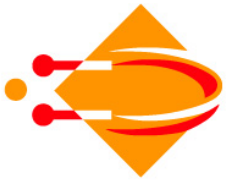
AN ENVIRONMENT SIMULATOR FOR MOBILE CONTEXT-AWARE SYSTEM DESIGN

KÁROLY SZÁNTÓ



IT University
of Copenhagen

Thesis Subtitle

June 2014 – version 0.1

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

1 INTRODUCTION

Before the early 1970s, computers were owned by large institutions like corporations, universities or government agencies. These machines, also know as *Mainframes*, were costly and space consuming, most institutions owning only one piece of such equipment. Lacking any kind of user interface, Mainframes would accept jobs in the form of punched cards, an early input mechanism for data and programs into computers. A group of people in the institutions were instructed how to use the machines (creating punched cards, operating the machines) and they represented the interface between everyday users and the Mainframe. Quite a bottleneck!

In 1971 the first commercial microprocessor was released, representing the rise of a new era: the Personal Computer (PC). The PC represented a huge step forward, offering in perspective the possibility for anyone to own a computer and to operate it using novel input/output mechanism: graphical user interface, mouse, keyboard and other peripherals. This is the standard PC setup that we know and, after such o along time, still use. By this I want to point that although we have came a long way, by hardware and software advancements, the human computer interaction (HCI) form that is mostly used to interact with PCs is just as it was envisioned a little more than 40 years ago.

The PCs revolution was followed by the idea of portable personal devices like laptops, dynabooks (an educational device similar to a nowadays tablet). Although novel at that time, all the fever generated by this revolution was viewed by a group of people at PARC[1] as being ephemeral: "[...] My colleagues and I at PARC think that the idea of a "personal computer" itself is misplaced, and that the vision of laptop machines, dynabooks and "knowledge navigators" is only a transitional step toward achieving the real potential of information technology. Such machines cannot truly make computing an integral, invisible part of the way people live their lives. Therefore we are trying to conceive a new way of thinking about computers in the world, one that takes into account the natural human environment and allows the computers themselves to vanish into the background". [10]. This futuristic concept envisioned at PARC which further shaped the

1 Palo Alto Research Center

way we see and use technology and devices is known as *ubicomp*[2].

The novel vision in ubicomp can be viewed as a second major advancement in the world of technology. The first major advancement, the transition from Mainframe to PC, took the relation between people and computer from a many-to-one to a one-to-one relationship. Ubicomp envisioned a transition from one-to-one to many-to-one, where each user will own not one but many device. These device would not be only PCs, they would be all sorts of portable device, some specialized at accomplishing specific tasks and performing under various circumstances. Classical HCI was not fit for this new setup. More advanced and more intuitive concepts were needed to make the best of these technological advances. One such emerging concept was *context-aware computing*. Context is "any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object. We define context-awareness or context-aware computing as the use of context to provide task-relevant information and/or services to a user. Three important context-awareness behaviours are the presentation of information and services to a user, automatic execution of a service, and tagging of context to information for later retrieval". [1]. Context-awareness takes HCI to a whole different level. We are not thinking anymore of only sitting down in front of a computer and interacting with it using a few peripherals. Users and developers have a whole new spectrum they can explore, in dynamic and continuously evolving technological environment. Take for example a piece of software that gives information on historical monuments. If this software is running on a PC, the user would manually search for a specific monument to retrieve the information. The same software running on a mobile device with localization capabilities (i.e. GPS), could take away all the explicit interaction by determining the user's location and if this is in near proximity of a certain historical monument, the displaying the relevant information.

While designing mobile pervasive computing environments, software developers are bound to work with cutting edge hardware and software technologies and to adopt new communication models for HCI. Throughout the years, in ubicomp much of the effort has been devoted to address hardware and software challenges, but the area of designing systems in which device are distributed as such has been barely touched. Hence, with the explosion of devices in the form of mobile phones and tablets having a wide range of sensing capabilities, modeling of context-awarene software became *device-centric*. This means that designers of mobile context-aware systems base their de-

2 Ubiquitous Computing

sign mostly on device sensory data.

In our everyday lives we encounter and interact with a wider range of mobile device. But the spectrum of everyday, physical objects we interact a lot broader. As context modeling became device centric, physical objects have been greatly overlooked making it a challenge to work design for them in a pervasive system. To address the struggling on conceptually incorporating the real world into the system design, Pederson in [7] introduced a *body-centric* modeling framework that incorporates physical and virtual objects of interest on the basis of proximity and human perception, framed in the context of an emerging "egocentric" interaction paradigm. "Egocentric" signals that it is the human body and mind of a specific human individual that acts as center of reference to which all modeling is anchored in this interaction paradigm. Part of the egocentric paradigm is the Situative Space Model (SSM): an interaction model and a design tool which captures what a specific human agent can perceive and not perceive, reach and not reach, at any given moment in time.

Evaluating the SSM consisted in creating a real-life setup, with digitally enhanced physical objects (various sensors to track agent proximity) and using a wide range of technologies to track the agent's current situation (body position, orientation, visual spectrum etc.). This is a costly and time consuming process making further development and evaluation of the egocentric paradigm a challenge. As Pederson also stated in [8]: "accurate tracking of objects and mediators needed for real-time application of the SSM will probably remain a challenge for years to come".

In this Master Thesis we aim at designing and developing a simulation environment for mobile context-aware system design, while modeling our framework to fit all SSM related concepts.

Finally, I want to emphasize on the need for this simulator, relating to the need for simulators in ubicomp as identified by Reynolds, Cahill and Senart in [9]: "The use of simulation technology in ubiquitous computing is of particular importance to developers and researchers alike. Many of the required hardware technologies such as cheap reliable sensors are only reaching maturity now, and many of the application scenarios are being designed with the future in mind and well in advance of the hardware actually being available. Furthermore, many of the target scenarios do not lend themselves to onsite testing, in particular, scenarios which require deployment of large numbers of nodes or devices. In addition, simulation enables researchers to evaluate scenarios, applications, protocols and so forth without the difficulties in dealing with hardware sensors and actua-

tors, and also offers greater flexibility since it is easy to run a set of simulations with a range of parameters".

1.1 PROBLEM STATEMENT

1.2 GOAL

1.3 METHOD

1.4 THESIS OVERVIEW

Thesis Overview goes in here

2 RELATED WORK

2.1 A TESTBED FOR EVALUATING HUMAN INTERACTION WITH UBIQUITOUS COMPUTING ENVIRONMENTS

TATUS [6] is as a ubiquitous computing simulator aimed at overcoming the challenges of effectively evaluating human interaction with adaptive ubiquitous technologies. These challenges are mainly imposed by the cost and logistics of building and controlling the context in such real-life environments. In other words, TATUS is a simulator supporting research and development of adaptive software controlling ubiquitous computing environments.

Figure 1 depicts the high-level overview of the simulator. We can identify two main components: the 3D Simulated Ubiquitous Computing Environment and the System Under Test (SUT).

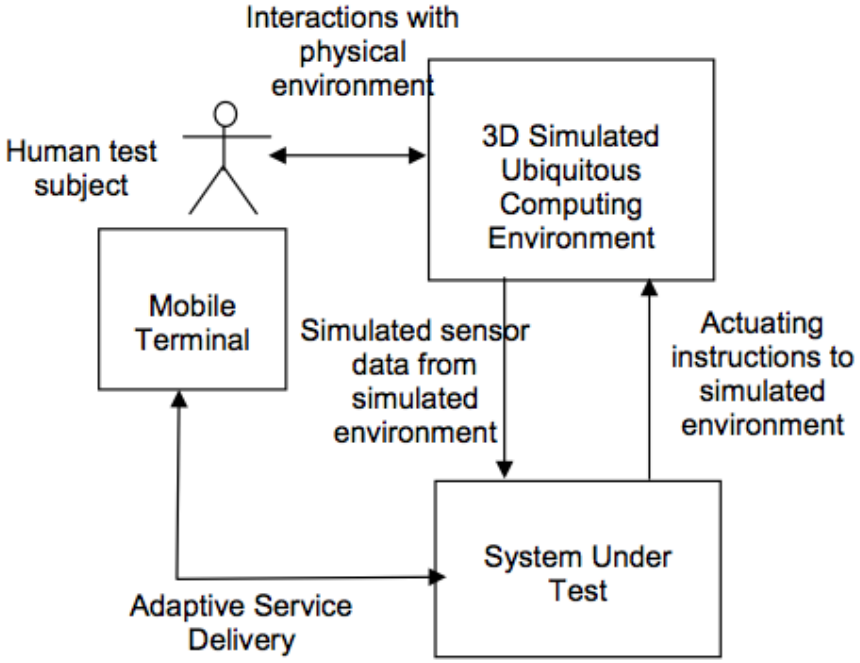


Figure 1: High-level simulator overview

The 3D simulator was implemented on top of the Half-Life[1] Game Engine's software development kit (SDK), written in C/C++. Half-

1 http://www.valvesoftware.com

Life is a 3D first-person-shooter network game. It is implemented based on client-server architecture with multi-player capabilities (up to 32 players simultaneously). The reason for choosing this game engine was to exploit the 3D graphics engine in order to provide a realistic user experience.

The actual simulated environment is created using a map editor from Valve called Hammer (a drawing tool for building maps). This can be used to generate complex and realistic environments as the tool offers a wide range of graphical editing possibilities, as exemplified in Figure **??**. The SDK is then able to load a simulated environment's physical settings from the files generated by the map editor.

Figure 2: Simulated meeting room with multiple characters

To simulated sensors and actuators they have exploited a concept called *Triggers* from the SDK. They are used to generate associated events based on a player's movements and location. The state of these triggers together with the agents location at a certain point in time represent the state of the simulated environment. To allow easy access to the data, the simulator exposes an API which offers both querying and modifying capabilities. Queries allow to select certain aspects of the current state based on various parameters, whereas through modifications the SUT can impose changes on the simulated environment.

A SUT is written as independent pieces of software. It can even run on a separate machines, while multiple SUTs can connect simultaneously to the same simulator. To allow SUTs to run on separate

machines the simulator has embedded networking capabilities and communicates through messages. Outgoing messages carry state information, while incoming messages care instructions to modify the simulated environment. Moreover, to abstract out the network interaction from the SUTs, the simulator provides a Proxy making the communication from the test software's side really easy.

2.1.1 *Conclusions*

One of the main challenges in developing the simulator was that the learning curve of the SDK, in order to extend it, was difficult and time-consuming. But, as a result, they have tried to provide a flexible simulation environment where researchers can easily simulate other environments without the need to modify any SDK level code.

The context-awareness is limited to sensors and actuators reacting to the user's position. This is actually not that bad because in the game engine SDK the developer can extract the position of entities, determine proximity of other entities within a given radius and determine the presence of other entities within a field of view. All these data helps to implement simulations for a wide range of sensors.

The user control provided by the game engine is pretty advanced allowing the agent to move in any direction, crouch, jump, sit etc. This is a big plus as it adds a good sense of reality to the game.

Multiple users can join and experience the same simulated environment at the same time. Not all player have to be human, not-player-characters (controlled by the AI of the engine) can be present as well.

Unfortunately the simulator is not open-source so making it unusable for research outside the institution it was developed in.

2.2 SERVICES INSIDE THE SMART HOME: A SIMULATION AND VISUALIZATION TOOL

In this work [5] the aim is to reduce the testing costs of smart homes. The goal was achieved by implementing a simulation and visualization tool which replaces services in a smart home with virtual stubs behaving just like the real hardware installed in a house.

For the implementation of virtual stubs they have employed the service-oriented computing paradigm, which is widely used to implement systems requiring high interoperability, scalability, security and reliability. One of the main advantages is that such software can

seamlessly integrate other systems written in other programming languages and even deployed under different operating systems.

The simulation scenarios are built using Google SketchUp[2]. This was extended with a set of tools extending its visual representation of a house with virtual home interactive web services supporting SOAP messages. Further, the visualization component is written as a set of plug-ins for Google SketchUp.

The simulation and visualization tool allows to simulate any possible home automation scenario, with the possibility of modeling the user and its interactions with the home.

2.3 SIMULATION OF SMART ENVIRONMENTS

Armac and Retkowitz developed a tool called *eHomeSimulator*, which was build in order to support the simulation of smart environments, or as the authors refer to them in the present work *eHomes* [3]. The motivation behind the work is that smart environments constitute already an important research area, but building a real eHome is associated with high effort and financial costs. Therefore, the eHome-Simulator helps in abstracting out from creating buildings (the actual physical environment) and purchasing devices, allowing us to focus only on software engineering aspects and challenges involved in eHome development (e.g. developing services, deployment etc.).

Describing in details the eHome is out of scope, but for clarity, we will a offer a short description. Therefore, eHome is basically a framework consisting of a hardware platform (the residential gateway) and a software platform (service gateway, runs on top of the hardware platform). The devices and appliances, which make up the eHome, are connected to the hardware platform. They are of two basic types: sensors and actuators. Sensors offer contextual information like temperature, humidity etc. while actuators can change the environment's state (e.g. a speaker, a heater etc.). The hardware platform is governed by the software platform which acts as a runtime environment for eHome services. These services can be of two types: basic and integrating. Basic service are meant to control devices (e.g. a lamp driver) while the integrating services are composed of various basic service. These services are developed based on the OSGi component model [2], imposing a highly decoupled architecture to the service model and offering high reusability of the developed services. One novelty introduced by the architecture of the framework is that service can work both with real device and simulate device.

2 http://www.sketchup.com

The eHomeSimulator is built on top of eHome framework to allow intuitive interaction and to visually represent the state of the simulated environment. In the evaluation process of the simulator, eHome environments consist of three main elements: rooms devices and persons (agents interacting with the environment). The graphical representation is a 2D view with a view point from top, which is built up from three layers:

- The bottom layer contains the graphics to represent the background (floor, walls and furniture) build up from tiles placed one next to each other.
- On top of the background, there is the middle layer representing the graphics of the devices. These are only static devices and they can have different colors based on their state.
- The top layer represents the agents interacting with the environment. They can move freely in all accessible areas, interacting with the devices.

There is a environment editor which assists in setting up a simulated environment. The process starts by designing the room, walls and furniture using Google Sketchup [3] which then is uploaded into the editor. On top of this base we can further place the devices and appliances.

The eHomeSimulator's architecture is based on the Model View Controller design pattern [4]. The model of the system holds the data structures representing the current state of the system, the view implements the graphical representation based on the current model and the controller process the user interaction with the simulator, updating the view according to changes in the model.

The image depicted in Figure 2 displays a simulation in progress. On the right side, we can observe the agent interacting with the environment while on the left side there is a control panel displaying various context data of the current status of the system and allows to interact with nearby devices (e.g. turn on/off a lamp).

3 http://sketchup.google.com/

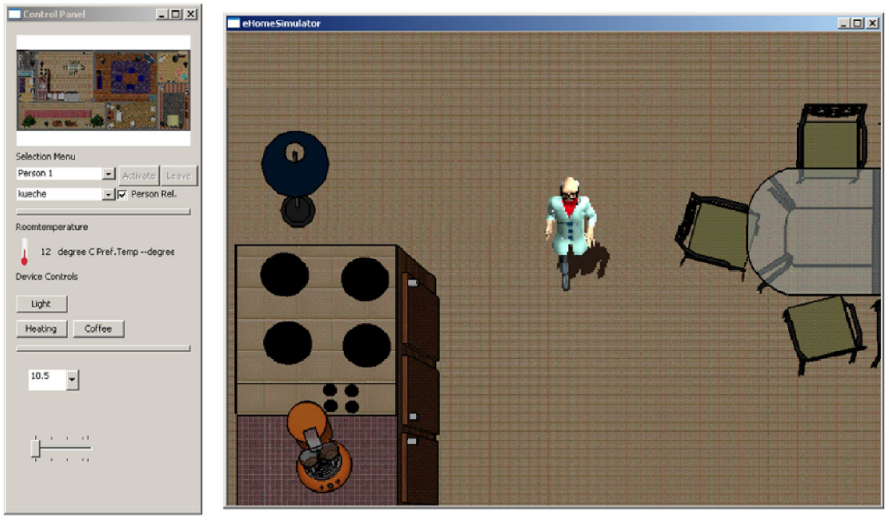


Figure 3: The GUI of the simulator

The eHomeSimulator has an architecture providing loose coupling between components. Also, the underlying framework's architecture offers great support for reusability of services and integration with real-life devices. The devices are trivial, offering basic interactions (e.g. on/off) and they are static (the agent can't move them). Actually, all the agent can do is move around between some given bounds and interact with some of the device. There is no support for mobility of devices and wearable device, nor is it in plan for future work. Moreover, the representation of the agent is in 2D where he can be facing 4 directions, offering no support for various states like sitting, standing etc.

The project is close sourced, making it impossible to extend, to reuse or to contribute to it. In conclusion, there are a few lessons to be learnt from the architectural choices made within this project.

BIBLIOGRAPHY

[1] Gregory D Abowd, Anind K Dey, Peter J Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In *Handheld and ubiquitous computing*, pages 304–307. Springer, 1999.

[2] OSGi Alliance. Osgi service platform core specification release 4 (august 2005).

[3] Ibrahim Armac and Daniel Retkowitz. Simulation of smart environments. In *Pervasive Services, IEEE International Conference on*, pages 257–266. IEEE, 2007.

[4] Gamma Erich, Helm Richard, Johnson Ralph, and Vlissides John. Design patterns: elements of reusable object-oriented software. *Reading: Addison Wesley Publishing Company*, 1995.

[5] Elena Lazovik, Piet den Dulk, Martijn de Groote, Alexander Lazovik, and Marco Aiello. Services inside the smart home: A simulation and visualization tool. In *Service-Oriented Computing*, pages 651–652. Springer, 2009.

[6] Eleanor O'Neill, Martin Klepal, David Lewis, Tony O'Donnell, Declan O'Sullivan, and Dirk Pesch. A testbed for evaluating human interaction with ubiquitous computing environments. In *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*, pages 60–69. IEEE, 2005.

[7] Thomas Pederson, Lars-Erik Janlert, and Dipak Surie. Towards a model for egocentric interaction with physical and virtual objects. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, pages 755–758. ACM, 2010.

[8] Thomas Pederson, L-E Janlert, and Dipak Surie. a situative space model for mobile mixed-reality computing. *Pervasive Computing, IEEE*, 10(4):73–83, 2011.

[9] Vinny Reynolds, Vinny Cahill, and Aline Senart. Requirements for an ubiquitous computing simulation and emulation environment. In *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, page 1. ACM, 2006.

[10] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.