Hybrid prototyping by using virtual and miniature simulation for designing spatial interactive information systems

Yasuto Nakanishi, Koji Sekiguchi, Takuro Ohmori,
Soh kitahara, and Daisuke Akatsuka

Keio University, Faculty of Environment and Information Studies,
5322 Endo, Fujisawa, Kanagawa, Japan
{naka,t07412ks,ohmori,soh335,dadaa}@sfc.keio.ac.jp
http://cc.unitedfield.net/

Abstract. In this paper, we introduce CityCompiler, an integrated environment for the iteration-based development of spatial interactive systems. CityCompiler visualizes interactive systems in a virtual 3D space by combining the Processing[1] source code and the 3D model of the real space, designed with Google SketchUp. A simulation in virtual space enables us to test a spatial layout and a combination of components. In addition, the system function of smoothly switching between a virtual sensor and a real sensor realizes hybrid prototyping by means of virtual simulation and miniature simulation. These features integrate the space design with the software design and allow the smooth deployment of spatial interactive information systems into the real world.

Keywords: software design, space design, prototyping, deployment, IDE

1 Introduction

In recent years, the use of large displays in public or commercial spaces has become increasingly popular.These displays are attractive and eye-catching, and they bring with them embodied and spatial interactions. Some notable examples of emerging applications of visual displays are sharing large-size visualized data, smart office applications, digital signage, interactive public art, and interior and exterior architectural displays. These spatial interactive systems consist of both software and real-space components. For instance, interactive digital signage or applications for urban environments mostly use software, cameras, sensors, projectors, speakers, PCs, and real spaces, such as exhibition rooms and urban buildings for projection. In order to make these systems work in the right or effective manner, developers are required to simultaneously configure software and real-space components. For example, configuring the camera/projection location, size, direction, or designing the real space is a significant process.

[1] http://www.processing.com/

Various input and output devices are available for constructing such interactive environments, spanning a large design space. The choice of devices determines modalities, which affect both the nature and the impact of the interactions of such environments with a system. They are crucial for the usability and acceptance of the system by its users, and the integration of interaction design with space design is a topic of considerable interest.

However, in general, after the input and output devices have been decided and the software for the interactive system has been developed, the problem of where to physically place these devices arises. In particular, it should be noted that optical devices such as cameras or displays have a limited field of view, which needs to be considered. Presently, it is difficult for software developers to configure such system components before the system is deployed and runs in the real space. Some developers use low-fidelity techniques, such as paper prototypes and mental walkthroughs, and others have to wait for a full-scale deployment. In other words, software developers are unable to properly test the entire system in the early stages of system development.

To solve these problems, we propose CityCompiler, which enables spatial interactive system developers to create their systems using an iteration-based development process, using iterative visualization or trial-and-error. CityCompiler simulates how an interactive system developed with Processing runs in a 3D virtual world, modeled using Google SketchUp. Simulation in the virtual 3D world allows not only collaboration between the software developer and the space designer, but also trial-and-error testing when sketching a spatial interactive system by choosing input and output devices. The objective of CityCompiler is to support the interaction design concerning the choice of sensors and actuators and their placement in the 3D world, including the assessment of their range of sensitivity and effect such as the visibility of a display.

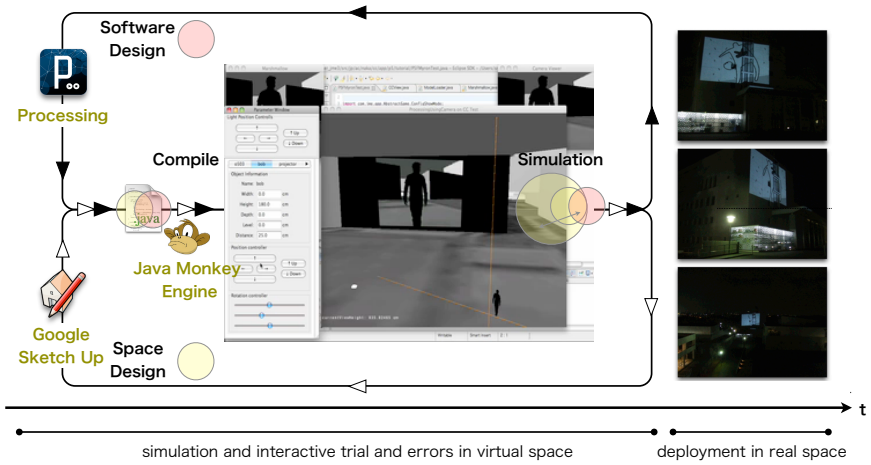

Fig. 1. Overview of CityCompiler.

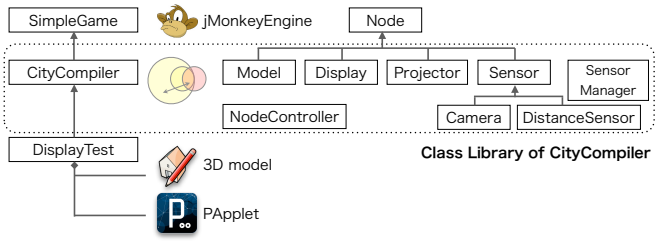

Fig. 2. Class Diagram for CityCompiler.

2 Hybrid prototyping with CityCompiler

2.1 Sketching and prototyping

CityCompiler is composed of two modules: a Java class library and a 3D viewer. The 3D viewer is based on the jMonkeyEngine, which is a Java-based 3D game engine[2]. These modules are compressed in jar format, which allows developers to use the prototype with an integrated development environment (IDE), such as Eclipse or NetBeans. Our Java class library provides several Java classes and interfaces that support the development of spatial interactive systems, and some spatial components that work in the virtual spaces are also implemented. Figure 2 shows a part of the CityCompiler class diagram.

- The Model class loads the 3D models saved in the .obj file format created with 3D modeling tools.
- The Projector class projects the Processing application onto the 3D models.
- The Display class shows the application on the surface.
- The Camera class captures images in the virtual world and transfers them to any other Java application.
- The DistanceSensor class returns distances to other objects on the basis of collision detection in the virtual world.

The developer creates a subclass of the CityCompiler class in order to integrate 3D models, the Processing application, and input/output devices in a virtual world. These three main elements that the developer creates are as follows.

(a) Source code as a subclass of the PApplet class for a Processing application
(b) A 3D model for the surrounding environment
(c) Source code based on CityCompiler for integration of the system components

To be more specific, (a) would be installed in (b), and (c) would be used to visualize (a), which runs in (b) and supports several trial-and-error tests with regards to (a). For (c), the developer defines a class that is a subclass of the CityCompiler class, and has several instances of the Display class or Camera class together with the general Java class library.

[2] http://www.jmonkeyengine.org/

2.2 Testing layout and combination of components

Once the source code for (a) and the model for (b) are prepared, the developer then compiles (c). After compiling the source code (c), the 3D viewer is displayed along with the parameter window, the camera viewer, and the user application's window. The parameter window allows the developer to manipulate several parameters of the 3D components, such as the position and rotation during runtime. For example, the developer can use the parameter window to configure and check the elevation of the virtual cameras with realistic heights, such as 180 cm or 120 cm. This allows the developer to simultaneously compare the layout of actuators from multiple viewpoints, such as their display or projector.

After carrying out the above-mentioned processes, the developer can return to the process of coding the source (a) or (c), or both, or designing the 3D models (b), which happens in the case of iteration.

2.3 Switching virtual devices and real devices

In most cases, the space required for developing and testing these types of systems and the space required for deploying them are different. It is impossible to reliably prepare specific visual, spatial, and sensory contexts for the final deployment. Thus, when deploying a system into a real space, unexpected situations often arise, and parameters of the software logic must be adjusted in order to handle inputs from sensors. In some cases, the logic itself is even revised on the site, although such cases should be avoided as much as possible; the visualized simulation in our CityCompiler will make the software logic robust and flexible. In particular, we suppose that operating a system within both a virtual space and miniature space and comparing the corresponding source code will help developers to handle the parameters and revise codes. Obtaining different inputs from a number of sources will allow for further development. In addition, comparing operations in a virtual space and miniature space will aid the successful installation of a system in real space.

In CityCompiler, a SensorManager class is implemented for managing input devices that allow for smooth switching operations in virtual, miniature, and the real space. This class tries to gain access to a real sensor connected to the PC and selects a real sensor or a virtual sensor as the data source. For example, when a USB camera is detected, real-world images are captured and are sent to the application, otherwise, virtual-world images are captured and are sent by the system. Our current implementation can use both USB cameras and Phidgets[3] for USB sensing and control, and these devices are available as real sensors and virtual sensors. Switching virtual sensors with real sensors, and adding the number of various virtual sensors also supports the design decision of choosing between available sensors and actuators.

Figure 3 shows a system deployed into the real space after carrying out repeated simulations both in the virtual space and in the miniature space. The

[3] http://www.phidgets.com/

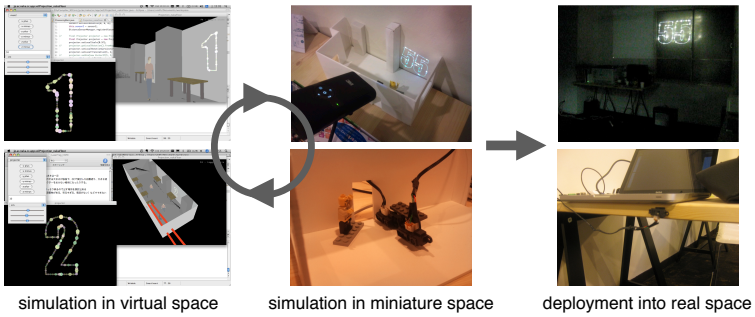

Fig. 3. Deployment into real space after carrying out repeated simulations both in the virtual space and in the miniature space.

system comprises two distance measuring sensor units connected to a Phidgets board and a projector. It counts the number of visitors at the entrance of a room, and displays the results with motion graphics. First, we developed a system that works in the virtual space and arranged virtual sensors and a virtual projector in a model of our room. Next, we ran it in a miniature space made of white styrofoam with real sensors and a mini-projector. In the virtual space, we can arrange any numbers of projectors in any location, and it is easy to arrange them horizontally. However, in the miniature space, such an ideal placement is difficult. Therefore, in the Processing application, we added a function to change the size, angle, and location of the counted number. We also ran the distance sensors in the miniature space by using a finger to represent one person and moving it around. However this did not work as well as in the virtual space, because the values of parameters for sensing people were different. Thus, a function was added to change the thresholds of the sensing logic during runtime.

Figure 4 shows a system that displays several photos those include location information on a map. We planned to exhibit it with a screen set to the window of our room, which faces the road. First, we developed the processing application, and arranged a virtual projector in the virtual model of our room. Next, we ran it in a miniature space with a mini-projector. The miniature model was made of white styrofoam, and we used a small piece of cloth as a miniature screen. Our current system does not calculate the size of the styrofoam and the cloth automatically, so we calculated the sizes of the parts based on the 3D model and assembled them. We checked how the system worked by watching a movie sent from a wireless camera on the head of a doll, and matched the scale of the miniature to the height of the doll. We found that the size of displayed photos was too small, and so we added a function that changes the sizes of photos for the whole application during runtime. Next, we noticed that the brightness of both the room and the road were the same in the miniature space, although the room was bright and the road was dark at night. Therefore, we added a white LED in the miniature room to introduce such a brightness gap. We also found that the brightness difference between the room and the road changed the

appearance of the photos. Thus, we added a function to adjust the brightness of the whole application.

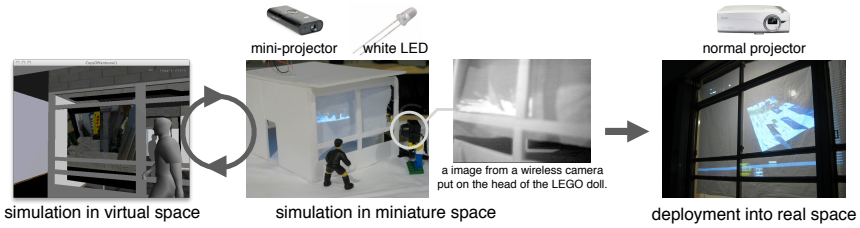

Fig. 4. Another case of hybrid prototyping.

In these two cases, we obtained the required parameters and the revised points of codes by carrying out the simulation in the miniature space. Such improvements make the system more robust and flexible, and will ensure that the system is smoothly deployed in the real world.

We call this hybrid prototyping, and it is a process that can be used to integrate a space design with a software design. In order to deploy devices that match the simulation, we have to implement virtual devices that have the same specifications as the real devices. However, it is difficult to make devices the same as they are in the miniature space. In the above-mentioned cases, the differences between the virtual simulation and the miniature simulation helped us to discover parameters and brought about spiral design cycles. Running the code in several different spaces, and with different devices helps to evolve both the software and the spatial arrangement of devices, which is a benefit of hybrid prototyping. From this viewpoint, it might not necessarily be good that our system helps to automatically form an exact miniature model with a 3D printer.

There are limitations in hybrid prototyping. For example, it cannot tell all the necessary parameters, even if we repeat virtual and miniature simulations, and we might find further problems in the final deployment. However, updating the source code using hybrid prototyping would allow the flexibility to handle new problems. A further limitation is that it is time-consuming, expensive, and complicated to arrange several displays or cameras in a miniature space. In this case, it is more realistic to use a virtual simulation and a partial miniature simulation concurrently.

3 Related works

There have been many prototyping tools for various domains. Topiary is a tool for prototyping location-enhanced applications, and enables iteration on designs by using a map that demonstrates scenarios composed of interaction sequences[8]. Papier-Mâché is a toolkit for building tangible user interfaces using computer

vision, etc., and introduces a high level event model[5]. Both these systems and our system aim to make it easy to prototype, evaluate and iterate on augmented environments. Incorporating the functions of Topiary and Papier-Mâché will makes our system a better tool. Singh et al. proposed the use of immersive video as a means of rapid prototyping and an evaluation tool for mobile and ambient applications[6]. This approach uses immersive video with surround sound as a simulated infrastructure to create a realistic simulation of a ubiquitous environment for software design. The objective of CityCompiler is also to simulate for a ubiquitous computing environment. However, it aims at having both virtual and miniature simulations rather than creating just one realistic simulation. UbiREAL[3] and eHomeSimulater[1] are simulators used for developing and testing devices that run on a smart home, and their main aim is developing hardware or software. CityCompiler also simulates a smart environment, and supports not only indoor smart space environments, but also outdoor systems, such as urban areas. The objective of CityCompiler is to integrate software design and spatial design with iteration-based development. CityCompiler places great emphasis on iteration-based development, rather than accurate simulation. This is the reason why we selected Processing and SketchUp.

Firefly is a set of software tools that bridge the gap between 3D modeling software, micro-controllers and the internet[4]. It also allows near real-time data flow between virtual and model spaces, and will read/write data to/from internet feeds or sensors. However, it changes shapes made by parametric and algorithmic design, and does not target sound, video, graphics, or animation, which Processing and CityCompiler do. Nakanishi et al. proposed two multiagent-based participatory simulation methods for a large-scale socially embedded system[4]. One involves participatory simulation, in which scenario-guided agents and human-controlled avatars coexist in a shared virtual space and jointly perform simulations. The other involves an augmented experiment, in which an experiment is performed in a real space by human subjects, enhanced by a large scale multi-agent simulation. They can be Cross-Reality[2] or Augmented-Virtuality to combine a virtual space and a model space and to combine people with multi-agents. Hybrid prototyping as Cross-Reality or Augmented-Virtuality would be a promising way to design and develop spatial information systems or responsive augmented environments.

Yamashita et al. demonstrated that seating arrangements exert an important influence on video-mediated conversations; different seating arrangements yield differences in speech patterns, senses of unity, and quality of solutions[7]. The display layout allowed the participants to change their body orientations, head movements, and seating arrangements, creating different patterns of video-mediated conversations. This means that both the software design and the space design, along with the orientation and disposition of input/output devices influences the way people interact. The integration of interaction design with space design, and in particular, spatiality, is a topic of considerable interest.

[4] http://www.fireflyexperiments.com/

4 Conclusion and Future work

One problem in developing pervasive computing applications is the simulation of the required input/output devices in the environment in which they are to be deployed. In this paper, we introduced CityCompiler, an integrated environment for the iteration-based development of spatial interactive systems. CityCompiler visualizes an interactive system in a virtual 3D space by combining the Processing source code and a 3D model designed with SketchUp. In the Web interface design, graphic design is integrated with software design. In the case of the spatial information systems design, space design should be integrated with software design in the same way, and simulations using CityCompiler are studied to realize the integration. CityCompiler enables the developer to carry out interactive trial-and-error tests with the testing layout and a combination of components. Here, the developer uses both the virtual space and the miniature space before the final deployment into the real space. The simulation allows the software designer, space designer, or interaction designer to browse the intended activities, and to collaboratively highlight their context in the urban environment by considering spatial regions and the installation of input and output devices. In this paper, we introduced only two of our processes employing hybrid prototyping. In future, we will investigate the effectiveness of hybrid prototyping by analyzing more cases. We believe hybrid prototyping would be effective not only for deploying a system into the real world but also for designing a system with a new concept. We will also use both virtual and miniature simulations to come up with an idea and investigate the advantages and disadvantages of each method.

Acknowledgments. This research is supported by the JST PRESTO program.

References

1. Armac, I. and Retkowitz, D. Simulation of Smart Environments. Proceedings of the IEEE International Conference on Pervasive Services 2007, pp. 257-266.
2. Lifton, J. , et. al. Metaphor and Manifestation - Cross Reality with Ubiquitous Sensor/Actuator Networks. IEEE Pervasive Computing, vol. 8, no. 3, pp. 24-33.
3. Nishikawa, H., et. al. UbiREAL: Realistic Smartspace Simulator for Systematic Testing. Proceedings of UbiComp2006, pp.459-476.
4. Nakanishi, H., Ishida, T and Koizumi, S. Virtual Cities for Simulating Smart Urban Public Spaces. Handbook of Research on Urban Informatics: The Practice and Promise of the Real-Time City, IGI Global, pp.256-268.
5. Scott R. , et. al. Papier-Mch: Toolkit Support for Tangible Input. CHI Letters, Human Factors in Computing Systems: CHI2004. 6(1).
6. Singh, P., et. al. Immersive video as a rapid prototyping and evaluation tool for mobile and ambient applications. Proceedings of Mobile HCI '06, pp.264-264.
7. Yamashita, Y., et.al. Impact of Seating Positions on Group Video Communication. Proceedings of CSCW'08, pp. 177-186.
8. Yang Li, et. al. Topiary: a tool for prototyping location-enhanced applications. Proceedings of the 17th annual ACM symposium on User interface software and technology (UIST '04), pp. 217-226.