Widget-Based Simulator for Testing Smart Space*

Changgu Kang, Yoosoo Oh, and Woontack Woo

GIST U-VR Lab.
Gwangju 500-712, S. Korea
{ckang,yoh,wwoo}@gist.ac.kr

Abstract. In this paper, we propose a widget-based simulator for testing smart space by using virtual space. It is especially useful within this simulation field, because virtual space alleviates limited constraints such as time, space, technique, and cost. Researchers within this ubiquitous computing field use virtual space to simulate entities such as sensors, actuators, and services. Previous works focus more on systematic functions rather than simulation space configurations, and they consider only single service unit simulations. However, our proposed simulator focuses on the easy configuration of simulation space, which users want, and the simulation of entities such as detailed units (e.g., actuators, sensors, and context-awareness). Our simulator supports the set up of a variety of services by using dynamic links. For the design of our simulator, we consider widget-based virtual entities and plug-in techniques. We configure a virtual smart home in order to test our proposed simulation. We also evaluate the robustness of our proposed simulation. Finally, the proposed simulator is expected to provide a simulation environment in which simulation space and testing are made more effective for users.

Keywords: Simulation, virtual reality, smart space, context-awareness.

1 Introduction

A variety of services using virtual space, such as *World of Warcraft*, *Google Earth*, *Virtual Earth*, and *Second Life* has been recently released due primarily to an increase in the interest of virtual space. As a result of various constraints within real space, (e.g., cost, time, and space) virtual space is now viewed as more effective because of the space it provides for indirect experiences and useful information [1]. Specifically, since virtual space utilizes an environment which is not restricted by time and cost, virtual space is more useful for simulations environment. Since 3D virtual space creates realistic visuals, simulation space which uses the virtual space seems more realistic, and users receive realistic visual feedback from such a system. Because of these benefits, the ubiquitous computing research field has used virtual space to test smart services, or smart spaces [5,6,7,8].

* This research is supported by Korea Creative Content Agency (KOCCA), Ministry of culture, Sports and Tourism (MCST), under the Culture Technology (CT) Research & Development Program 2009.

Z. Pan et al. (Eds.): Transactions on Edutainment III, LNCS 5940, pp. 59–69, 2009.

The process by which a smart space provides vital services is as follows: First, sensor data is collected within a smart space. Secondly, sensor data is interpreted to generate context information. Thirdly, context information is applied to service actuators to provide services for users. Services provided in the smart space consists of sensors, context-awareness, and actuators and most researches work independently during this process to create these entities [2,3,4]. Therefore, when designing such simulators for the testing of smart space, consideration has to be given to simulation environments in respect to each entity (e.g., sensor, context-awareness, and actuator). Moreover, simulators have to be flexible in order to be applied within various simulation scenarios. To satisfy these requirements, simulators must be modified in order for simulation space to become detailed units. It must also be able easily to add new entities to simulation space for various simulation scenarios.

There are several previous works, CAST [5], UbiREAL [6], CASS [7], C@SA [8], etc., which have tested smart space by using virtual space. Most of these works deal with context-awareness for simulations because of its importance in the field of ubiquitous computing. Particularly, UbiREAL was designed for realistic simulation and systematic testing. Moreover, C@SA used intelligent agents for managing and simulating virtual services within virtual smart space. As we mentioned, smart service consists of actuators, sensors, and context-awareness. Therefore, simulators must be able to change each entity within simulation space. To adopt diverse scenarios, simulators must be able to provide new, easily-made virtual entities, and then be able to apply newly developed entities within simulation space. However, previous works usually exclude such these points.

To address these points, we propose a widget-based simulator to test smart space. First, we adopt a widget concept so that the simulator can provide flexible simulation space. Our proposed widget-based simulator provides independent implementation of widgets, and users can configure virtual services by using dynamic links between widgets. Second, we consider plug-ins as appropriate methods to register new widgets. Users can apply developed widgets to simulation space by moving new widgets to a fixed folder. Not only can users create a variety of virtual services by using shared widgets with our proposals, they can also more easily share widgets among themselves. Previous works depended on smart home scenarios within smart space in ubiquitous computing. Our proposed simulator is more general in design than previous works because our target domain not only includes smart homes but also diverse smart spaces.

The rest of this paper is organized as follows: Section 2 contains the design of our widget-based simulator. Section 3 contains the implementation of our simulator. Section 4 contains the evaluation of our proposed simulator. Section 5 contains the conclusion and a discussion of future works.

2 Architectural Design

2.1 Design Goals

In this paper, we consider the following two points as being vital for the design of our proposed simulator:

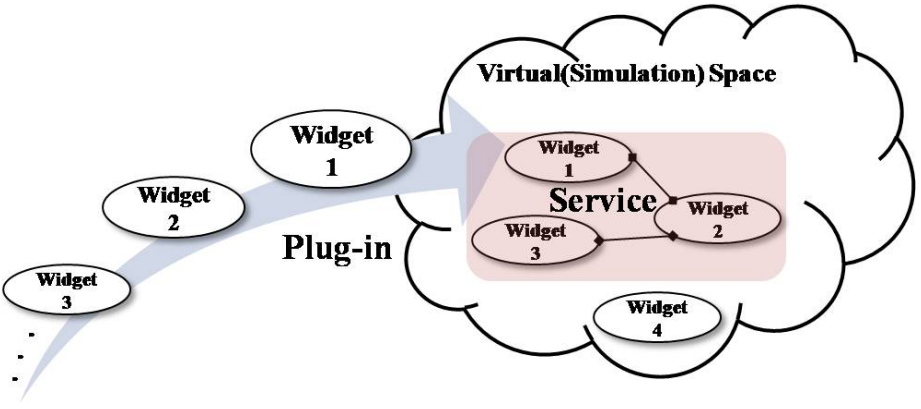


Fig. 1. Concept of widget-based Simulator

1)Widget-based virtual entities
2)Plug-in techniques to register new widgets

1) Widget-based virtual entities. The concept for our proposed simulator is based on the implementation of virtual entities in the form of widgets. This concept is considered in order that the simulation spaces are configured as smaller than service unit. Services should not be determined before simulation programs are enacted, but rather during operation. It is possible to configure services at real-time. Basically, connectivity between widgets is important because new services are generated through the connection of widgets. We define interface based on formal and primitive information (actuator-command) or on the data (sensor-raw data) of entities. Eventually, a context-aware widget must be implemented with input gates from sensor widgets, and output gates must be connected to actuator widgets. Because widgets are connected by context-aware widgets, dependency exists between context-aware widgets, and sensors or actuators. We discover similar points when considering the simulation of actuators and sensors in previous works. The simulation space of previous works includes all actuators and sensors, and only one context-aware module exists. By defining rules of operation for context-aware modules, services can be established. Although these simulators provide new simulation entity configurations for sensors and actuators, they do not support new context-awareness configurations. Also, because rules which control input interface in context-awareness are dependent on simulators and context-aware entities, it is difficult to independently configure context-aware entities within simulation space.

2) Plug-in techniques to register new widgets. Plug-in techniques should be used to register new widgets. When applied to simulators, this technique can assist users in the registration of widgets within simulators. Most previous works don't include precise methods which implement virtual services. Furthermore, there is no way for users to register new virtual services. However, when users share widgets by using our proposed simulator, they can easily register them without much difficulty. Also, our proposed simulator supports the registration of widgets during operation. Our two key points enable users dynamically to configure services by selecting sensors, actuators,

and context-awareness widgets which users want. Fig. 1 shows the concept behind our widget-based simulator.

2.2 Architecture

As shown in Fig. 2, we design the architecture for our widget-based simulator based on our two previously mentioned points. It includes the *Object Manager* to manage objects, the *Linking Manager* to manage links between widgets, and the *Input Rule Interface* to input user-defined rules. The *Object Manager* uses the *Object Tree* (oT-ree) which consists of *Category* nodes and *Object* nodes. The *Category* node doesn't have objective data, and are only used for the classification of objects. The *Object* node can utilize *Widget* or *3D Model* properties to visualize virtual space. The *Linking Manager* manages relations between widgets in order to configure services. When users establish a connection between two widgets, our proposed simulator generates *Link* nodes to the *Link Group* node. Fig. 3 shows the structures for the *Object Manager* and the *Linking Manager*.

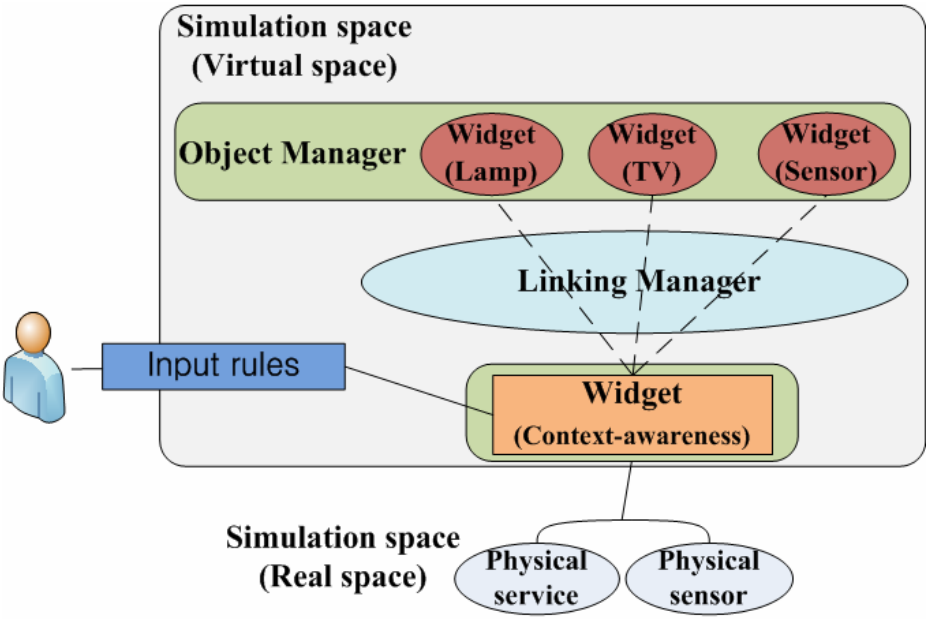


Fig. 2. Architecture of widget-based simulator

We consider two interfaces in order to use a variety of context-aware widgets. First interface is an interface to input user-defined rules. Second interface is an interface between context-awareness widgets and other widgets. Sensor widgets automatically send data to context-aware widgets, including a command by which input actuator widgets are activated. Commands for actuator widgets are deduced by user-defined rules and context information through the conditional matching of context-aware widgets.

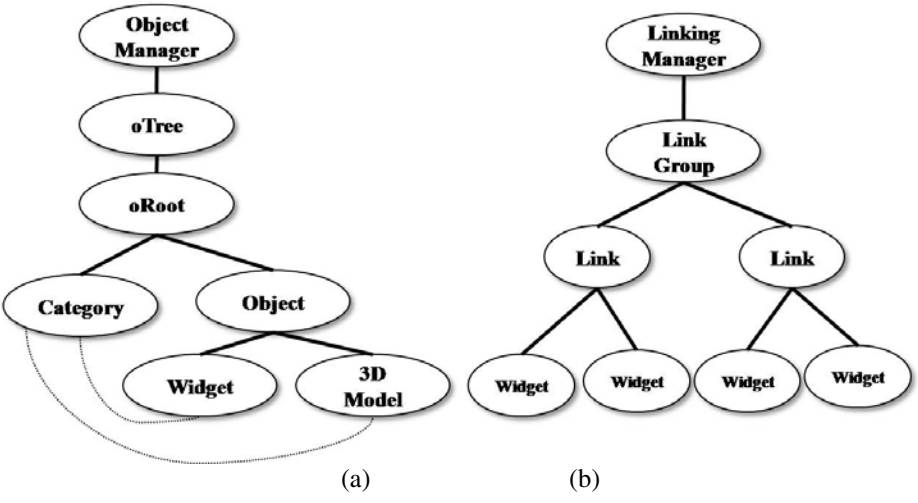


Fig. 3. The structure of (a) The *Object Manager* and (b) The *Linking Manager*

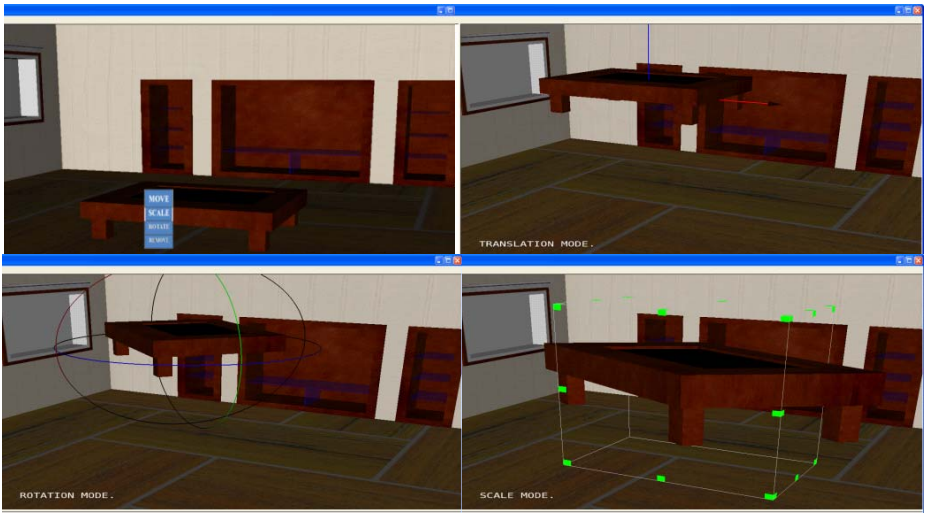
3 Implementation

We implement a widget-based simulator, actuator widgets and a context-aware widget. Actuator widgets are based on a lamp actuator and an air conditioner widget. We use OpenSceneGraph [11] for 3D visualization and 3D Studio MAX 9 [12] for 3D models. We develop our proposed simulator with the use of the visual studio 2005(MFC).

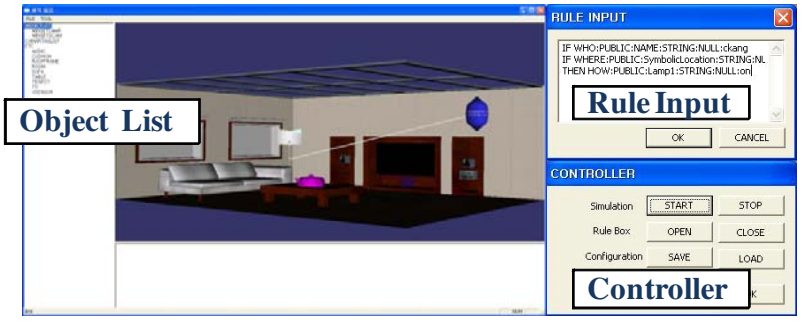
3.1 Widget-Based Simulator

Our simulator includes manipulation functions for configuring simulation space such as translation, rotation, and scale. After exporting our model, or the widgets which users want, users can then use the manipulation functions by clicking on exported widgets or models, and selecting manipulation menus. Fig. 4 shows object manipulation and configured simulation space with the use of the manipulation functions.

Also Fig. 4 (b) shows other interfaces used for simulation. Our simulator supports three interfaces in order to configure virtual (simulation) space: the **Object List**, **Rule Input**, and **Controller**. The **Object List** enumerates widgets and 3d models. Our simulator adds widgets and 3d models to the **Object List** when our simulator starts. Added objects appear in simulation space through the *Export Function* of the **Object List** interface. In addition, users can add objects to the **Object List** by using the *Add Function*. The **Rule Input** Interface saves inputted rules according to the context formulation for context-aware widgets. The **Controller** interface provides control buttons to save and load simulation space as well as some functions for simulation.



(a)



(b)

Fig. 4. (a) Manipulation functions (e.g., translation, rotation, and scale) and (b) configured simulation space and simulation interface

We implement the SimLib.lib to develop widgets. Users are able to handle events by some functions in the SimLib.lib. Table 1 shows the functions of the SimLib.lib. Additionally, widget developers can establish widget properties such as categories, names, and commands. As we mentioned, our proposed simulator manages widgets and 3D models by using the categories within the tree structure. The Object Manipulator exists to position each 3D model in our simulator. Users can change position, rotation, and scale of 3D models to configure simulation space. Also, users can save and read simulation configuration files in the form of XML. Fig 5 shows the XML format.

Table 1. Functions of SimLib.lib

Type	Function	Description
Required	SetNameObj	Set the name of a widget
	SetCategory	Set the category of a widget
	CreateModel	Register a 3D model for a widget
Optional	DoActionHandle	Set a behavior according to itself commands
	KeyEventHandle	Set a behavior for key input
	MouseEventHandle	Set a behavior for mouse input
	RefreshModel	Change a 3D model

```
<?xml version="1.0" encoding="utf-8" ?>
- <VirtualSpace>
  - <VEntities>
    - <AUDIO Trans="0.00000 25.73100 0.00000" Rotate="0.00000 0.00000 0.0
        <Category>MODELLIST</Category>
      </AUDIO>
    - <CUSHION Trans="0.00000 0.00000 0.00000" Rotate="0.00000 0.00000 0.
        <Category>MODELLIST</Category>
      </CUSHION>
    - <ROOFFRAME Trans="0.00000 3.66970 25.91607" Rotate="0.00000 0.000(
        NodeMask="2147483647">
        <Category>MODELLIST</Category>
      </ROOFFRAME>
    - <ROOM Trans="0.00000 0.00000 0.00000" Rotate="0.00000 0.00000 0.00
        <Category>MODELLIST</Category>
```

Fig. 5. The XML format saving and loading simulation space

3.2 Widgets

We develop a context-aware widget, as well as lamp and air conditioner widgets for simulation. The context-aware widget is based on the Unified Context-aware Application Model (UCAM) [3]. Existing UCAM consists of a service and a sensor. We modify that it becomes one unified UCAM. Likewise, we develop context-aware widgets by using modified UCAMs. We implement rule parsing and condition matching modules to modify UCAMs in order to apply user-defined rules from the Rule Input Interface to express UCAMs. Fig. 6 shows the formula for the context-aware widget.

```
IF WHO:PUBLIC:NAME:STRING:NULL:CKANG [CONDITION]
IF WHERE:PUBLIC:SymbolicLocation:STRING:NULL:ubiHome [CONDITION]
THEN HOW:PUBLIC:Lamp:STRING:NULL:BLUE [ACTION]
```

Fig. 6. Rule formula of context-aware widget

Fig. 7. 3D models of implemented widgets

Lamp and air conditioner widgets are developed from the form of actuator widgets. Each widget possesses the 3D model. Fig. 7 shows the 3D model for the implementation of widgets. The lamp widget has command properties by the following colors: WHITE, RED, GREEN, and BLUE. The air conditioner widget has ON, and OFF.

4 Simulation and Evaluation

Fig. 8 shows a simulation model using implemented widgets. Users can simultaneously use physical sensors and virtual services for simulation within virtual and real space by using implemented context-aware widget. We configure virtual smart homes based on the simulation model in Fig. 8 and simulate lamp services by using sensor data inputted from particle sensors [13] such as those found in physical sensors. Fig. 9 shows the simulation of lamp and air conditioner services. We define the rules in Fig. 10. Virtual smart homes are based on the ubiHome [9] as shown in Fig. 11.

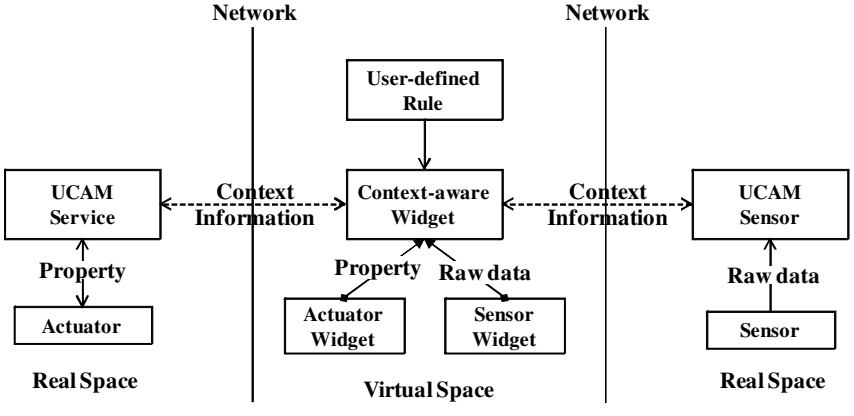


Fig. 8. Simulation model using implemented widgets

Fig. 9. Simulation of lamp and air conditioner

```
IF HOW:PUBLIC:LightLevel:STRING:NULL:LOW [CONDITION]
THEN HOW:PUBLIC:Lamp:STRING:NULL:RED [RESULT]
IF HOW:PUBLIC:TemperatureLevel:STRING:NULL:HIGH [CONDITION]
THEN HOW:PUBLIC:Aircon:STRING:NULL:ON [RESULT]
```

Fig. 10. User-defined rules for simulation

Fig. 11. Real ubiHome and virtual ubiHome

We measure the robustness of our application within evaluation areas. Measured items include system error rates, adaptation time, and simulation time. The setup for the measurement of the system error rate is the same as in Fig 10. We use the following conditions to conduct our experiment: Intel® Core™2 Duo 1.66GHz, 2GB RAM, and nVidia GeForce 8400. We measure the resulting error rate by comparing user-defined rules with data inputted from outside UCAM sensors. The adaptation time equals the duration from the time when user change rules to the time when changed

rules are applied to simulation environment. Finally, we measure simulation time according to the number of services. The purpose of simulation time also includes whether real time simulation is possible or not. When users add services to simulation space at real time, we measure the time it takes all services to become active. For this measurement, we assume that the data which are for all services to be active always input from a sensor.

As the result of our first experiment, user-defined rules and context information from UCAM sensors were successfully matched without error by using condition matching. The duration for adaptation time averages 8.2 ms. The frame rate for 3D rendering is about 60fps within our experiment environment. The result shows that our modified rules can be applied to simulation environments in less time than the time it takes for 1 frame. The result shows that the number of services does not influence simulation time. In conclusion, even if users add services to simulation space, most simulation time almost does not increase. Table 2 shows the experimental results for adaptation time and simulation time.

Table 2. Experimental results for adaptation time and simulation time

	Adaptation Time	Simulation time according to the number of services					
		1	2	3	4	5	6
Time(ms)	8.2	9.8	8.1	10.5	7.3	9.9	8.2

5 Conclusion

In this paper, we proposed a widget-based simulator for the testing of smart space. Also, we focused our attention on two points in the design of our proposed simulator. Firstly, we focused our research on implementation of widget-based virtual entities. Secondly, we focused on plug in techniques. Users can not only independently implement widget-based virtual sensors, context-awareness, and actuators, but can also configure services with the use of dynamic linking. Users can also configure a variety of services by using and sharing other widgets which other users have developed.

We measured error rates, adaptation time and simulation time according to the quantity of services in order to reveal the application robustness for our proposed simulator. The results show that there are no errors involved in rule matching, and furthermore, it is possible to change and simulate services at real time. In our future, we are considering the simultaneous connection and simulation of services within augment reality (AR), virtual reality (VR), and real space. We have referred to it as the U-VR environment [10]. By implementing our application, users can not only configure U-VR services, but also simulate their own services.

References

1. Kock, N.: E-Collaboration and E-Commerce in Virtual Worlds: The Potential of Second Life and World of Warcraft. International Journal of e-Collaboration 4(3), 01–13 (2008)
2. Choi, A., Oh, Y., Park, G., Woo, W.: Stone type Physiological Sensing Device for Daily Monitoring in an Ambient Intelligence Environment. In: Aarts, E., Crowley, J.L., de Ruyter, B., Gerhäuser, H., Pflaum, A., Schmidt, J., Wichert, R. (eds.) AmI 2008. LNCS, vol. 5355, pp. 343–359. Springer, Heidelberg (2008)
3. Oh, Y., Woo, W.: How to build a Context-aware Architecture for Ubiquitous VR. In: IEEE ISUVR, CEUR-WS, pp. 032–033 (2007)
4. Shin, C., Woo, W.: Socially aware TV Program Recommender for Multiple Viewers. IEEE Transaction on Consumer Electronics 55(2), 927–932 (2009)
5. Kim, I., Park, H., Lee, Y., Lee, S., Lee, H., Noh, B.: Design and Implementation of Context-Awareness Simulation Toolkit for Context learning. In: IEEE SUTC, pp. 96–103 (2006)
6. Nishikawa, H., Yamamoto, S., Tamai, M., Nishigaki, K., Kitani, T., Shibata, N., Yasumoto, K., Ito, M.: UbiREAL: Realistic Smartspace Simulator for Systematic Testing. In: Dourish, P., Friday, A. (eds.) UbiComp 2006. LNCS, vol. 4206, pp. 322–331. Springer, Heidelberg (2006)
7. Park, J., Moon, M., Hwang, S., Yeon, K.: CASS: A Context-Aware Simulation System for Smart Home. In: 5th International Conference on Software Engineering Research Management and Applications, pp. 461–467. IEEE CS, Los Alamitos (2007)
8. De Carolis, B., Cozzolongo, G., Pizzutilo, S., Plantamura, V.L.: Agent-based home simulation and control. In: Hacid, M.-S., Murray, N.V., Raś, Z.W., Tsumoto, S. (eds.) ISMIS 2005. LNCS (LNAI), vol. 3488, pp. 404–412. Springer, Heidelberg (2005)
9. Jang, S., Shin, C., Oh, Y., Woo, W.: Introduction of “UbiHome" Testbed. IPSJ SIG Technical Reports (60), pp. 215–218 (2005)
10. Lee, Y., Oh, S., Shin, C., Woo, W.: Recent Trends in Ubiquitous Virtual Reality. In: International Symposium on Ubiquitous Virtual Reality, pp. 33–36 (2008)
11. OpenSceneGraph, `http://www.openscenegraph.org/`
12. Autodesk 3D StudioMax9, `http://usa.autodesk.com/adsk`
13. Particle (TECO), `http://particle.Teco.edu/`