in Section 4.

Network simulator Network simulator simulates communication between virtual devices as well as between virtual and real devices, taking into account their positions and obstacles on their line-of-sights. Details are described in Section 5.

Simulators for physical quantities In order to reproduce temporal variation of physical quantities such as temperature, illumination and loudness in a specific region in the virtual Smartspace, a dedicated simulator is prepared for each physical quantity. Details are described in Section 6.

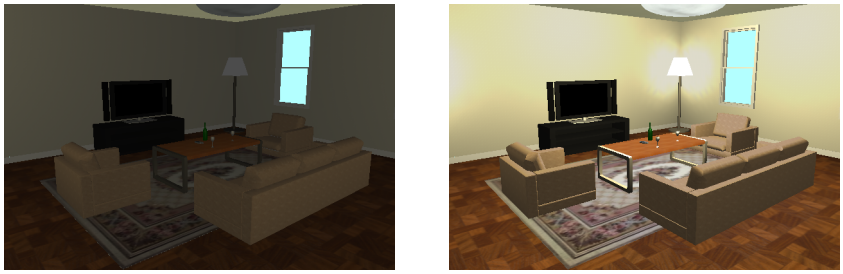
Systematic tester Systematic tester systematically generates many possible contexts and tests if given ubiquitous application operates as expected, in cooperation with simulators for physical quantities. Details are given in Section 7.

4 Visualizer and GUI for Designing Smartspace

UbiREAL has a GUI and a visualizer with which application users and/or test examiners (users, hereafter) can design Smartspace and observe how the Smartspace works depending on temporal variation of contexts. We call the software to realize this functionality simply GUI module, hereafter.

4.1 GUI for Designing Smartspace

GUI module allows users to place virtual devices in a 3D virtual space to compose Smartspace. Appearances of virtual devices can be designed using 3D modeling software on the market or can be substituted by 3D object data in VRML or other languages. 3D virtual space such as homes and rooms are constructed by



(a) (b)

Fig. 3. Visualization of Illumination of Lamps: (a) off, (b) on

(a) (b)

Fig. 4. Visualization of Heater's State: (a) off, (b) on

(a) (b)

Fig. 5. Visualization of TV's State: (a) off, (b) on

3-dimensionalizing a floor plan drawing using 3D modeling software. Users place various objects such as furniture (e.g., desk, chair, etc) and networked appliances (e.g., TV, stereo, air conditioner and sensor) in the 3D space. Devices are classified into static objects and movable objects. Static objects are selected from a pull-down menu, and placed by drag-and-drop with a mouse. Movable objects like a virtual inhabitant is placed by specifying a route in the 3D space as shown in Fig. 2. Routes can be either specified manually or automatically by the systematic test function in Section 7. Each route of a movable device can include actions like pushing button or manipulating remote control, at specified coordinates. Each movable object which represents an inhabitant is called *avatar*.

4.2 Visualization of Smartspace

When a state of a virtual device changes (e.g., lamp is turned on) or physical environment changes (e.g. room darkens after sunset), these changes should be observed by users through visually changing appearances of the virtual devices or virtual space as shown in Fig. 3. UbiREAL visualizer allows users to choose one view among 2D bird-eye view, first person view from the avatar, and so on. If there are more than one avatar, view can be switched among those avatars.

Now, we give a simple example of how Smartspace behavior is visualized using a scenario where a virtual inhabitant with an RFID gets close to an RFID reader, and consequently air conditioner is turned on.

When the inhabitant gets close to the RFID reader, RFID tag device and reader device communicate with each other via our network simulator in Section 5. Wireless communication is simulated based on positions of the RFID and RFID reader and obstacles on line-of-sight between them. The RFID reader communicates with a home server to turn on an air conditioner. Then, device driver on the air conditioner updates its state to change direction of its louver. GUI module obtains the updated state of the air conditioner and changes appearance of the air conditioner appropriately. The simulators for temperature and humidity also obtain the updated state, and change those physical quantities in the room gradually as time passes.

UbiREAL has a function to output results of simulation as a log file. So, users can observe simulation results repeatedly from the log file.

We show snapshots of how Smartspace is visualized in Figures 3, 4 and 5, when an avatar approaches lamp, heater and TV devices by following the route in Fig. 2.

5 Network Simulator

In this section, we describe details of the network simulation module of UbiREAL. As explained before, it is important to allow driver programs implemented for real devices to be executed in a virtual space, in order to keep compatibility between virtual and real ones. So, first, we explain the difference between UbiREAL network simulator and existing network simulators. Then, we describe the architecture of our network simulator.

5.1 Simulation using Driver Programs for Real Devices

Existing network simulators like ns-2 do not assume that programs for real devices are used in simulation. Instead, users of these existing network simulators have to write programs dedicated for virtual devices to be used in simulation. So, even when simulation was successful, programs must be re-written for real devices to execute the target application in a real world. In terms of development efficiency, it is desirable to reduce this labor. However, since many of existing network simulators require programs for devices to be written in event-driven manner,

and thus it is difficult to use the programs for real devices for simulation, which are usually written in flow-driven manner.

In UbiREAL network simulator, programs written in flow-driven manner can be used in simulation. As we mentioned in Section 4, some devices require modifications of device driver programs in order to send the latest state of the device to GUI module of UbiREAL. By changing compile-time options, binary programs for real devices and virtual devices can be built from the same source program.

5.2 Architecture

For programmers of device drivers, UbiREAL network simulator can be accessed via ordinary service primitives of TCP/IP protocol stack. Protocols like UPnP can be used via existing libraries which run on TCP/IP. As for MAC layer, our network simulator supports IEEE802.11a/b/g, ZigBee and Bluetooth besides ordinary Ethernet (10/100/1000MBps). As for network layer, it supports AODV and DSR to simulate wireless ad hoc networks. The user can change simulation granularity of each layer by selecting one of supported simulation models. Network configurations of each device such as IP address and ESS ID are given via configuration files for each device. Physical network structures like subnet of Ethernet are simulated by virtual hub devices.

Positions of movable devices change as time passes. Conditions of communication between devices change accordingly, and this can be reproduced by simulation of physical layer. Physical layer simulation keeps track of the positions of movable devices, and when each device transmits, e.g., an Ethernet frame, it decides which devices can receive the frame taking into account positions of devices and obstacles. It also adds error to frames according to condition of communication. Queries of positions for each device are implemented as ordinary method invocations. When a device enters wave range of another device, these devices can hear beacons from each other, and thus these devices will be able to communicate with each other. When a device receives a frame, the frame is handed to upper layer protocol.

5.3 Communication between Virtual Device and Real Device

UbiREAL network simulator allows virtual devices to communicate with real devices using TCP/IP protocol. With this feature, real devices can be used as the devices placed in the virtual Smartspace during tests. For example, when we need some of the virtual devices under test to behave especially correctly, real devices can be used for these virtual devices. This feature is also useful when we want to test some part of Smartspace through human perception.

By using operating systems like Linux, any form of Ethernet frames can be transmitted and received. Thus, it is possible to make a PC on which the simulator is running to be regarded as a router connected to the virtual network, and make real and virtual devices communicate with each other. Also, some network interface card can alter MAC addresses for every transmitted frame, and this makes virtual devices as if they are running on the same network as real devices.

6 Simulation of Physical Quantities

In order to know that a given ubiquitous application runs appropriately in a real world environment by simulation, we must be able to reproduce the temporal variation of physical quantities such as temperature and illumination in a virtual space considering effects of device and human behavior as well as characteristics of the target space.

Various simulators for real-time 3D environments have been researched and developed, and Ref.[15] surveys them. For example, Unreal Game Engine [16] uses a high performance physical engine and it calculates effects of light and sound. However, these existing simulators focus only on visible or audible physical phenomena. On the other hand, TATUS [6] simulates the effect of human behavior with respect to variation of packet loss ratio in a wireless LAN depending on how frequently human users cross the line-of-sight between devices. However, other invisible physical quantities are not simulated interactively with devices and human users.

UbiREAL simulates invisible physical quantities such as temperature, humidity, electricity and radio as well as visible (audible) quantities such as acoustic volume and illumination. In order to support any physical quantity, we adopted a publish-subscribe model for communication between virtual sensor devices and *physical quantity simulators*. Each physical quantity simulator is implemented based on appropriate formula in Physics for simulating a physical quantity. Each software driver of a sensor device concerning a physical quantity subscribes to the corresponding physical quantity simulator with parameter values necessary to update it. Each physical quantity simulator periodically calculates the latest value of the physical quantity and sends the value to subscribers if it has been changed. As necessary parameter values, each physical quantity simulator needs the previous quantity value, the characteristics such as size and capacity of the target room in the virtual space, device states, human behavior, other physical quantities and the time elapsed from the previous calculation.

For example, when we want to simulate temperature, the following formula in Physics is used as necessary parameters where C denotes heat capacity of the target room, and t, $\Delta T(t)$ and $Q(t)$ denote elapsed units of time, temperature difference and obtained heat quantity from previous evaluation, respectively.

$$\Delta T(t) = \frac{Q(t)}{C}$$

Each physical quantity simulator is implemented as a Plug-In program. Thus, it is easy to add new physical quantities and replace each Plug-In program by advanced one.

7 Systematic Testing

UbiREAL provides a function to systematically test the correctness of given application software in a given environment. For this purpose, we define a formal

model to represent the service specification (i.e., service requirement) of ubiquitous applications. We also define the correctness of the application with respect to the service specification. Based on the formal model, we propose a method for systematically test the correctness of the application.

7.1 Formal Model for Service Specification

A Smartspace U is defined as a tuple $U = (R, D)$, where $R = \{r_1, ..., r_n\}$ denotes the set of rooms in U and $D = \{d_1, ..., d_m\}$ denotes the set of devices.

Each room $r_i \in R$ has the following attributes: *pos* representing the position of r_i in U; *base* representing the shape and size of r_i's base; *cap* representing the capacity of r_i; and physical quantities of r_i such as temperature *temp*, humidity *humid*, heat capacity *heatcap*, illumination *illum* and acoustic volume *vol* in r_i. Each attribute is denoted like $r_i.temp$.

We assume that initial values of physical quantities at each room are given in advance. If doors or windows are open in some rooms, we suppose that the physical quantities calculated for those situations are given as initial attribute values.

Each device $d_j \in D$ has the following attributes: r representing the room where d_j is deployed; *pos* representing $d'_j s$ position in r, and *state* representing the state of d_j. Each attribute is denoted e.g., by $d_j.state$.

Each device d_j may have sensors which can obtain physical quantities around the device as sensed values. Device d_j may have actuators which execute actions to change physical quantities such as cooling, dehumidifying and lighting. As a result of action, $d_j.state$ and some physical quantities of room $d_j.r$ may change.

We assume that each attribute of a room or a device can have a discrete value in a predefined range. We define the global state of Smartspace U as a tuple of values for attributes of all rooms and devices in U.

In the proposed test method, we assume that the service specification *Spec* is given as a set of rules $AP = \{l_1, .., l_l\}$ and a set of requisite propositions $P = \{p_1, ..., p_l\}$. Here, each rule $l_i = (c_i, a_i)$ is a tuple consisting of a condition c_i and an action a_i, representing that a_i is executed when c_i holds. Here, as condition c_i, only linear inequalities with constants and sensor variables can be specified. As a_i, an action to a device can be specified.

We assume that there are no conflicting rules which simultaneously execute different actions to a device[2].

We can specify each proposition $p_k \in P$ with temporal logic such as CTL [17]. In this paper, we restrict each proposition to be represented by the following style of CTL.

$$AG(\phi_1 \Rightarrow AF(\phi_2))$$

Here, ϕ_1 and ϕ_2 represent propositions and are defined to be *true* or *false* for each state of U. $AG(\phi)$ means that proposition ϕ is *true* for every state reachable

[2] Such conflicting rules can automatically be detected when specifying rules with the technique in [2].

from the current state, and $AF(\phi)$ means that ϕ will eventually be *true* in a state reachable from the current state.

As a result, the above example proposition means that if the system is in a state which makes ϕ_1 *true*, the system will eventually transit to a state which makes ϕ_2 *true*, and that this always happens.

Using temporal logic like CTL, we can intuitively specify a proposition such that "if User1 is in room A, then the temperature and the humidity of room A will eventually be regulated around 23C and 60%, respectively".

Currently, the proposed test function is mainly targeting rule-based applications since we can easily obtain the specification from the application scenario. In order to support applications which are not rule-based, we may need a tool to facilitate derivation of the set of rules from the application scenario. It is beyond the scope of this paper.

7.2 Correctness of Application and Testing Method

Given Smartspace U with initial attribute values, the service specification $Spec = (AP, P)$ and the service implementation I under test, we say I is correct with respect to $Spec$ iff the following conditions hold.

(1) For every rule $l = (c, a) \in AP$ and every state s of U, if condition c holds for state s, then action a is executed.
(2) Every proposition $p \in P$ holds.

In order to test condition (1), we must generate all possible states of sensor values for each rule of AP, and input each state to I and observe whether the expected action is executed. However, all possible states of sensor values would be numerous even when we restrict sensor values to be discrete. So, in the proposed method, for a predefined number C, if each rule fires for C sample states which approximately cover all possible states, we regard that the rule is correctly implemented in I. By using a large number for C, we can improve reliability of tests, whereas the cost of tests increases. So, appropriate number must be decided as C by test examiner considering tradeoff.

For example, for a condition of a rule "if temperature and humidity are more than 28C and 70%", we can generate sample states $(temp, humid)$={(28,70), (28,80), (32,70), (32,80), (36,90)} when $C = 5$, where other attribute values in states are omitted.

In order to test whether implementation I satisfies each requisite proposition, we must examine how the action executed by some rule influences the physical quantities of the room as time passes. In the proposed method, we use simulators for physical quantities to know the physical quantities at each time after some action is executed. By obtaining the latest values of physical quantities from the simulators periodically, we can test whether each proposition holds or not.

When multiple devices are running at the same time in a room, physical quantities may change in a different way compared with the case when a single

device is running. For example, if an air conditioner is working at a specified power, how fast the temperature/humidity decreases/increases may be different among cases with or without other running devices such as TV and PC located in the same room. To cope with this problem, for a room with n devices, we test I for all possible combinations of those devices, that is, 2^n patterns (each device is working or not) as a total. If n is large, the patterns will be numerous. In that case, we can reduce the patterns by eliminating some patterns which unlikely happen.

7.3 Test Sequence Generation

In this section, we briefly explain how to generate test sequences using detailed examples. Suppose that the service specification $Spec = (AP, P)$ is given as follows.

$AP = \{(Exist(u_1, r_1) \wedge 28 \leq r_1.temp \wedge 70 \leq r_1.humid, Aircon_1.on(24, 60)),$
$(Exist(u_2, r_1) \wedge \neg Exist(u_1, r_1) \wedge 30 \leq r_1.temp \wedge 60 \leq r_1.humid, Aircon_1.on(28, 50)),$
$(Exist(u_1, r_2) \wedge Exist(u_2, r_2) \wedge Day = "Sun" \wedge Time = "6:30pm", TV_2.on(\text{"CNN"})),$
$("11:00pm" < Time, Lamp_1.off), ...\}$

$P = \{AG(Exist(u_1, r_1) \Rightarrow AF(23 \leq r_1.temp \leq 25)),$
$AG("11:00pm" < Time \Rightarrow AF(r_1.illum \leq 10))\}$

The first rule in AP specifies that air conditioner $Aircon_1$ should be turned on with 24C of temperature setting and 60 % of humidity setting, if user u_1 is in room r_1 and room temperature and humidity are more than 28C and 70 %, respectively. The second rule similarly specifies the behavior of air conditioner based on user u_2's preference, but if users u_1 and u_2 are in room r_2 at the same time, this rule are not executed (i.e., the first rule for u_1 is executed prior to u_2's rule). The third rule specifies the control of television TV_2 so that it will be turned on with channel "CNN" if two users u_1 and u_2 are in room r_2, the day is Sunday and the time is 6:30PM. The last rule specifies lamp $Lamp_1$ to be turned off if the time is over 11:00PM.

The first proposition in P specifies that the room temperature of r_1 will eventually be regulated between 23C and 25C if user u_1 is in room r_1. The second proposition specifies that the illumination of room r_1 will eventually be regulated less than 10 lux if the time is over 11:00PM.

The above rules in AP, our proposed test method generates the test sequence as shown in Table 1.

In Table 1, $\leftarrow$ means the value assignment, $Check(a)$ represents a function which returns *true* if action a has been executed, and parameters like $\%v11$ are assigned appropriate values which are generated every test sequence execution. During test sequence execution, if every $Check$ returns *true*, then we think that the test is successful, otherwise, the test fails.

For each proposition in P, our method generates the test sequence as shown in Table 2.

In Table 2, $CheckProp(\phi)$ represents a function which periodically checks if ϕ holds for the current state of U. It returns *true* if ϕ holds, and returns *false* if ϕ

Table 1. Test Sequence for Rules

(1) $u_1.pos \leftarrow r_1.pos + \%v11; r_1.temp \leftarrow \%v12; r_1.humid \leftarrow \%v13; Check(Aircon_1.on(24, 60))$
(2) $u_1.pos \leftarrow r_1.pos + \%v21; u_2.pos \leftarrow r_1.pos + \%v22; r_1.temp \leftarrow \%v23; r_1.humid \leftarrow \%v24; Check(Aircon_1.on(24, 60))$
(3) $u_2.pos \leftarrow r_1.pos + \%v31; u_2.pos \leftarrow r_3.pos; r_1.temp \leftarrow \%v32; r_1.humid \leftarrow \%v33; Check(Aircon_1.on(28, 50))$
(4) $u_1.pos \leftarrow r_1.pos + \%v41; u_2.pos \leftarrow r_1.pos + \%v42; Day \leftarrow$ "Sun"$; Time \leftarrow$ "$6 : 30pm$"$; Check(TV_2.on($"CNN"$))$
(5) $Time \leftarrow \%v51; Check(Lamp_1.off)$

Table 2. Test Sequence for Propositions

(1) $u_1.pos \leftarrow r_1.pos + \%v11; r_1.temp \leftarrow \%v12; r_1.humid \leftarrow \%v13; CheckProp(23 \leq r_1.temp \leq 25)$
(2) $Time \leftarrow \%v51; CheckProp(r_1.illum \leq 10)$

does not hold for a predefined time interval. Test sequence execution is conducted similarly to the test sequence for rules.

The test sequence is likely to be long. To shorten the length and reduce the time for testing, we can combine multiple test sequences and execute part of them simultaneously so that the common part is executed only once. Also if test examiner wants to execute test sequences with visual 3D animation in real-time, we can calculate the shortest route of a user to travel all devices in U. The positions of the calculated route are assigned to parameters such as $u_1.pos$ and $u_2.pos$ in the test sequence every execution of test sequence.

7.4 Inconsistency Detection

If some rules are inconsistent (e.g., in some contexts, cooling and heating devices run simultaneously at the same room), they should be detected. With UbiREAL simulator, basically users can detect such rules through graphical animation during simulation time.

There are other possibilities to realize inconsistency detection, although we do not go into detail. Some of them are shown below.
(1) Implement a tool to solve expressions consisting of logical product of conditions specified in rules which conflict with each other (e.g., simultaneously operate the same device) [2].
(2) Add a test sequence to detect, e.g., abnormal oscillation of some physical quantities.

8 Experiments

In this section, we describe the results of experiments to test performance of the proposed method. We used a PC with AMD Athlon 64×2 4200+ (CPU), ATI

Radeon X1300Pro (graphics card), 2GB of memory on Windows XP Pro SP2 and Java JSE5.0 in the experiments.

8.1 Performance of Test Sequence Execution to Validate Rules

First, we conducted the following experiment in order to check simulation speed when testing if action a is correctly executed in each rule $l = (c, a) \in AP$, as described in Section 7.

We placed in a virtual space a temperature sensor device, an air conditioner device and a home server device with a rule l which is "If the temperature is higher than 26 degrees Celsius, turn on the air conditioner". We measured the time t_1 when temperature sensor detects temperature change and the time t_2 when the execution of the action to turn on the air conditioner completes, in order to know the response time defined by $t = t2 - t1$.

In the experiment, we assumed that 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 emulates protocol layers above session layer, which includes the set of UPnP protocols. A home server device is developed based on CADEL framework [2] so that it evaluates condition c of rule l and sends a request to execute a to a target device if c holds.

By repeating the experiment more than 100 times, we found that it takes approximately 17.5ms to execute a test sequence for validating a rule. This suggests that supposing ordinary home environments, if there are 5 inhabitants, 100 devices, one rule is associated for each pair of a device and an inhabitant, and 100 tests are executed for each rule (i.e., $C = 100$), total time needed for the whole tests is about $5 \times 100 \times 100 \times 0.02s = 17$ minutes, and we believe that this would be practical enough. In reality, tests may take longer time since multiple devices are executed in parallel and conditions of all rules must be evaluated periodically in a PC executing simulation.

8.2 Performance of Simulation with Visualization

Next, we evaluated performance of the simulator when we use visualization function of UbiREAL. It is important to check if simulations can be performed in real time in order to allow users to intuitively observe how devices work as context changes through realtime 3D graphics. In the experiments, we let GUI module, network simulator module and a home server device developed based on CADEL framework run cooperatively, increasing the number of devices and rules. We measured framerate of 3D view when all rules are executed. We used 3 virtual rooms, and placed some objects in the rooms. The screen resolution of 3D view is 1600 $\times$ 1200 pixels, and the number of polygons drawn is 103,000 at maximum independently of the number of devices. We changed the number of devices from 0 to 25 and assigned two rules to each device, which means that the number of rules is changed between 0 and 50. The route of an avatar is set so that all

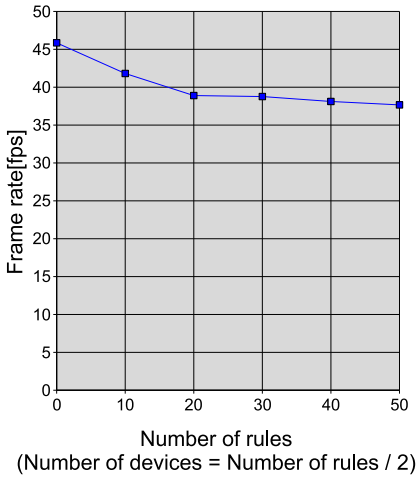


Fig. 6. Achieved Framerate vs. Number of Rules

rules are executed one by one. We measured framerate of visual simulation from avatar's view when the avatar moves following the route.

The experimental results are shown on Fig. 6. Fig. 6 shows that even when the numbers of devices and rules increase, there is only small influence to framerate, and thus realtime simulation is feasible even with large number of devices and rules.

9 Conclusion

In this paper, we proposed a ubiquitous application simulator *UbiREAL*. The UbiREAL simulator can simulate Smartspace so that users can intuitively grasp how devices are controlled depending on temporal variation of contexts in a virtual space and systematically test the correctness of the given application implementation for a given environment.

Main contribution and novelty of UbiREAL are that it incorporates simulators for physical quantities and network simulator with a visualization mechanism which cooperate to achieve systematic tests for ubiquitous applications using software for real devices. Through experiments, we showed that the tests for validating rules can be conducted in practical time for realistic ubiquitous applications, although the tests for requisite propositions depend on how accurately simulating temporal variation of physical quantities. Part of our future work includes evaluation of systematic test performance for requisite propositions with accurate physical quantity simulations.

References

1. G. Biegel, and V. Cahill : "A Framework for Developing Mobile, Context-aware Applications", *Proc. of 2nd IEEE Int'l Conf. on Pervasive Computing and Com-*

munications (PerCom2004), pp. 361–365, 2004.
2. K. Nishigaki, K. Yasumoto, N. Shibata, M. Ito, and T. Higashino : "Framework and Rule-based Language for Facilitating Context-aware Computing using Information Appliances", *Proc. of 1st Int'l Workshop on Services and Infrastructure for the Ubiquitous and Mobile Internet (SIUMI'05) (ICDCS'05 Workshop)*, pp. 345–351, 2005.
3. T. Yamazaki, H. Ueda, A. Sawada, Y. Tajika, and M. Minoh : "Networked Appliances Collaboration on the Ubiquitous Home", *Proc. of 3rd Int'l Conf. on Smart homes and health Telematic (ICOST 2005)*, Vol. 15, pp. 135–142, 2005.
4. N. Kawaguchi : "Cogma: A Middleware for Cooperative Smart Appliances for Ad hoc Environment", *Proc. of 1st Int'l Conf. on Mobile Computing and Ubiquitous Networking(ICMU2004)*, pp. 146–151, 2004.
5. J.J. Barton, and V. Vijayaraghavan : "UBIWISE, A Simulator for Ubiquitous Computing Systems Design", *Technical Report HPL-2003-93*, HP Laboratories, Palo Alto, 2003.
6. E. O'Neill, M. Klepal, D. Lewis, T. O'Donnell, D. O'Sullivan, and D. Pesch : "A Testbed for Evaluating Human Interaction with Ubiquitous Computing Environments", *Proc. of 1st Int'l Conf. on Testbeds and Research Infrastructures for the Development of NeTworks and Communities*, pp. 60–69, 2005.
7. E. Niemelä, and T. Vaskivuo : "Agile Middleware of Pervasive Computing Environments", *Proc. of 2nd IEEE Annual Conf. on Pervasive Computing and Communications Workshops (PerCom 2004 Workshop)*, pp. 192–197, 2004.
8. M. Roman, C.K. Hess, R. Cerqueira, R.H. Campbell, and K. Narhstedt : "Gaia: A Middleware Infrastructure to Enable Active spaces", *IEEE Pervasive Computing Magazine*, Vol. 1, pp. 74–83, 2002.
9. E. Chan, J. Bresler, J. Al-Muhtadi, and R. Campbell : "Gaia Microserver: An Extendable Mobile Middleware Platform", *Proc. of 3rd IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom2005)*, pp. 309–313, 2005.
10. A. Messer, A. Kunjithapatham, M. Sheshagiri, H. Song, P. Kumar, P. Nguyen, and K.H. Yi : "InterPlay: A Middleware for Seamless Device Integration and Task Orchestration in a Networked Home", *Proc. of 4th IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom2006)*, 2006.
11. S. Consolvo, L. Arnstein, and B.R. Franza : "User Study Techniques in the Design and Evaluation of a Ubicomp Environment", *Proc. of 4th Int'l Conf. on Ubiquitous Computing (UbiComp2002)*, pp. 73–90, 2002.
12. J. Nakata, and Y. Tan : "The Design and Implementation of Large Scale Ubiquitous Network Testbed", *Proc. of Workshop on Smart Object Systems (SObS05) (UbiComp 2005 Workshop)*, 2005.
13. K. Sanmugalingam, and G. Coulouris : "A Generic Location Event Simulator", *Proc. of 4th Int'l Conf. on Ubiquitous Computing (UbiComp2002)*, pp. 308–315 , 2002.
14. N. Roy, A. Roy, and S.K. Das : "Context-Aware Resource Management in Multi-Inhabitant Smart Homes: A Nash H-Learning based Approach", *Proc. of 4th IEEE Int'l Conf. on Pervasive Computing and Communications (PerCom2006)*, 2006.
15. J.L. Asbahr : "Beyond: A Portable Virtual World Simulation Framework", *Proc. of 7th Int'l Python Conf.*, 1998.
16. Unreal Engine: `http://www.unrealtechnology.com/`
17. E.M. Clarke, E.A. Emerson, and A.P. Sistla : "Automatic verification of finite state concurrent systems using temporal logic specifications", *ACM Trans. on Program Languages and Semantics*, Vol. 8, No. 2, pp. 244–263, 1986.