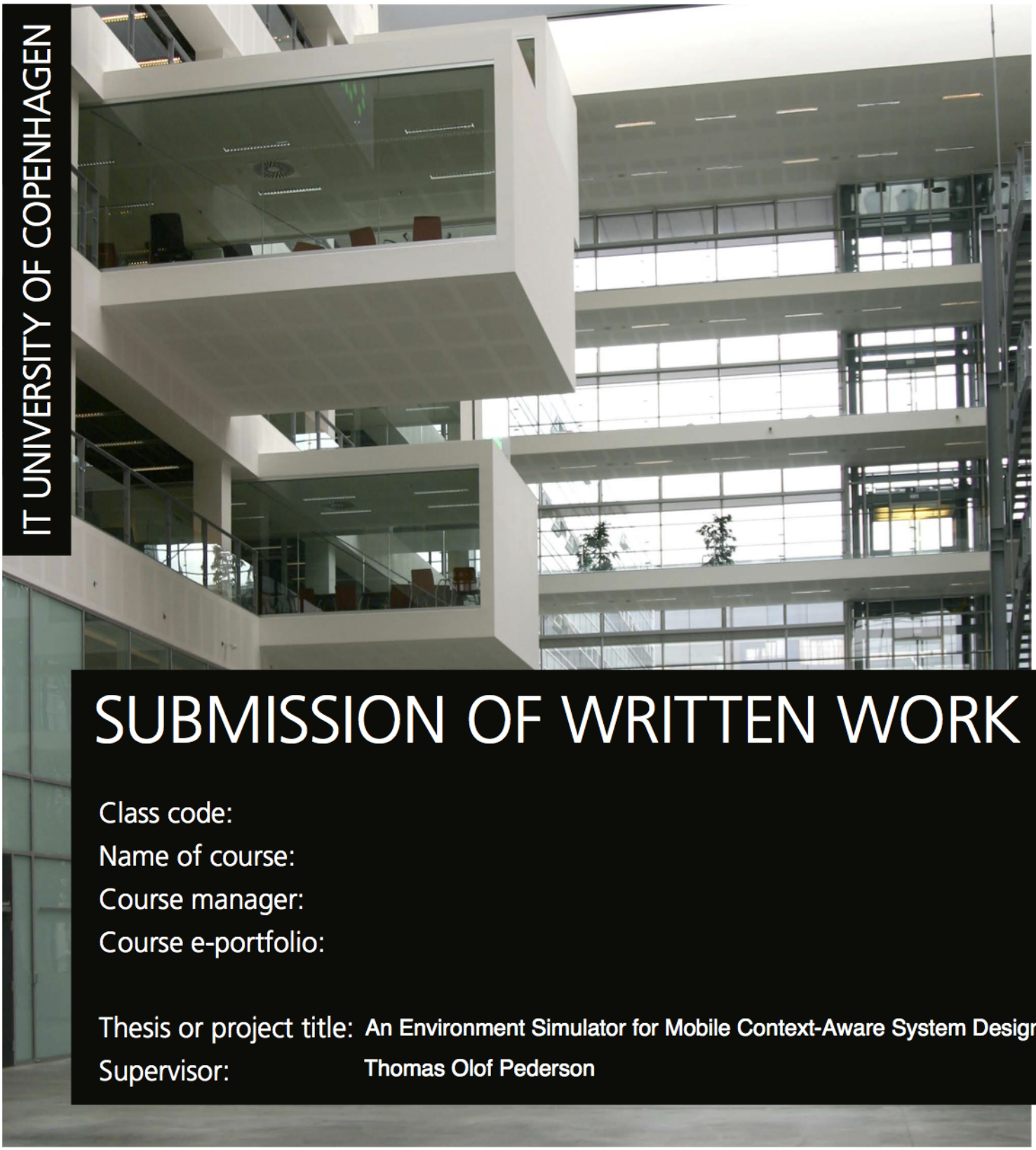


IT UNIVERSITY OF COPENHAGEN

SUBMISSION OF WRITTEN WORK

Class code:

Name of course:

Course manager:

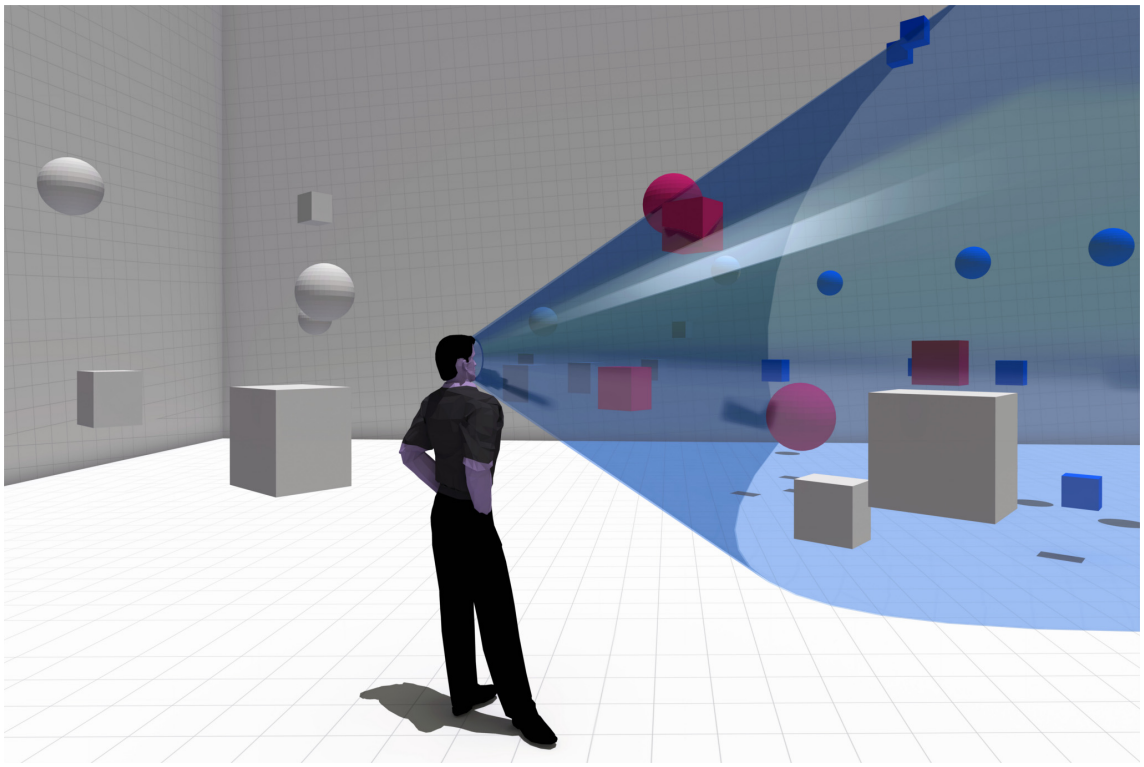
Course e-portfolio:

Thesis or project title: **An Environment Simulator for Mobile Context-Aware System Design**

Supervisor: **Thomas Olof Pederson**

Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1. **Károly Szántó**	**14/07-1985**	**ksza**@itu.dk
2.		@itu.dk
3.		@itu.dk
4.		@itu.dk
5.		@itu.dk
6.		@itu.dk
7.		@itu.dk

An Environment Simulator for Mobile Context-Aware System Design



Author:
Károly Szántó

E-Mail:
ksza@itu.dk

Supervisor:
Thomas Olof Pederson

E-Mail:
tped@itu.dk

Master's Thesis
IT University of Copenhagen

June, 2014

ABSTRACT

Designing and developing mobile context-aware systems in a real-life set-up represents a tedious, costly and time consuming process. This is why the use of simulation technology in ubiquitous computing is of particular importance to developers and researchers alike. Simulation enables researchers to evaluate scenarios and applications without the difficulties of dealing with hardware sensors and actuators [1].

Some state-of-the-art systems simulate pervasive computing applications in smart environments based on sensors and actuators; others are focused on simulating ubiquitous computing devices before physical prototypes are available.

We believe that physical objects and devices are an essential part of the context of a human agent. All the aforementioned simulators are driven by the presence of a human agent within the simulated environment, but none of them provides explicit details about what the human agent can perceive and act on, in a given moment in time, to the context-aware logic which the system designer is about to design.

We have developed EgoSim, a mobile context-aware simulation framework which classifies objects surrounding the human agent according to the Situative Space Model (SSM). The SSM defines object (physical or virtual) classifications with respect to what degree they can be interacted with, at any given point in time.

To evaluate our system we have recruited 17 participants with vast experience in the field of software engineering, which have provided valuable feedback. We have assessed the quality of our system based on 4 characteristics: usefulness, usability, responsiveness and accuracy of the classification process.

All participants have found EgoSim useful, highly usable and responsive, concluding that we have successfully designed a solution based on SSM to the aid ubiquitous system design, contributing with another dimension to the worlds of ubiquitous computing simulators.

We conclude this work with a number of improvements to our system and a list of new features as a result of the evaluation. The classification process will be given priority as it presents inaccurate results in edge cases.

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth [2]

ACKNOWLEDGEMENTS

First and foremost I offer my sincerest gratitude to my supervisor, Thomas Olof Pederson, who has supported me throughout my thesis with his patience, knowledge, continuous involvement and tireless help, whilst allowing me the room to work in my own way. His useful remarks and engagement through this master thesis helped in shaping the quality of this work. Without him, this thesis would have not been possible.

Thanks to Domiţian Tămaş-Selicean for the reviews and advices throughout the thesis, but also for the encouragements throughout my studies at the ITU. Thanks to Dan Brânda for helping out to get better environment models. Without his help, the 3D models in this thesis would have not been of the same quality. Also, I would like to thank all the participants of the survey, who have willingly shared their precious time during the evaluation process.

I would like to thank Jakob Bardram and Mads Frost, which I had the honour to work with, for the opportunity to get hands-on experiences with developing pervasive systems. The work we have done together remains by far one of the most interesting and important projects I have been involved with throughout my career as a software engineer. Furthermore, thanks to the members of the pitLab for their friendship and for the utterly professional environment. These experiences, together with my thesis, made my life as a student at the ITU not only extremely interesting, but also inspiring due to all these exceptional people I have been in touch with.

Furthermore, I want to thank my parents for all the support they have provided throughout all of my studies and my life, for their encouragements, appreciation, sacrifices and love. I want to thank my sister, brother, family and friends for always being there for me.

Finally, I want to express my deepest love and appreciation to my life partner, Cătălina Borotici, for all the encouragement, support, kindness and unlimited understanding she has offered me throughout my studies. I will be forever grateful for your love.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

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 intuitive user interface, Mainframes would accept jobs in the form of punched cards, an early input mechanism for data and programs into computers. Only a group of people in each institution had the knowledge on how to operate 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 a novel input/output mechanism: graphical user interface, mouse, keyboard and other peripherals. This is the standard PC setup that we know and, after such a long time, still use. By this we want to point out that although we have came a long way, by hardware and software advancements, the human computer interaction (HCI) that is mostly used to interact with PCs is the same as envisioned a little more than 40 years ago.

The PCs revolution was followed by the idea of portable personal devices like laptops and 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”. [7]. This futuristic concept envisioned at PARC, which further shaped the way we see and use information technology and devices, is known as

1 Palo Alto Research Center

Ubiquitous Computing, or"Ubicomp" for short.

This introduction was beautifully summarised by Mark Wiser: "Ubiquitous computing names the third wave in computing, just now beginning. First were mainframes, each shared by lots of people. Now we are in the personal computing era, person and machine staring uneasily at each other across the desktop. Next comes ubiquitous computing, or the age of calm technology, when technology recedes into the background of our lives" [8].

1.1 CONTEXT AWARE SYSTEMS

The novel vision in Ubicomp can be viewed as a second major advancement in the world of information 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 one-to-many, where each user will own not one but many device. These device would range from PCs to 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". [9]. 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 a 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, automatically display the relevant information.

1.2 EGOCENTRIC INTERACTION PARADIGM

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, hence, with the explosion of devices in the form of mobile phones and tablets having a wide range of sensing capabilities, modelling of context-aware software became *device-centric*. This way, designers of mobile context-aware systems base their design mostly on device sensory data.

In our everyday lives we encounter and interact with our surrounding environment through a wide range of mobile devices and physical objects. While the current device-centric modelling makes it easy to incorporate devices, incorporating physical objects into a context-aware system design, remains a challenge. When trying to integrate physical objects into the system design, the designer is bound to think in terms of existing technologies that can facilitate their integration; for example, making a system aware of a chair, a designer might think about attaching various sensors to it (pressure, motion, proximity, etc). While the device-centric modelling makes it natural to think about devices, when it comes to integrate physical objects, it limits the designer instead of offering a natural way to solve the problem. To address the struggling on conceptually incorporating the real world into the system design, Pederson in [10] introduced a *body-centric* modelling framework that incorporates physical objects, mediators (i.e. devices) and virtual objects[2] 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 centre of reference to which all modelling 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.

The SSM is meant to categorize objects around the agent, into a number of sets, based on it's continuously evolving state as it interacts with the surrounding environment. The sets maintained by the SSM are, as described in [10]:

- World Space. Contains all physical objects, mediators and virtual objects that are part of a certain model.

2 part of a running software, like a window, a stream of sound, interacted with through mediators

- Perception Space (PS). Objects around the agent that can be perceived at a given moment (i.e. objects currently in the users field of vision).
- Recognizable Set (RS). Objects in the PS within recognition distance.
- Examinable Set (ES). Objects in the PS that are within their examination range.
- Action Space (AS). Objects surrounding the agent that are currently accessible to the agent's physical actions.
- Selected Set (SdS). Objects being currently physically or virtually handled (i.e. touched).
- Manipulated Set (MS). Objects whose state is currently being acted upon by the agent.

Evaluating the SSM was a tedious work and consisted in creating a real-life setup, with digitally enhanced physical objects (various sensors to track agent proximity and actions) while 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 [3]: "accurate tracking of objects and mediators needed for real-time application of the SSM will probably remain a challenge for years to come".

In order to simplify further research into mobile context-aware systems based on the egocentric paradigm, in this Master Thesis we aim at designing a simulation environment for mobile context-aware system design. The framework will monitor the agent's context in regards to the world space and the context of surrounding entities. Based on the design, we will implement the *EgoSim* framework[3]. To emphasize on the need for this simulation framework, we relate to the need for simulators in Ubicomp as identified by Reynolds, Cahill and Senart in [1]: "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 actuators, and also offers greater

3 The source code of the framework is available; see Appendix C

flexibility since it is easy to run a set of simulations with a range of parameters".

We are designing this system for Ubicomp researchers[4] and developers designing mobile context-aware systems that change their behaviour on how the body of the agent is in regards to the surrounding environment and entities. To offer the most to this target community, we would like the source code to be accessible for anyone who is interested, to provide feedback, to further develop it or to modify it in order to fit certain individual needs. These aspects are best covered by the open-source software model.

1.3 PROBLEM STATEMENT

Designing and developing mobile context-aware systems in a real-life set-up represents a tedious, costly and time consuming process. In addition, most systems require testing and experimentation in their target location for valid and beneficial results to be produced; however it is difficult and costly to set up a certain location to support the designed system. Moreover, incorporating physical objects into the system design, adds up to the aforementioned challenges.

1.4 GOAL

The goal of this project is to design and implement an open-source simulation framework that can help developers of mobile context-aware systems to explore design alternatives. The framework has to dynamically classify physical objects, virtual objects and mediators that are close to the human agent according to the situative space model (SSM) [3], allowing the context-aware system developer to design system logic on the basis of this classification. The human agent is to be represented as an avatar that can move around and perform basic interaction with objects in the simulated environment.

The requirements of the framework are:

1. Allow researchers to easily build a simulated environment of their target mobile context-aware systems, based on situative space model concepts. The researchers will simulate the interaction of the end users with the egocentric system, controlling a virtual avatar in the simulated environment to move around and interact with object.

4 From now one we will be referring to a researcher as someone designing a mobile context-aware system based on the egocentric paradigm, which changes its behaviour based on how the body of the agent is in regards to the surrounding environment and entities.

2. Classify dynamically, in real-time, the monitored objects that are close to the human agent according to the situative space model, keeping the SSM spaces updated. The classification algorithms will be triggered as the agent interacts with the environment (move around, look around, pick up objects, etc).

3. Provide an API for third party services to query the current state of the SSM spaces. The API will empower third party client services to query the status of a particular space or all spaces at once. Moreover, on top of the API, the framework will offer an easy way to visualize the SSM spaces in real-time.

4. To be open-source and freely available.

Chapter 3 details the solutions that we propose to accomplish these goals.

1.5 METHOD

In order to achieve the goals detailed in Section 1.4, we are going to take an empirical approach. First, we will research related work to assess what approaches have others taken in order to solve similar problems. Next, we will establish the best technological set-up we find suitable to fit our problem. Finally, we will design and implement a single solution which we continuously improve throughout the lifetime of this work.

To support our design decisions, in Section 1.6 we have imagined an end-to-end scenario on how our framework would be used. This scenario, together with the goals will guide both the design and the implementation of our framework.

Based on the goals and the scenario, we have determined three user roles for our framework, as detailed in Section 3.1.1. Once the framework will be implemented we will carry out a two step evaluation: in the first step we will evaluate the framework from the *simulation user's* point of view, while in the second step we will evaluate the system from the *system designer's* point of view.

For the first evaluation step we will implement a simulation prototype based on our framework which will be used as the target of evaluation in the first step. Using this prototype, the simulation users will test the usability of the simulator, the responsiveness of the ContextClient, the correctness and usability of the SSM sets, as well as how intuitive is the interaction with the environment.

For the second evaluation step, we will imagine a problem to be solved by the system designer using our framework. To reduce the

evaluation time, we will provide the participants with a ready made model of their supposed target environment. The system designers will evaluate the usefulness and ease-of-use of the EgoSim framework, by setting up the simulation of the evaluation scenario based on the framework and the documentation. In this step, the system designer will also assess the API provided by the framework for *third party services*.

We will wrap up our research by drawing conclusion based on our findings and establishing goals for future work.

1.6 SCENARIO

To better visualize the importance of the EgoSim framework, in this section we present a concrete scenario which has been used during the framework's evaluation. The labels in the scenario's description will be referenced in the requirements Section 3.1.

The hypothetical problem presented in this scenario is that families cannot make their homes secure enough for their children. To provide a solution for this problem, there's a need for a system to constantly monitor the objects a child should keep away from and should not be interacting with. Based on this context information, we can further build various software service to secure the house from the child's actions. One example of such service is the "Secure Outlets Service" (SOS). This service has to detect whenever a child is approaching an outlet with a small object, in which case it should shut down the electricity switch for outlets, preventing the child from the possibility of getting electrocuted.

A good solution for this problem could be based on the egocentric interaction paradigm, as it can categorizes all the objects of interest around the human agent. Before implementing the system in a real world set-up, it is best to validate the design by simulating it. To visualize how the EgoSim framework could help, this is how we imagine a researcher would use it:

> *The researcher starts out by setting up the simulation, using the EgoSim framework. First, she/he identifies the target environment – it resembles a living room where the child spends most of the time. Next, she/he needs a virtual model of the environment (1). The model is populated (1A) with everyday physical objects (1B) and devices (1C). Some of the simulated entities have to be constantly monitored (1D) by the system. Moreover, some of the entities should have the option to be picked-up (1E) and put-down (1F) onto surfaces (1G) (e.g. a table, the floor, etc). Optionally, picked-up entities could interact with other entities (1H), other than surfaces. Finally, the monitored entities need to provide the necessary*

configuration information required by SSM model in the egocentric interaction paradigm (1I).

After the researcher has finished setting up the simulation, using the EgoSim framework, she/he runs the simulation in order to test out the system design. Once the simulation is started up, the researcher finds himself impersonating a child, the entity this system is designed for. The child is represented by a virtual avatar (2), placed within the simulated target environment, in this case the living room. The avatar can be moved around the living room (2A) and can look around to inspect the surroundings (2B). If the agent is close enough to an entity, it can be interacted with (i.e. picked-up, put-down, etc) (2C). As any of the previous actions are carried out, the SSM monitoring service (2D) keeps track of the agent's position as well as the distance to each object in the agent's visual spectrum (what can be seen at a certain moment in time). Based on this information, the entities are being classified, in real time, into SSM sets (2E).

The Secure Outlets Service is implemented as a third party service, listening to changes in the SSM sets through the EgoSim's API provided by the running simulation. As the simulation unfolds, the researcher controls the avatar to pick up a pen located on the living room table. With the pen picked up, the agent is controlled to approach one of the outlets; if close enough, the service detects an immediate possible threat, as there is a high possibility the child could insert the pen into the outlet. The service shuts down the electricity switch for outlets, preventing the child from the possibility of getting electrocuted (3).

The points in the above scenario marked with (1-3) will help motivating the high level requirements we will come up with for EgoSim in Section 3.1.

1.7 THESIS OVERVIEW

The rest of the work is structured as follows: Chapter 2 reviews similar systems, with respect to the criteria defined in the current work.

Chapter 3 and 4 constitute the core of the thesis. Chapter 3 presents the design of the system, while Chapter 4 discusses the implementation details of the practical work.

Chapter 5 is dedicated to the evaluation of EgoSim. Chapter 6 summarizes the contributions brought by the this thesis and presents the main goals of our future work.

2

RELATED WORK

Researchers studying Ubicomp environments have typically been facing one common problem: evaluating the system. This is because to set up the target environment requires space, money and a lot of time; and even if they have access to the whole infrastructure, sometimes is almost impossible to experiment on real case scenarios because of the technological limitations on gathering the contextual data required by the targeted system. Even if all these challenges are overcome, real and large scale experiments are really expensive, hence only a limited number can be conducted.

To address these issues, a number of projects have tried to develop simulation tools and frameworks in various research areas, some of them being generic, others being more targeted towards a certain domain.

In this chapter we will present the work in this area, we found most relevant and inspiring for this work.

2.1 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* [11]. 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 eHomeSimulator helps in abstracting out from creating buildings (the actual physical environment) and purchasing devices, allowing the researcher 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. 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). 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 [12], imposing a highly decoupled architecture to the service model and offering high reusability of the developed services.

The eHomeSimulator was built on top of the 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 consisted of three main elements: rooms, devices and persons (agents interacting with the environment). The graphical representation was 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 colours 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 an environment editor which assists in setting up a simulated environment. The process starts by designing the room, walls and furniture using Google Sketchup[1] 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 [13]. 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 1 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 contextual data of the monitored entities and allows to interact with nearby devices (e.g. turn on/off a lamp).

1 http://www.sketchup.com

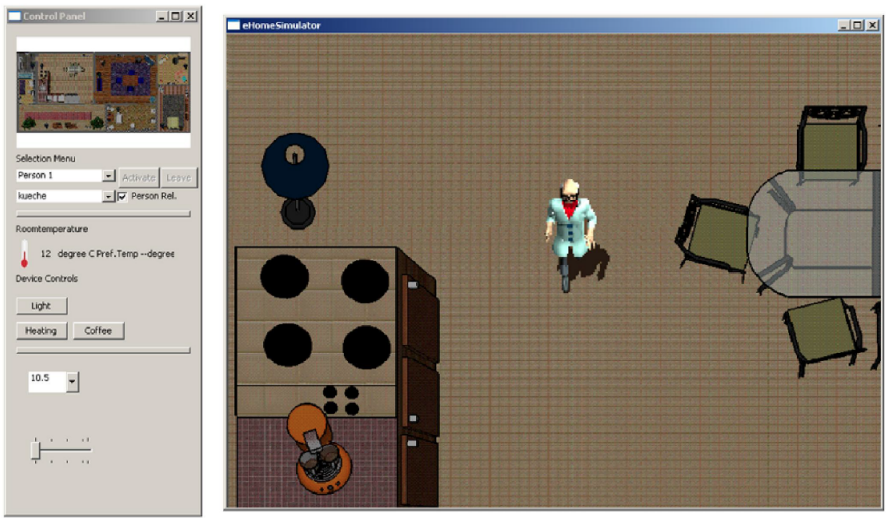


Figure 1: The GUI of eHomeSimulator

The framework allows to simulate only sensors and devices. Physical objects are passive and unidentifiable, they are part of the first layer, the bottom layer containing the graphics to represent the background (floor, walls and furniture).

As for interacting with the simulated environment, the user can manipulate the agent (the virtual avatar), to walk around in the 2D environment. Given the nature of the visualization, the agent can be facing four directions: forward, backward, left or right. Sensor and devices are activated on a per room basis, not on actual proximity to each individual entity. Devices in the current room can be operated through the *Control Panel* (e.g. turn lamp on/off).

2.2 SIMACT

SIMACT: a 3D Open Source Smart Home Simulator for Activity Recognition with Open Database and Visual Editor.

SIMACT is a smart home infrastructure simulator, developed in Java, designed to help researchers working in the field of activity recognition [14]. This work is specifically focused on the interaction between an agent and the surrounding environment in the smart house. It is built to reproduce everyday life scenarios, on a step-by-step basis.

The simulator was built entirely on Java based technologies. The GUI was based on the swing library, while the 3D renderer was based on the Java Monkey Engine (JMonkey) version 2.0 [15]. For the 3D design, they have used a modelling tool for house design and interior

accessories from Google called SketchUp[2].

To further detail, we will briefly discuss on the system's architecture, as illustrated in Figure 2.

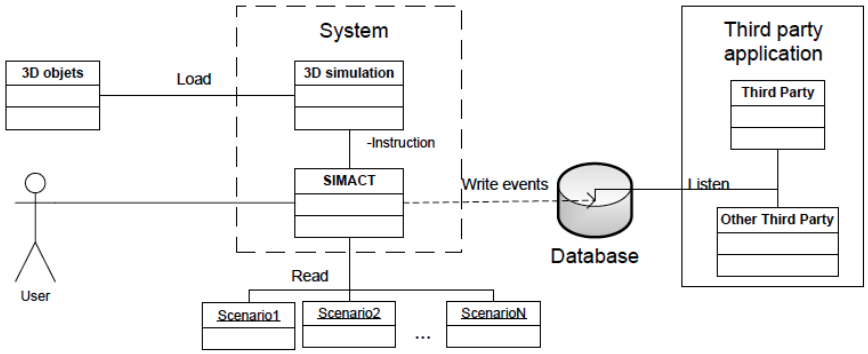


Figure 2: SIMACT: System Architecture

To create a custom simulation, a researcher starts out by defining the environment and the object models contained within. The simulator loads these definitions to create the virtual model of the environment and to initialize the 3D simulator. To define the interaction with the habitat, the researcher provides the simulator with an XML file defining the interaction scenario. The scenario is defined as a sequence of steps, which is read, interpreted and execute by the simulator (it plays the role of an interpreter). As the steps are executed and the environment gets modified by these actions, the generated events are written into a database. Further, third party applications can communicate with the database to retrieve data in real-time, which can be used in the logic of the application.

A snapshot of the simulation tool in action is depicted in Figure 3.

2 http://www.sketchup.com

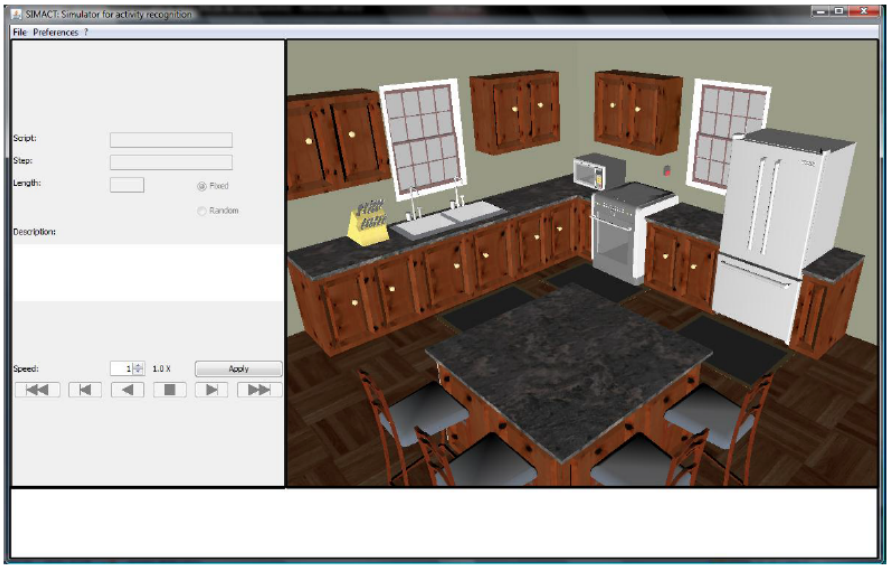


Figure 3: SIMACT: Simulation Tool in Action

The novel approach in this simulator is that entire scenarios and interactions can be designed only by using scripts and visual editors, without having to write one line of code. For the 3D design, the researchers have used a modelling tool for house design and interior accessories from Google called SketchUp. This comes with the advantage of a community maintaining a library of free 3D models, where one can find many accessory that would fit in a home.

The framework enables simulation of both devices and physical objects. The entire simulation is controlled by the framework as it interprets the simulation scenario, described in an XML file. This is a step-by-step description of how various parameters of the system (e.g. the temperature in a room) and of objects (e.g. door opened/-closed, lamp on/off) evolve. During the simulation, context data can be retrieved by third party service from the data base exposed by the framework.

The 3D visualization is a simple animation of the simulation allowing the researcher to look around within the ongoing simulation to better observe the evolution of the system. "In the 3D zone, one can freely move the camera around the environment to take different points of view and then decide where is the most appropriate position to see what is going on. It is simply done by clicking on the frame and using a keyboard: the arrow keys to rotate and pg up/pg down to zoom" [14].

The framework's source code is publicly available as open-source.

2.3 DIASIM

DiaSim: A Parameterized Simulator for Pervasive Computing Applications.

DiaSim is a Java based simulator for pervasive computing applications based on sensors and actuators [16]. In this project, the simulation process starts out by defining a high-level description of the target pervasive computing environment, in a domain specific language: DiaSpec. Part of this definition are *classes of entities* and *data types* which can be exchanged by the entities. Based on this definition, DiaGen produces a customized *programming framework* and an *emulation layer*.

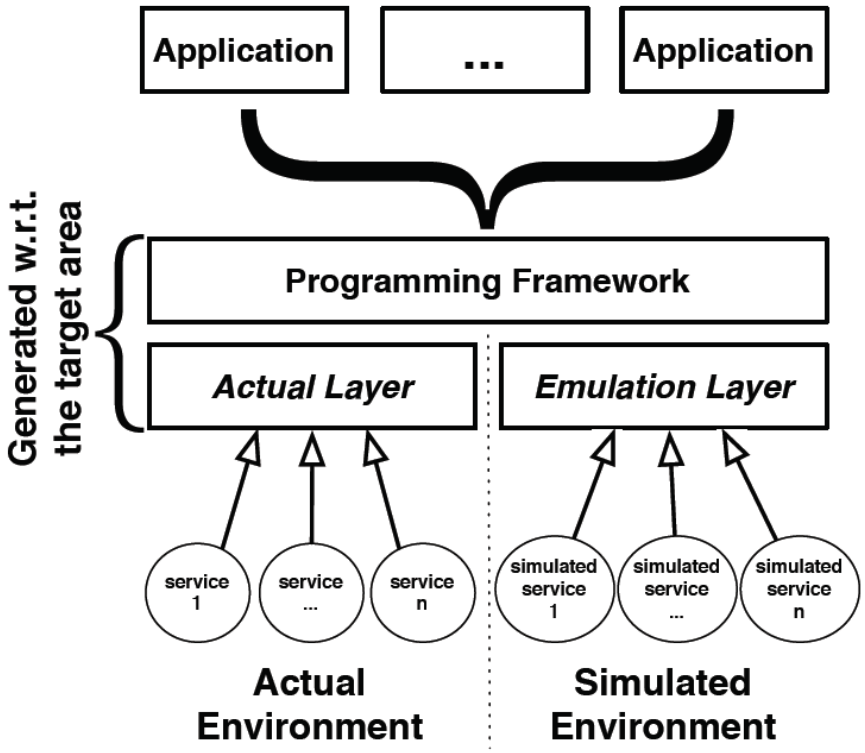


Figure 4: DiaSim layered architecture

As depicted in Figure 4, the simulator has a modular, layered architecture. Applications written using the generated programming framework, can be run both in a simulated and a real-world environment. In the real environment, the data comes from real sensors, while in the simulated environment the data comes from simulated sensors in the emulation environment. Besides the components mentioned so far, DiaSim also includes a simulation renderer, enabling the researcher to visually monitor and debug the pervasive comput-

ing system.

At the heart of this simulator, is the simulation model. The core entity is called a *stimuli*. This represents changes of the environment observable by the sensors in the system. Entities that generate stimuli are called *stimulus producers*, producing only one type of *stimulus*. The generated stimuli may trigger sensors (e.g. a motion detector) which publish events, that may in turn stimulate actuators or services. Making an analogy with object-oriented programming, stimulus would be a class, stimuli would be an instance of that class and the stimulus producer would be a factory [17], producing instances of the stimulus class based on a certain set of rules. These rules define the evolution of the stimuli in terms of space, time and intensity (e.g. an agent moving from one location to another). Each stimulus has a type associated with it, matching the type at least one sensor in the system.

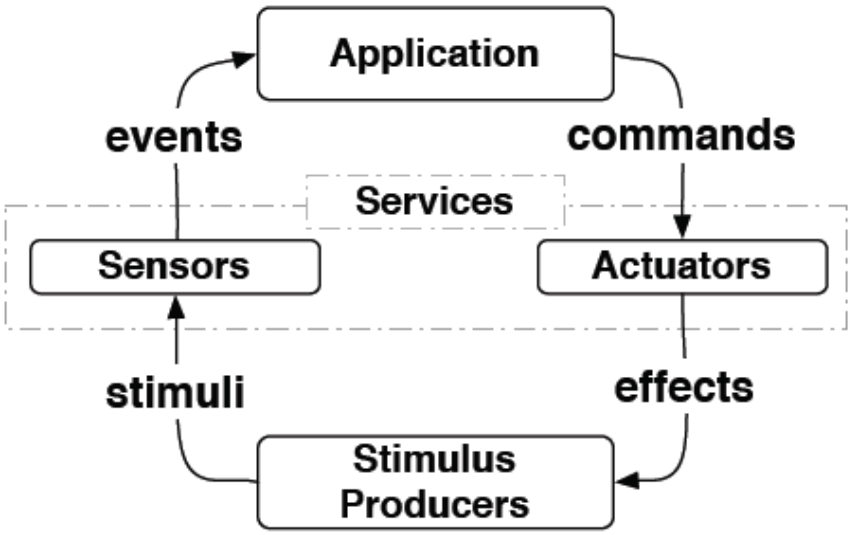


Figure 5: DiaSim simulation model

As illustrated in Figure 5, besides stimulus producers, the simulated environment also contains simulated services; they process stimuli, perform actions and interact with other services. The two key services in this simulator are *sensors* and *actuators*. The interaction between services and the environment are based on one of the existing three interaction modes:

- Command (one-to-one, synchronous interaction). Usually received by actuators to modify their state (e.g. on/off command for a light)
- Event (one-to-many, asynchronous interaction). Usually generated by sensors when they receive stimuli (e.g. a motion detec-

tor when receiving location stimuli matching its room, would generate an event containing the room's number)

- Session (one-to-one with information exchange over a period of time). For example, when motion from an unknown source is detected in a restricted area, an audio notification service could stream a warning audio notification to a nearby speaker service.

2.3.1 *Siafu*

To render the simulation, DiaSim was coupled with Siafu[3], a highly customizable, open-source Java based simulator for mobile context-aware applications and services [18].

Siafu was meant to be a flexible simulator. Hence, they have separated the main information sources from each other:

- Agent Model. Takes decision on what a certain agent should do given it's current context and the status of surrounding entities. As a result, the model will change the properties of an agent (e.g. walking). A special property is the destination. This will make the agent move in the surrounding environment; movement and path finding routines are handled automatically by the simulator.
- World Model. Consists of an environment model, places of interest (e.g. office, rooms) and global events model (e.g. holidays, happy hour at a restaurant)
- Context Model. Manages context variables used in the simulation.

As illustrated in Figure 6, the main models in DiaSim are extended from the three abstract information sources in Siafu: AgentModel, ContextModel and WorldModel. Further, DiaSim aggregates the simulated entities and stimuli producers. Based on this approach, DiaSim managed to easily integrate a 2D simulation renderer in their work.

3 http://siafusimulator.org/

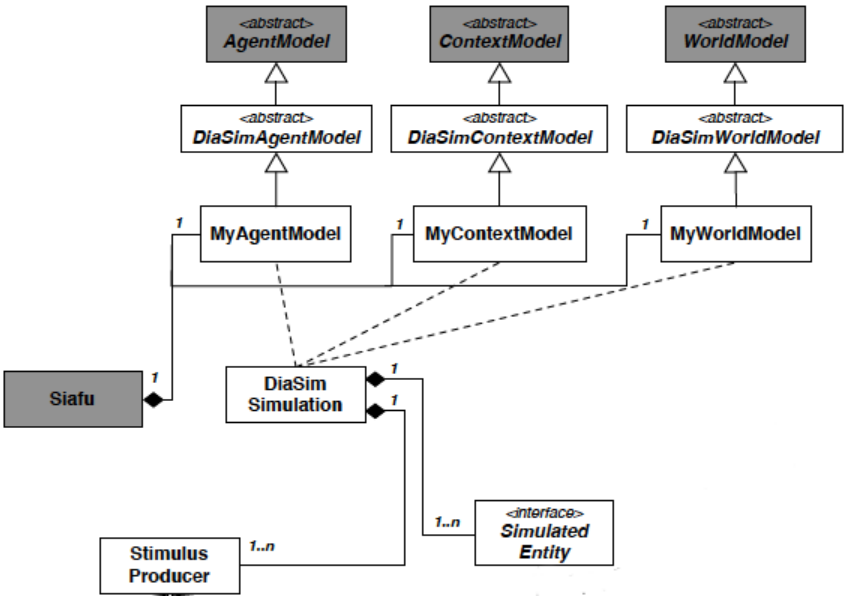


Figure 6: Siafu integrated with DiaSim

The simulated data is available in a 2D graphical representation and through a web service interface. Using the current 2D interface, the agent can be moved around the environment and can be facing four directions. The simulated sensors and actuators trigger actions based on the agent's proximity and the current context of other monitored entities in the system. At the moment of writing this thesis, no other interactions are available within DiaSim; the agent cannot interact with devices by explicitly triggering actions, nor can it interact with physical objects. Future work plans to visualize the simulation in a 3D graphical representation.

2.4 UBIWISE

UbiWise, A Simulator for Ubiquitous Computing Systems Design.

UbiWise [19] is a device-centric simulator. The target of this work was to allow researchers to simulate ubiquitous computing devices[4] before physical prototypes are available. The purpose of this is to empower the researcher to write software for the device and test it from a user interaction's point of view, in the actual physical environment the device was envisioned to be used in. This project emerged from two independent simulators: UbiSim and WISE.

4 networked, context-aware devices

To give an example of devices simulated with UbiWise, in the evaluation they have presented a wireless digital camera and wireless picture frames. The camera could easily connect to one of the frames and transfer a digital picture to be displayed.

UbiSim is a 3D environment simulator, aimed at producing context information in real-time, in a close to realistic environment. To simulate the environment, they have used the Quake III Arena (Q3A)[5] first person shooter gaming engine, written in the C programming language. On top of the raw simulated data outputted by Q3A, they have built a *context server* which processes the simulated data and delivers meaningful context data to external applications and services. The simulator is also able to process data from sensors in the real-world, generating mixed reality together with the simulated data.

WISE is a 2D device interaction simulator written in Java. This allows the user to interact with the software running on the device, interacting with real-world web services and other simulated devices.

In the 3D physical environment, the set of simulated devices are used in a context-aware manner, reacting to physical events and contextual changes. The devices might be portable, hand-held by a virtual agent the user controls (e.g. a digital camera enhanced with Internet connectivity) or they might be static devices, attached to a physical entity (e.g. a wireless enhanced picture frame on a wall). The simulation monitors the proximity between the devices, triggering events and allowing the software running on the devices to take specific actions.

In the 2D environment, the set of devices are presented in desktop-alike windows and they react to mouse clicks and network events. This allows to user to directly interact with the device through the simulated interface. Figure 7 depicts a screenshot of the 2D view.

Figure 7: UbiWise 2D view to interact with devices

5 http://www.idsoftware.com/gate.php

The simulator was developed on top of 3D game engine, empowering the agent to freely walk around the simulated environment. As the framework is device centric, the interaction between the agent and the environment is focused on interacting with the simulated devices: "the simulator concentrates on computation and communications devices situated within their physical environments." [19]. One of the devices might be hand-held and carried around by the agent. There is no mention in the paper about the ability of the agent to put the hand held device down or to pick up another one. Physical objects are part of the environment, but they are not monitored and their state is not part of the system's context data. Figure 8 presents a screenshot of the 3D view.

Figure 8: UbiWise: 3D view of the environment

UbiWise empowers the researcher to write actual software that runs on the simulated devices. The framework monitors proximity of the agent towards devices and proximity of the devices to each other, making this information available, together with other contextual data, to the software running on the devices.

The framework's source code is publicly available.

2.5 TATUS

A Testbed for Evaluating Human Interaction with Ubiquitous Computing Environments

Tatus [20] 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 9 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).

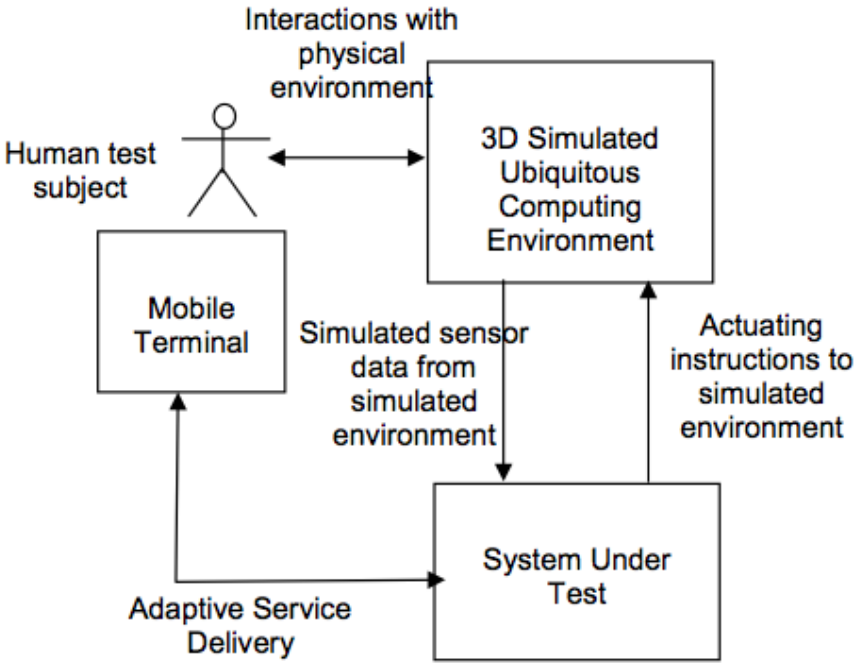


Figure 9: TATUS: High-level simulator overview

The 3D simulator was implemented on top of the Half-Life[6] Game Engine's software development kit (SDK), written in C/C++. Half-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). Choosing this game engine was meant to provide a realistic user experience. Figure 10 presents a screenshot

6 http://www.valvesoftware.com

of a running simulation.

Figure 10: TATUS: Simulated meeting room with multiple characters

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. The SDK is then able to load a simulated environment's physical settings from the files generated by the map editor.

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 carry instructions to modify the simulated environment. Moreover, to abstract out the network inter-

action from the SUTs, the simulator provides a Proxy making the communication from the test software's side easier.

As the simulator was developed on top of 3D game engine, it empowers the agent to freely walk around the simulated environment, while the interaction is based on proximity only. This means that entities will take a specific action when the agent is detected nearby; "when a player enters a region of a map occupied by a trigger the associated event is activated e.g. a door is opened" [20]. These triggers can be attached to both physical objects and devices, enabling both categories of objects to be interacted with, as such.

The framework exposes an API so that third party software can easily access and modify the context data within the ongoing simulation.

2.6 DISCUSSION

The eHomeSimulator 2.1 design promotes loose coupling between components. The underlying framework's architecture offers good support for reusability of smart home services and integration with real-life devices. Sensor and devices are operated on a per room basis, not on actual proximity to each individual entity. Devices in the current room can be operated through the *Control Panel* (e.g. turn lamp on/off). The simulated entities are static (the agent can't move them). Actually, all the agent can do is move from room to room, this will trigger sensors to be activated/deactivated, which in turn might change the state of other entities. There is no support for mobility of devices, nor is it in plan for future work. Moreover, the user can manipulate the agent (the virtual avatar), to walk around in the 2D environment. Given the nature of the visualization, the agent can be facing four directions: forward, backward, left or right.

The project is close sourced, making it impossible to extend, to reuse or to contribute to it. In conclusion, the simulator is targeted solely on simulating sensors in a smart home and how the presence of the human agent influences the environment. The framework was not designed to simulate the environment from the human agent's point of view, it was rather designed to simulate the evolution of sensors based on the human agent's presence. From this project's perspective, eHomeSimulator imposes a limited number of interaction possibilities. That is why we are not including it in further discussions.

Table 1 concludes this section with the characteristics of interest for the current work, highlighting whether or not they are offered

by the related systems. In this work we aim at designing all these characteristics for our framework as detailed in Section 3.1.

Characteristics	Simact	DiaSim	Tatus	Ubiwise
Visual representation of the simulated environment	yes	yes	yes	yes
Interact with the environment through a virtual avatar	no	yes	yes	yes
360°control of the avatar	no	no	yes	yes
Simulate sensors	yes	yes	yes	yes
Directly interact with physical objects	yes	no	no	no
Interact with physical objects using attached sensors	yes	yes	yes	yes
Interact with devices (mediators)	yes	yes	yes	yes
Interact with software (virtual objects)	no	no	no	yes
Pick objects up	no	no	no	no
Put objects down	no	no	no	no
Carry objects around	no	no	no	yes
Monitors the agent's location	no	yes	yes	yes
Agent's field of vision	no	no	yes	yes
Classify objects based on agent proximity	no	no	no	no
Provides API for third party services to access context data	yes	no	yes	no
Is open source?	yes	no	no	yes

Table 1: Features comparison

3 DESIGN

In this section we will present in details the possibilities and choices involved in accommodating the goals of this system. We start out by defining the requirements of the framework. We discuss briefly the the challenges of building onto some of the related systems. Next, we define the overall architecture of the system that can accommodate the requirements, followed by detailing the possibilities and choices for each component in the architecture. We conclude this chapter with the final design of the framework.

3.1 REQUIREMENTS

In Section 1.4 we have defined the main goals of the system as a list of high level system requirements. They are very generic and hard to establish an architecture upon, because they do not contain details of the functionality the end-user of the system will be able to use. To have a clear vision of what this system needs to achieve, in this section we will defined a list of requirements from the end-user's perspective.

We have decided to use the user stories concept [21], as a way to express the requirements of this system.

Agile Software Development is an iterative and incremental software development methodology, promoting adaptive planning, evolutionary development and delivery, while encouraging rapid and flexible response to change [22]. It is mainly used in the industry. One of the best practices in Agile Software Development for requirements is *user stories*. We have not employed Agile Software Development in this work, we have only used the user stories concept to express the system requirements.

"A user story describes functionality that will be valuable to either a user or purchaser of a system or software" [21]. User stories are written from the perspective of the system's end users. A specific group of users, will use the system in a certain way; the feature requested by this group represents a *user role*. The user role does not necessarily have to be assigned to a human user; it can also be an external service in need of accessing some data.

3.1.1 *User Roles*

Writing the requirements following the user story concept, helps us reflect upon the needs this framework must satisfy, from the end user's perspective. Based on the goals of the system defined in Section 1.4 and supported by the use case scenario defined in Section 1.6, we have first identified the user roles for this framework:

Role	Who	Role Description
System Designer	The researcher	As discussed in the introduction 1.2, someone designing a mobile context-aware system based on the egocentric interaction paradigm. Uses the framework to design a simulation of the egocentric system.
Simulation User	The researcher, a QA engineer or an end-user of the egocentric system	This role can be fulfilled by users interested in the resulting egocentric system.
Third Party Service	An individual piece of software	An individual piece of software that builds business logic based on the monitored context data, as the simulation unfolds.

Table 2: List of roles for the EgoSim framework

The "who" presents the possible user types that can fulfil a specific role, while the "description" briefly describes the role's main interest in the system.

3.1.2 *User Stories*

The following list of user stories represent the requirements of the EgoSim framework:

1. *As a system designer, I want to create a model of the environment my egocentric system will run in. The environment's model has to be populated with everyday physical objects and devices*. This requirement relates to goal 1. and emerges as a need from points (1), (1A), (1B) and (1C) in the scenario.

2. *As a system designer, I need to identify the objects I want to be monitored during the simulation*, as described in goal 2. and point (1D) in the scenario.

2.1 *As a system designer, I want to specify how certain entities can be interacted with*. Points (1E), (1F), (1G) and (1H) illustrate a few ways entities could be interacted with.

2.2. *As a system designer, I want to augment objects that will be monitored with information needed by the SSM model*, as illustrated in point (1I). In order for the SSM classification to work, it needs to be aware of specific parameters for each entity. By default, each monitored entity will carry a set of default SSM configuration data, but certain entities might need some adjusted values. The SSM configuration parameters are detailed in Section 3.1.4.

3. *As a system designer, I want to run a simulation based on the environment model I have created*.

4. *As a simulation user, I want to control a virtual avatar within the simulated environment*. This behaviour is mentioned in goal 1. and, as described in points (2), (2A), (2B) and (2C), I want to be able to control the movement and field of vision of the agent. Also, I want to be able to pick up certain objects, carry them around and make them interact with other objects (e.g. put it down on a surface).

5. *As a simulation user, I want to have a sense of reality in the simulated environment*. The user should comprehend and fell present in the target environment.

6. *As a system designer, I want the simulation to classify monitored objects into SSM sets as the agent is being controlled*. This requirement is mentioned in goal 2. and highlighted in the scenario throughout points (2D) and (2E). The SSM sets are detailed in Section 3.1.4.

7. *As a third party service, I want to access the content of SSM sets through a publicly available API*. Goal 3. covers this requirement, while point (3) describes a useful scenario.

8. *As a simulation user, I want to follow the state of SSM sets, in real time, as the simulation unfolds*. As described in goal 3., this requirement satisfies the need for an "easy way to visualize the SSM spaces in real-time".

While an extra requirement is needed to fully support the situative space model - namely, representation of and interaction with virtual objects - we leave that requirement for future work.

3.1.3 *Non-functional Requirements*

To strengthen the requirements above, we conclude this section with a list of non-functional requirements (NFR)[1]:

NFR	Relates to
The agent should not be able to pass through walls and other objects in the environment.	4.
The rendering of the simulated environment should be non-blocking. No matter how heavy are the computations carried out under the hood, the simulation user's experience should not be affected.	5.
Only objects within the field of vision of the user should be categorised	6.
The SSM classification should happen in real-time.	6.

Table 3: Non functional requirements

3.1.4 *SSM Sets and Configuration Parameters*

In this work, we have focused on two main parameters to classify objects around the agent: proximity and field of vision. This means that objects within the agent's field of vision will be classified based on the distance from the agent's current position. We have given a less generic interpretation to the SSM Sets defined in [3], to fit this use case, as presented in the list below:

WORLD SPACE contains the set of all entities (physical objects and mediators) in the environment's virtual model. The framework takes into account only the entities which have been identified to be monitored. Hence, not all visible objects visible are categorized.

PERCEPTION SPACE is part of space around the agent that is within the agent's field of vision at each moment. Objects within this space that are no further than *PERCEPTION_DISTANCE* from the agent, will be classified into this set.

RECOGNITION SET contains the entities that are currently in Perception Space and within their *RECOGNITION_DISTANCE*. Objects in this space can be directly associated with the agent's activities. For example, a hammer can be perceived as an object up to a certain distance, but when close enough, it can be recognised as a hammer.

1 constraints on the system's behaviour

EXAMINABLE SET contains the objects that are currently in Perception Space and within their *EXAMINATION_DISTANCE*. Based on this set it can be determined what actions can be performed with that object. For example, in the perception space a cell phone can be seen as a simple object, in the recognizable set it can be recognized as a mobile device and in the examinable set one can notice it has a screen, a power button and two volume button. We can deduct the type of action one can perform!

ACTION SPACE part of space around the agent that is currently accessible to the agent's physical actions. More specifically, the object has to currently be in Perception Space and within *ACTION_DISTANCE*. Objects in this set can be directly acted upon.

SELECTED SET objects currently being physically handled.

MANIPULATED SET objects whose states (external and internal) are currently being changed by the agent. Normally a subset of the Selected Set.

The image in Figure 11 presents a graphical representation of the SSM to clarify the relationship between the sets.

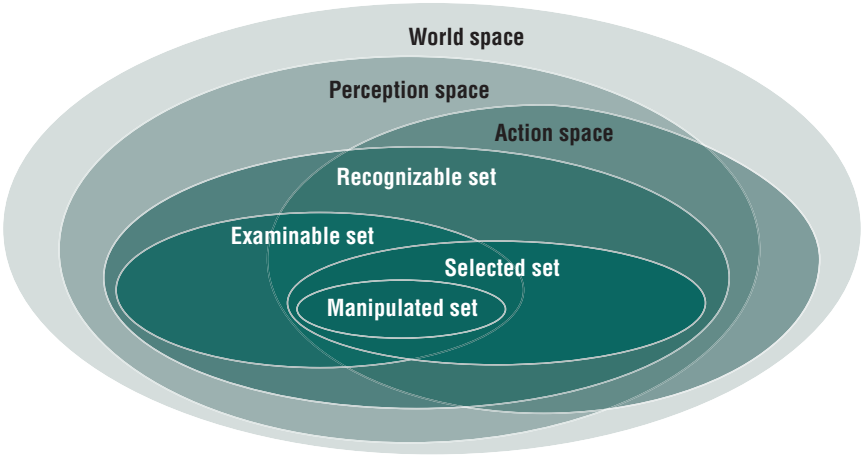


Figure 11: A situative space model (SSM). The spaces represent presence and approximate spatial relationship among physical and virtual objects with respect to what a specific human agent can perceive (perception space) and manipulate (action space) at a given moment in time. Whether objects are perceivable and manipulable depends on their relations to the human agent in all available interaction modalities (for example, vision, touch, and audio) [3]

.

As described in the definitions above, in order to correctly classify objects, they need to be augmented with the following parameters:

PERCEPTION_DISTANCE, *RECOGNITION_DISTANCE*, *EXAMINATION_DISTANCE* and *ACTION_DISTANCE*. Normally, they should be given meaningful default values, therefore configuring them should not be a mandatory step. Even so, some objects might need adjusted values, so the possibility should be given to the system designer.

3.2 CHALLENGES OF BUILDING ONTO RELATED SYSTEMS

While researching the systems presented in the related work chapter 2, we have analysed the possibilities of reusing some of them so we do not start from scratch. Unfortunately, none of them was fit for our approach. In this section we present the challenges we would face trying to design our solution in the context of some of the related systems.

3.2.1 *DiaSim*

After thorough analysis, we found DiaSim's 2.3 simulation model as being generic enough to possibly host our needs. The simulation model seems to be flexible and open for further modifications. The system is not open source and the research team is not yet open for external collaboration. We have contacted the research group at INRIA[2] proposing a collaboration in order to extend DiaSim in the directions required by this project. The answer we got back is that development of DiaSim is momentarily suspended as most of the group is focused on another project. They will resume development of DiaSim only six month later. Even so, they are yet unable to determine whether a partial release of the sources is possible.

Even if building on top of DiaSim would have been possible, some outstanding issues should be tackled. The simulated environment does not monitor the agent's field of vision. To accommodate this need, the renderer module should be entirely replaced. Besides, there is no support for direct interaction with physical objects and mediators. The only type of interaction is proximity based: i.e. a sensor attached to a door would command the door to open if the agent is within range. Moreover, DiaSim is lacking an open API to enable third party service to make use of the simulated system's context data.

3.2.2 *SIMACT*

SIMACT 2.2 is a smart home infrastructure simulator built to reproduce everyday life scenarios on a step-by-step basis. To make it easier

2 http://www.inria.fr

to comprehend the simulated system, SIMACT animates the simulation using the jMonkey Engine (JME) framework. SIMACT is not a tool to create simulations for simulation user, rather it a simulation framework for system designer design and run simulations of homes, with the possibility of observing the system's evolution over time in an animated 3D environment.

Although SIMACT does not offer the necessary means to support the implementation of our requirements, the JME game engine used to animate the simulations has caught our attention. JME has all the capabilities of a modern game engine which, as discussed in this chapter, we consider well fit to accommodate our requirements. In Chapter 4 we will argue for JMonkey as a possible candidate to be used in our work.

3.2.3 *TATUS*

TATUS 2.5 is a 3D simulator implemented on top of Half-Life's Game Engine. It has exploited the concept *Triggers* from the SDK to simulate sensors and actuators. They are used to generate associated events based on a player's movements and location. Therefore, TATUS already has proximity detection, but only based on sensors.

It would be possible to partially base our design on TATUS' triggers. By attaching such a trigger to each object we would like to classify based on the agent's proximity, we would probably have out of the box distance computation. But, TATUS does not simulate interaction with devices neither does it simulate interaction with everyday physical objects. Moreover, as discussed in section 5.4 the game engine TATUS is built upon is hard to master and get started with.

To conclude, TATUS is not open-source, making it unusable for research outside the institution it was developed in.

3.2.4 *UbiWise*

UbiWise 2.4 was designed to simulate the interaction of the agent with the device prototypes within the target environment, not with the actual environment. Moreover, the framework empowers the user to interact with the software running on them. Therefore UbiWise handles the representation of mediators and interaction with virtual objects. This could be a good starting point. But UbiWise is solely oriented towards simulating prototyped devices. The virtual agent can interact with the prototyped devices and with the software running

on them, some of the device may even be hand-held. It has no direct interaction with physical objects, the framework does no track the agent's position towards objects within the environment and does not determine the objects within the agent's field of vision. The underlying game engine might provide the necessary means to accommodate these requirements, but the effort depends greatly on individual experience in C programming.

To conclude, UbiWise was implemented with a radically different goal in mind and it poses an unnecessary technical challenge to try and accommodate EgoSim's requirements into its design.

3.2.5 *Conclusion*

In this section we have presented the challenges we would have faced trying to design our solution in the context of some of the related systems. None of the system were fit for our approach, but we had much to learn from them, nonetheless. Most systems which allow direct interaction between the agent and the simulated environment were build to some degree using game engines. We are considering to include game engines into our design as argued for in Section 3.5.

3.3 ARCHITECTURE

In this section we will describe the high level components we have designed to accommodate the requirements described in Section 3.1. To empower the system designer to set up a simulation we need a *Simulation Designer*. Using the simulation designer, the system designer is able to create a *Configured Environment Model*, where all the objects of interest for the simulation have been identified and configured. The *Simulation Runtime* loads a configured environment model and enables the simulation user to control an *Avatar* in order to interact with the environment. As the agent interacts with the environment, the *Context Manager* monitors the agent's position and field of vision, delegating the classification of objects around the user to the *SSM Classifier*. The *SSM Sets* can be visualized in the *Context Client* and can be accessed through the *API*.

For a better overview, the proposed architecture is illustrated in Figure 12.

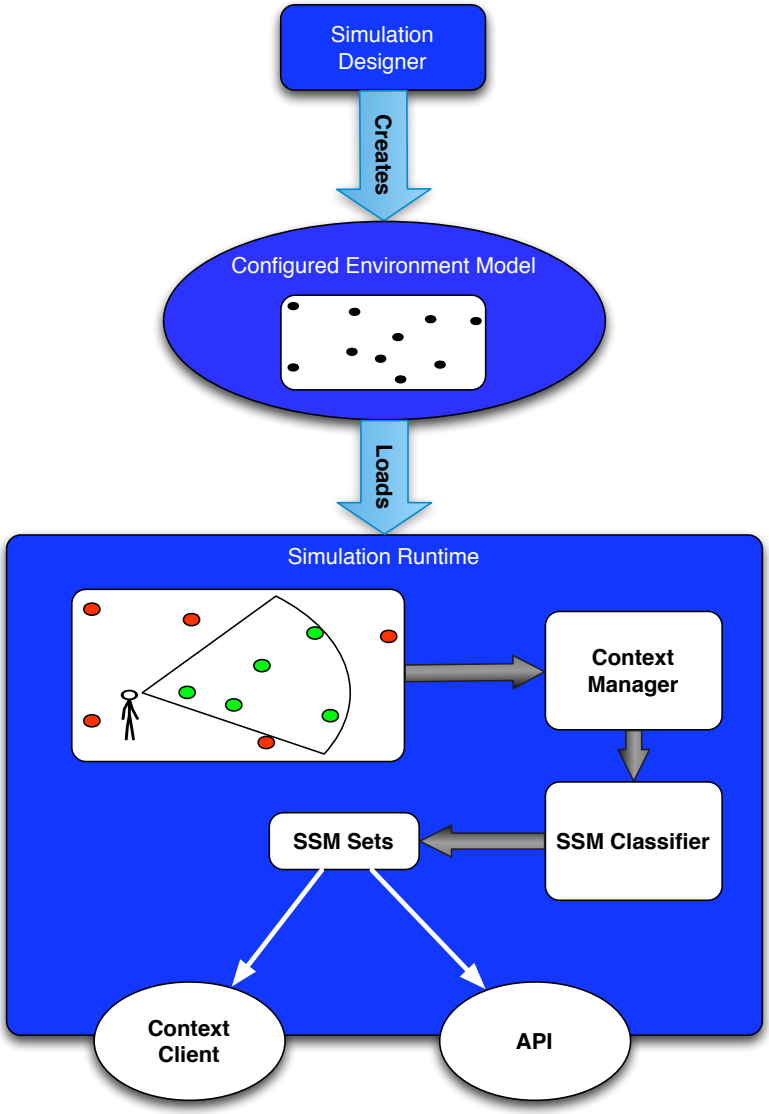


Figure 12: EgoSim: initial architecture diagram

3.4 SIMULATION DESIGNER

Creating the environment model is the first step required by our system. As described in requirements 1. and 2., the simulation designer needs to empower the system designer to create the model of the target environment and to identify and configure the objects to be monitored.

First, we need to figure out what kind of model we are going to use to represent the environment. Similar projects like DiaSim 2.3 have used a 2D representation, while Tatus 2.5 or UbiWise 2.4 have used a 3D representation of the environment. In this context, 2D stands for

two-dimensional while 3D stands for three-dimensional.

Thinking from the agent's perspective, a 2D model can cover two parameters: location and orientation. As mentioned in requirement 4., the simulation user must be able to control the agent's movement (location and orientation) as well as the field of vision. For a realistic field of vision, two more parameters are required: depth and height. The 3D model provides three dimensions that can host all the required parameters. Moreover, the projects which have used a 2D representation were focusing solely on proximity, while the projects aiming for a more realistic interaction have used a 3D model. Such a model can encapsulate more details, proving a better sense of reality as required in 5..

Based on the arguments above, we have decided to use a 3D representation of the environment. Within the model, the system designer has to identify the objects she/he desires to be monitored and classified during the simulation. An entity within the model, visually resembles a certain kind of object (e.g., a chair, a pen, etc), but it is up to the system designer to provided the SSM semantics for that object.

To provide meaning for an object for the upcoming simulation, we are augmenting the objects with a set of properties - *Ego metadata*. Therefore, when detecting the objects, the metadata can be retrieved and further used for monitoring an classification. To prepare the model as such, we have designed a set of properties each objects will need to carry; although all the properties defined in the Ego metadata set are used by the simulation, some can function with default values (for example the default object type is physical), while others must have the values specified by the designer (for example the ID must be uniquely specified by the designer). The first set of properties are meant to provide concrete identity to an object, while the second set of properties represent the SSM configuration parameters. The properties are detailed in the following list of Ego metadata Properties:

ID - uniquely identifies an object within the environment. This parameter is mandatory and it is up to the system designer to ensure that each object's ID is unique.

TYPE - describes the type of the object. We need to support physical objects and mediators 1., so this parameter can have two values: Physical or Mediator. The default value is Physical.

INTERACTION TYPE - specifies the type of iteration the agent can carry out with this object. It can be either Pick-Up or Custom. The

Pick-Up interaction type has to be offered out of the box by the framework, while the Custom interaction type empowers the system designer to design different iterations. The default value is Pick-Up.

IS SURFACE - picked-up objects can be put-down onto surfaces only (e.g. a desk). By default, an object is not a surface.

WEIGHT - how much the objects weights, in Kg.

SSM CONFIGURATION PARAMETERS - data required by the SSM Classifier to correctly classify the objects into SSM sets; made up by a list of four parameters: *PERCEPTION_DISTANCE*, *RECOGNITION_DISTANCE*, *EXAMINATION_DISTANCE* and *ACTION_DISTANCE*.

3.5 SIMULATION RUNTIME

The aim of this component is to enable the simulation user to interact with the 3D model, created using the simulation designer. As observed in the related work, some of the projects used a 3D model to represent the environment (2.2, 2.4, 2.5) and have built the frameworks on various game engines. A game engine represents a software framework designed for the creation and development of video games. Michael Lewis and Jeffrey Jacobson [4] argue for the power of game engines: "The most sophisticated, responsive interactive simulations are now found in the engines built to power games". Moreover, they argue for the usefulness of game engines in scientific research: "There are probably as many potential applications for game engines as there are research problems requiring medium-fidelity 3D simulation or high-fidelity interactive graphics. Our hope is to raise awareness of the high-power/lowcost alternative game engines can offer".

Our design proposal for the simulation runtime is to use a game engine as a starting point. There is wide range of available game engines written in different programming languages, some open-source, others proprietary, most of them providing out of the box components to be reused for specific programming tasks. In conclusion, game engines can fit the various needs and programming experience of the research group designing a simulation for research. The overall design solution we provide based on a game engine is not tied to a certain implementation, but it uses general concepts that can be found in most modern game engines.

To conclude this section, we will briefly discuss the architecture of a game engine and present the most important concepts used in game

engines which are relevant for this work.

3.5.1 *Architecture of a Game Engine*

Analysing the game engine's structure depicted in Figure 13, we can deduct that the game engine renders the virtual environment represented as a 3D or 2D model, while the game code is the custom behaviour for a specific game. Moreover, support for client-server behaviour (usually for multi-player games) can be implemented through the network code module. The fact that the game engine runs at a high-level within an operating system, above the system drivers, makes the client modules platform-independent. The game engine works just like an interpreter, running the same client code (game) on all the supported platforms. Hence, the game engine comes as a solution for requirement 3..

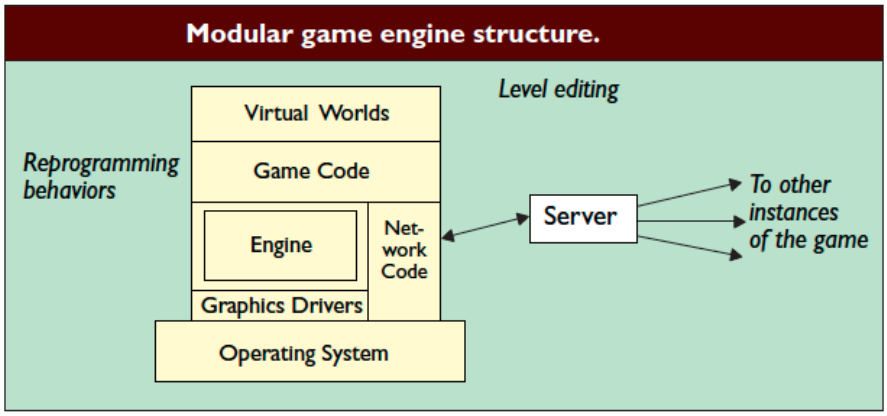


Figure 13: Modular game engine structure [4]

"A game engine includes all elements in Figure 14 that have no effect on actual content, that is, everything indicated by dashed lines plus an event loop" [5]. Therefore, the core features provided by a game engine include a graphics module (rendering engine for 3D graphics), input control (communication with peripherals like mouse, keyboard, joystick, etc), dynamics (e.g. approximate simulation of certain physical systems, such as rigid body dynamics, fluid dynamics, etc), sound. The *event handler*, *game logic* and *level data* (environment model) are client modules that are used to implement a certain type of game / simulation.

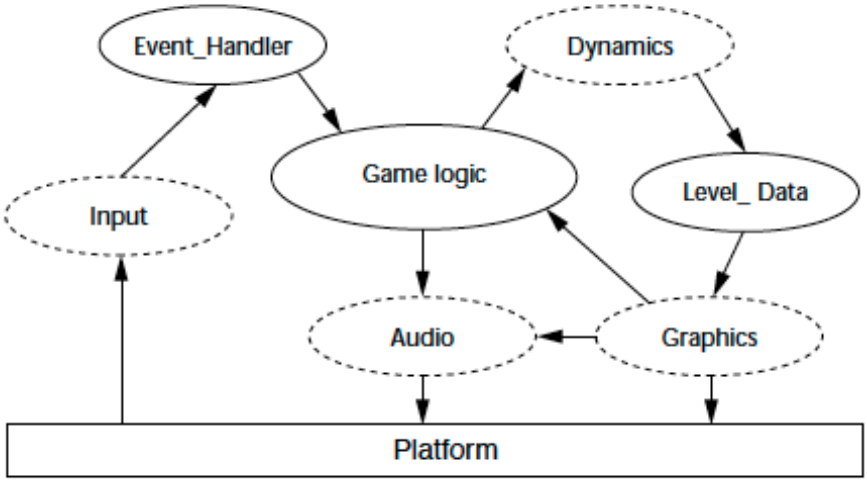


Figure 14: Game engine architecture [5]. The components marked with dashed lines, are contained within the game engine, while the one marked with solid lines (except for the platform) represent client code.

3.5.2 *Game Engine Concepts*

The graphics module (renderer) interprets the environment model as a *scene graph*, depicted in Figure 15. The scene graph is a tree-structured graph, representing the programmatic model of the environment inferred from the simulated environment's virtual model created with the Simulation Designer 3.4. Besides the information required for rendering, some nodes should carry the Ego metadata the system designer might have attached in the design process. The nodes in the scene graph are the entities used in programming tasks carried out within the game logic.

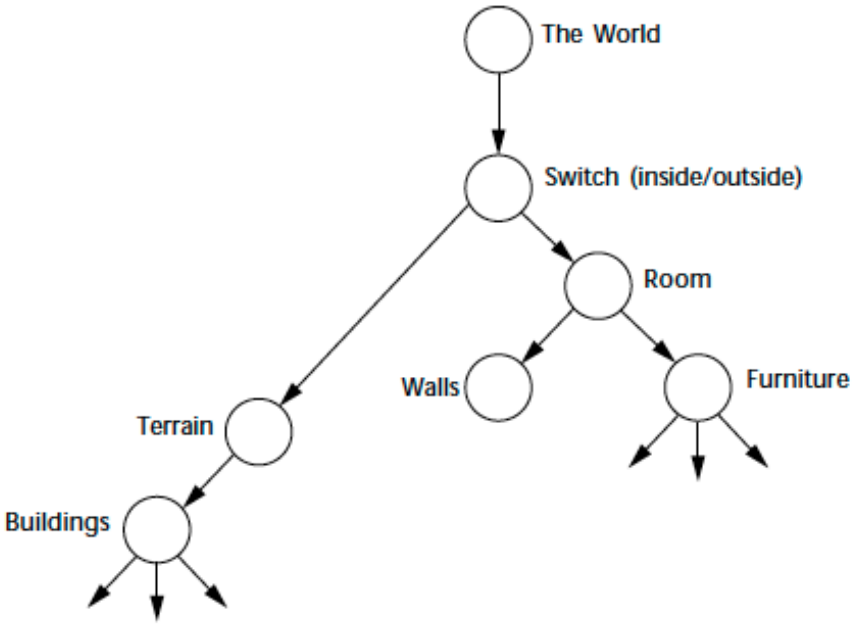


Figure 15: Scene graph [5]

"Missing from Figure 15 is the *camera*, an object not part of the scene graph [...]. The root of a scene graph is attached to a camera when both are initialized"[5]. Using the information of a certain camera (position and orientation) and the scene graph, the renderer computes how to draw the 3D scene graph to the 2D screen. This is what the user sees on the computer screen as the game unfolds. We can think of a camera positioned at a certain point and having a certain direction as the user's field of vision, displaying what the user would actually see if she/he would be located as such. In a game, there can be multiple cameras, providing different perspectives.

The mathematical computations used by the graphics engine are all based on absolute or relative coordinates of the entities. Coordinates represent a location in a coordinate system. 3D game engines use a 3D coordinate system. As illustrated in Figure 16, coordinates are relative to the origin at (0, 0, 0). In 3D space, we need to specify three coordinate values to locate a point: X (right), Y (up), Z (towards us). Similarly, -X (left), -Y (down), -Z (away from us).

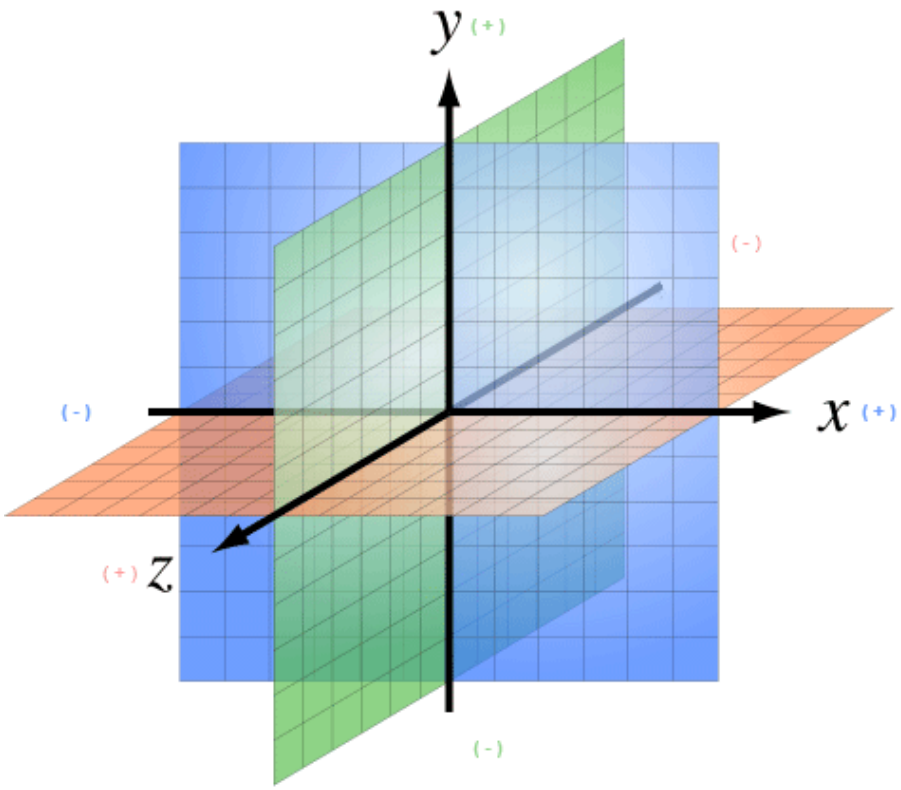


Figure 16: A 3D coordinate system [6]

As mentioned above, the origin is the central point in the 3D world, where the three axes meet, at the coordinates (0, 0, 0). A coordinate represents a location within the given coordinate system. To represent a direction, game engines use the concept of vectors. A *vector* has a length and a direction, just like an arrow in 3D space. A vector starts at a coordinate (x1, y1, z1) or at the origin, and ends at the target coordinate (x2, y2, z2). Backwards directions are expressed with negative values.

With this concrete information, let's take a step back to reflect on the concept of a camera. As we have already mentioned, the camera is made up by a location and a direction. The region of space that appears on the computer screen at a certain point is called the *view frustum*. This is basically the *field of vision* of the active camera and it is illustrated in Figure 17.

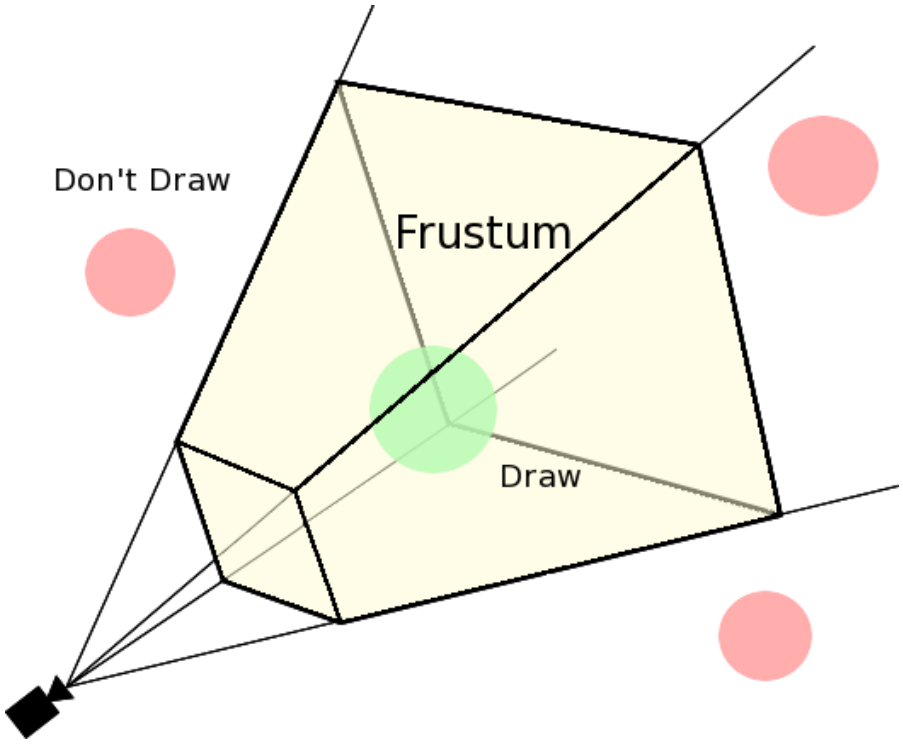


Figure 17: View Frustum or Field of Vision (http://blog.jaxgl.com/2012/06/render-smarter)

So, the camera contains both information about the location and direction of the camera within the 3D world, as well as information about the view frustum of the camera. The game engine uses the camera and the scene graph to determine which objects to draw and which not to. This concept is called *culling*.

Another useful term in game engines is the *bounding volume*. 3D objects can have irregular forms, other that basic 3D geometrical shapes. To aid mathematical computation, they are wrapped in a bounding volume which pragmatically represents the object as a sphere, cube, cylinder, etc. The bounding volume is then used to compute physical interactions (where physical objects collide, push and bump off one another), and non-physical collisions (mathematical intersections).

Finally, the concept of *ray casting* is useful to compute distances. Imagine the ray casts a line starting in a certain point of the 3D space having either a finite or infinite length. This ray can be used to determine what objects in the 3D space it intersected. With this information we can determine distance to certain objects.

3.5.3 *Simulation Runtime Summary*

To summarise, based on the research presented in this section we decided to base the simulation runtime on a game engine. In the last part of this section we have presented the most relevant concepts for our work in the game engines terminology. The scene graph represents a programmatic data structure of the simulated 3D world. The field of vision within the 3D space is provided by the camera based on the view frustum, which is computed using the current vector (location and direction) of the camera. To aid mathematical computations, the 3D representations of objects are wrapped in a bounding volume, regulating their form to basic 3D geometrical shapes. Last, to compute distances the concept of ray casting comes in handy.

The design of the following components will use the game engine components and terminology presented in this section.

3.6 THE AGENT

The agent is the active entity the simulation user can control, using various input mechanisms, in order to interact with the environment. In 3D games and simulations, there are two main graphical perspectives employed to interact with the environment – the first-person and third-person perspectives. These perspectives are the result of positioning the camera in a certain manner, providing a different perception of the environment. To control the agent's position and orientation, the camera's position and direction are coupled to a set of input controls. As the camera is moved and rotated, it creates the impression of interaction with the 3D environment. For a more realistic interaction, various animations are carried out.

3.6.1 *First-Person*

The first-person view is a graphical perspective rendered from the viewpoint of the agent. Figure 18 illustrates the first-person view from a running simulation built with Tatus [20].

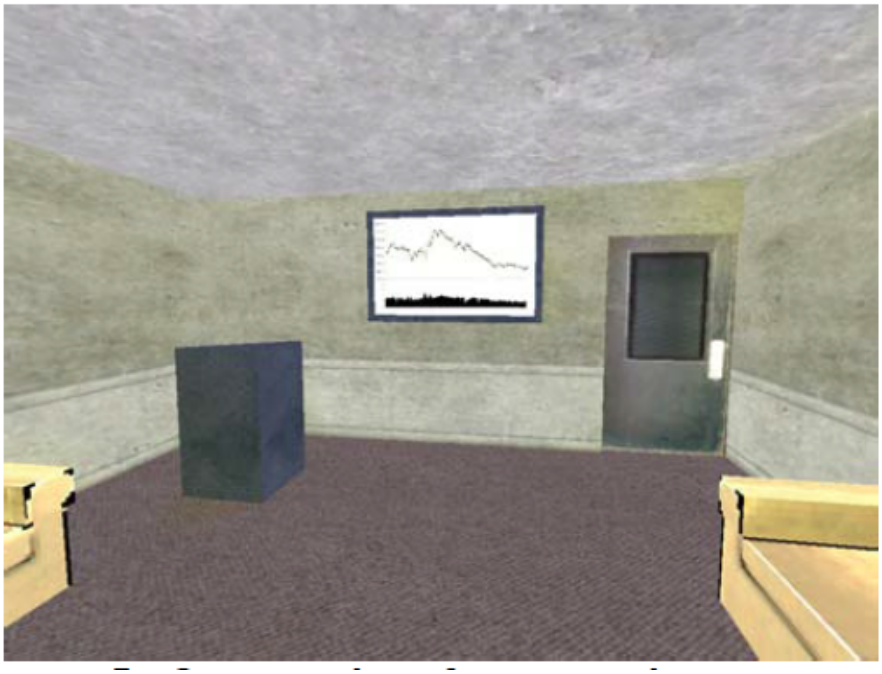


Figure 18: Example of a first-person perspective

Using this perspective, the simulation displays what the simulation user would see with the agent's own eyes, thus objects have to be scaled accurately to appropriate sizes. In this perspective, the simulation users typically cannot see the agent's body, although it might be helpful for the user to see the agent's hands. On the plus side, there is no need for sophisticated animations for the agent, which reduces development time. Also, it helps for easier, more realistic aiming (pointing at objects to interact with) since there is no representation of the avatar to block the simulation user's view. Not seeing the agent's body reduces development efforts, but might represents a drawback as well.

3.6.2 *Third-Person*

The third-person view is a graphical perspective rendered from a fixed distance behind and slightly above the virtual agent. The third-person perspective can provide an animated, strong characterized agent, directing the simulation agent's attention as she/he were watching a film. Figure 19 illustrates and example of a third-person from the game Second Life[3].

3 http://secondlife.com

Figure 19: Example of a third-person perspective

On the plus side, the third-person perspective allows the simulation user to see the area surrounding the agent more clearly. This viewpoint facilitates more interaction between the virtual agent and their surrounding environment.

As a drawback, the third-person perspective can interfere with accurately pointing at objects, as the agent's body might block the simulation user's view. Moreover, implementing this perspective takes a lot more effort as there is a need for an animated agent body and for custom code to make the camera follow the agent within the simulation.

3.6.3 *Discussion*

Based on the preceding review of existing perspectives, we have decided to include in our design a first-person perspective for the agent. An argument for choosing this perspective is that a first-person view provides the simulation user with greater immersion into the simulated environment.

Avatar based games and simulations usually provide both views, with an easy way to switch perspectives during runtime. For this project, a third-person perspective could be useful for future work if we are to represent wearable devices. Also, it enables the simulation

user to observe the various body gestures the agent is doing.

To fully support requirement 4., we need to define a way of controlling the agent. Most games and simulations use a combination of the keyboard and the mouse to control the agent. The standard control combination is described bellow:

- moving *the mouse* controls the agent's direction; hence mouse movements translate into rotating the agent's viewpoint within the environment. This control is used to look around and to point at objects the simulation user desires to interact with
- clicking *the left mouse button* triggers an interaction. The interaction is contextual; for example, clicking on a switch will turn on or off the switch. If the user clicks on a pen, the agent would pick the pen up, while if the user clicks on a surface while the agent is carrying a pen, the agent will put the pen onto the surface. Of course, the ability to carry out these actions depends on the distance between the object and the agent
- moving the agent is done by activating a set of four keys on the keyboard. We have chosen the standard W, A, S, D key combination. Each key moves the agent towards a certain direction, Therefore, W - forward, A - left, D - right, S - backward.

To help the simulation agent with the task of accurately aiming at objects, a cross icon will always be present in the middle of the screen. The object currently pointed at will represent the object the user intends to interact with.

In order to determine the object to interact with, we use the concept of ray casting. To achieve this, we cast a ray of infinite length from the current location of the camera along it's direction. This ray passes right through the middle of the cross and intersects all the objects along the way. We then determine the object to interact with by picking the closest object to the agent; that is the first object intersected by the ray. Moreover, the object can be interacted with only if it carries Ego metadata, otherwise the interaction is not possible. We have imposed this constraint because without the metadata we cannot take concrete decision on what to do with the object.

Whether an object can be interacted with or not, is given by the current distance towards that object. Therefore, the distance has to be at most equal to the value of the ACTION_DISTANCE property in the metadata.

Finally, the agent should not be able to pass through walls and other physical objects. Game engines provide various mechanisms

for collision detection that can be used in the implementation to accommodate this aspect.

3.7 MONITORING SERVICE

The purpose of the monitoring service component is to keep track of the objects marked with Ego metadata and classify them into SSM sets, as the simulation user controls the virtual agent to interact with the surrounding environment. In order to classify the objects, the service needs the following data: the agent's location and agent's orientation, the objects within the agent's field of vision and the Ego metadata of the monitored objects.

Next, we will briefly address how to determine whether objects are within the agent's filed of vision or not, followed by separate discussions on how the framework categorizes the objects into various SSM sets. The sets have been defined in Section 3.1.4.

3.7.1 *World Space*

Central to all computations and decisions in this framework is the World Space. It is ought to contain all the objects in the environment carrying Ego metadata. To design for easy access to the objects carrying Ego metadata, when the simulation is started up, we parse the scene graph and each object carrying Ego metadata is added to the World Space. Whenever we need to categorize the monitored objects, we can simply refer to the World Space.

The drawback of this choice is that we are duplicating a set of data already present in the scene graph. The advantage is that we don't have to parse to scene graph every time we are interested in the objects carrying Ego metadata; accessing them from a reduced set of data brings performance improvements. Moreover, the World Space has to always be available through the API and it never changes, so computing it once, at the simulation's start-up, comes as a natural decision.

3.7.2 *Objects within field of vision*

To determine the objects within the agent's field of vision, we can rely on the agent's view frustum and the World Space 3.7.1. The frustum describes six planes: top, bottom, left, right, near and far. If the object is fully contained within or partially intersects the view frustum, we can consider the object is within the agent's field of vision. This is the

same algorithm game engines use to determine whether to render or not a certain object (a process called culling).

The solution above does leave us one problem though: occluded objects will be considered visible. An object is considered to be occluded if other objects are in front of it blocking the agent's visibility over the target object.

To determine if an object is occluded, we have used ray casting. We are casting a ray from the agent's location (the agent's position vector) to the centre of the object (the object's position vector). If the ray intersects any other object before the target object, we consider it as being occluded, therefore not in the agent's field of vision.

This approach should work most of the time, but there are some edge cases with false positives. For example, a small object could be positioned in front of a really large object. Even if the centre point is occluded, the agent should still be able to perceive the object as it is large enough to recognize it based on all the details it provides. To overcome these kind of edge cases, we could cast a large number of rays to various points on the object and based on a more advanced algorithm, determine if the object is being occluded. We have decided to leave the design of a more advanced computation for future work, as the current solution is good enough to cover most of the cases, while developing a more advanced and correct algorithm needs more research in the field of 3D modelling and rendering.

3.7.3 *Categorisation into SSM spaces*

We have addressed the World Space 3.7.1 separately as most of the logic of the monitoring service revolves around it. We have addressed the reset of the SSM spaces in this section.

To determine whether an object is part of a particular SSM space or not, we need to know the distance to that object. The distance can be computed knowing the intersection of a ray cast from the agent's location to the target object. To avoid doing the same thing twice, this computation is part of the process where we determine if the object is within the agent's field of vision, as discussed in Section 3.7.2. There, we set a the distance in the Ego metadata as the *LAST_MEASURED_DISTANCE* property.

Computing the distance as described above, brings up the same discussion we had in the previous Section 3.7.2 about the occlusion of objects. It is based on the same algorithm so we compute the dis-

tance to the centre of the object. This can be improve in future work, along with the algorithm for detection of object occlusion.

Having all the necessary parameters, Perception Space, Recognition Set, Examinable Set and Action Space are computed as described in Section 3.1.4. We have used a less generic definition of these spaces than originally defined in [3], but future work could broaden them up to provide a classification closer to the ones defined in the theoretical framework. For example, an object in the Perception Space can be perceived as different entities depending on the distance to the agent. Given a hammer, from the furthest perception distance (X1) the agent would perceive it as an object; from a closer distance (X2), the agent would perceive it as an object certain kind of shape while from a more closer distance (X3), the agent would perceive it as hammer. So, an object should be perceived in different ways based on the agent's proximity. Such improvements to the classification into the aforementioned sets, are target of the future work.

The Selected Set contains objects currently being physically manipulated. This happens when the agent picks an object up. In order for the agent to pick up a certain object, she/he must be targeting it (aim at it with the on screen arrow) and click the Left Mouse Button (pick up action); also, the object must be in the Action Space.

In the current design we only support pick-up/put-down as interaction with objects in the surrounding environment. Therefore, the only manipulation the agent is empowered to do is to pick objects up and move them around. That is why in the current work, the Manipulated Set completely overlaps the Selected Set.

Categorisation is triggered by the agent movements. Hence, every time the agent is moved or is looking around, using the controls described in Section 3.6, a new categorisation is triggered and the SSM sets are refreshed.

3.7.4 *Monitoring Service Summary*

To summarize, in this section we described the design for object classification into SSM sets. It comes as a solution to requirement 6., classifying objects around the agent as the agent's field of vision changes and as the agent interacts with the environment.

3.8 THE API

In this section we will present the design for an API, accessible by third party services. As discussed in requirement 7., the API should provides real-time access to the SSM sets. The nature of third party service is that it is entirely decouple from the application providing the API, it is highly probable to be written in another programming language and could be running on any another software platform. To accommodate all these aspects we need to provide a loosely coupled, highly interoperable API, one that does not constrain the client service to certain programming languages or platforms.

An example of a system that successfully achieved a high degree of interoperability, through a fixed interface, is the Web [23]. In [24] the authors manage to point out the fast growth of the Web towards an ubiquitous environment. They propose an architectural style for this distributed, volatile environment: the REpresentational State Transfer (REST). It is a set of principles that, when correctly applied, help building software architectures and applications that benefit from all the qualities of the Web. Those qualities include greater scalability, efficient network usage and independent evolution of clients and servers – also called loose coupling.

This led us to adopt the REST architectural style and integrate it in EgoSim's interoperation mechanism. The entities used to communicate over REST are represented in the JavaScript Object Notation (JSON) encoding. This design choice ensures loose coupling and a reliable communication between components, making EgoSim easily extensible and opened for communication even with services written in other programming languages.

The JSON format for data transfer is a lightweight data-interchange format, it is easy for humans to read and write and it is easy for machines to parse and generate. We have decided to use JSON over other existing data-interchange formats as it is one of the most lightweight types of structured data. For example, the Extensible Markup Language (XML) is another type of data-interchange format, but it is often criticised for its verbosity and complexity.

Therefore, the content of the sets will be available through the RESTful API in the JSON format. As the SSM sets are made up by the monitored physical objects and mediators, the sets will contain the JSON representation of the Ego metadata (including the LAST_MEASURED_DISTANCE) property.

3.8.1 *RESTful API*

REST is an architecture style for distributed hypermedia systems. A web service is an API which is accessed through the HyperText Transfer Protocol (HTTP) and executed on a remote system, hosting the requested service. A RESTful web service is a web service implemented using HTTP and the principles of REST. The RESTful web service is defined by a collection of resource, each of which is defined by three main characteristics:

- the base URI identifying the web service
- the Multipurpose Internet Mail Extensions (MIME) type of the data supported by the web service (JSON, XML, etc)
- the web service's interface defined against the HTTP supported methods like POST, GET, PUT, DELETE etc

The REST architectural style imposes a client-server architecture, which fits well the requirement for a decoupled design we are aiming for.

3.8.2 *Discussion*

One of the drawbacks of the RESTful approach is that the third party service has to access it every time it needs information. So, if a service would like to take an action in real-time, it would need to access the RESTful API quite often. Even so, there will always be a *latency problem* – the client service will not receive a change in the SSM sets the instance it occurs. We have found this drawback acceptable for the current work. A service could be able to access the REST service every second which might provide a close to real-time access to contextual changes. At the moment we want to explore how third party service would use the data provided by the API and how they would build business logic around it. Based on this information, in future work we can improve the API to provide client services with real-time access to changes in the SSM sets.

3.9 THE CONTEXT CLIENT

To make it easy for the simulation user to follow the evolution of the SSM Sets as the simulation unfolds and the agent interacts with the environment, as discussed in requirement 8., we have designed the Context Client as a service working on top of the API 3.8. It is really just a nicely formatted visualization for easy understanding of what's going on under the hood. To provide a close to real-time visualisation,

the context client will be accessing the API every second to display an updated visualization of the SSM Sets. In future work, the context client is meant to be improved alongside the API.

3.10 THE DESIGN

To summarize, we present an overview of our design. We will represent the target environment as a 3D virtual model 3.4. We will build a simulation runtime based on a game engine 3.5. To empower the simulation user to interact with the environment, we will build an agent based on a first-person perspective. The agent will be controlled using standard game controls as described in Section 3.6. As the agent is interacting with the environment the monitoring service 3.7 will classify objects around the agent according to the Situative Space Model (SSM). The content of the SSM sets will be available for third party services through a RESTful API 3.8 and for the simulation user, in a visual representation, through the context client 3.9. Figure 20 show a visual overview of this design. It is a more concrete design than the one presented in the begging of this chapter 12. We have modified the initial design to fit the architecture of a game engine illustrated in Figure 14.

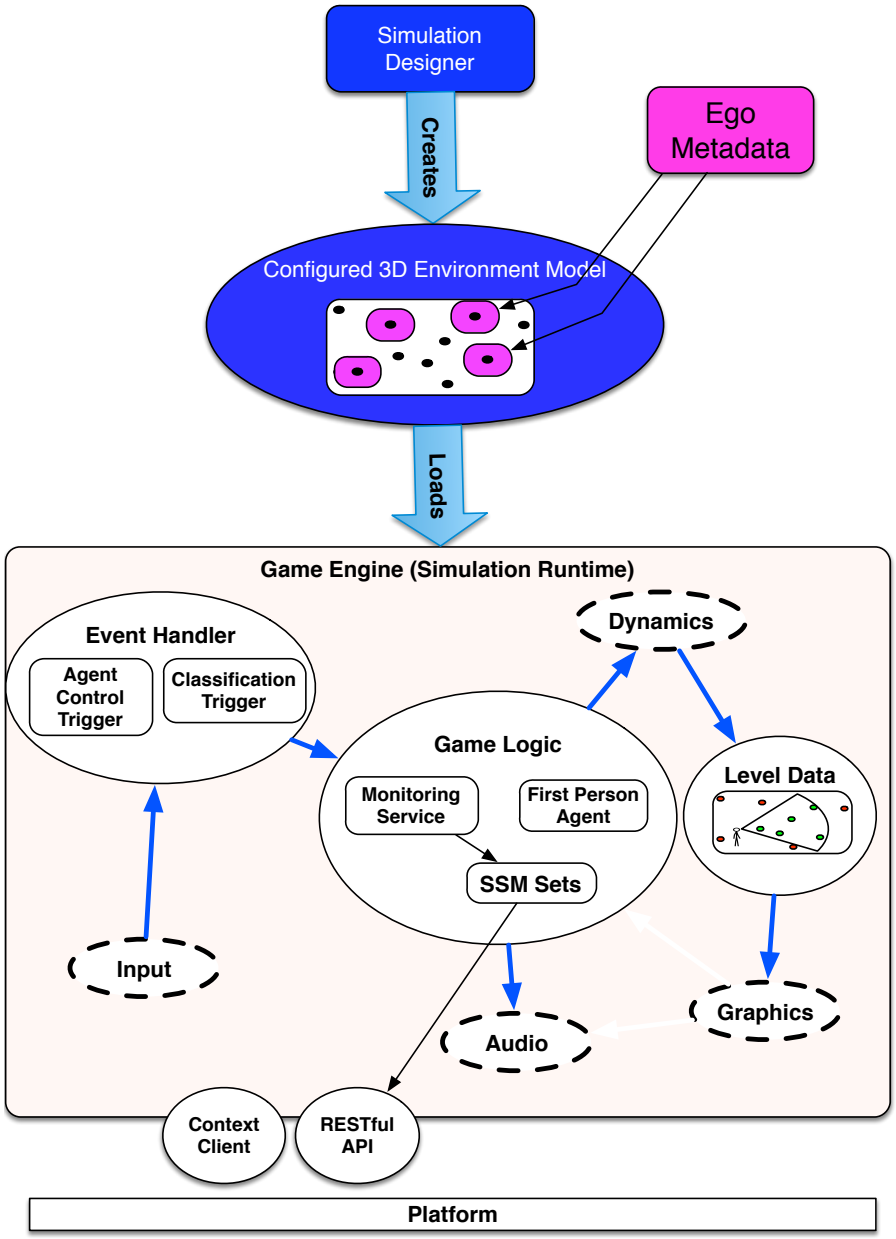


Figure 20: EgoSim: A visual overview of our final architecture

4

IMPLEMENTATION

This chapter describes the choices made during the implementation of the design defined in Chapter 3. Our system's architecture was tailored to fit within the architecture of a game engine, each individual component being tied to certain concepts from the game engines terminology.

In Section 3.5 we have argued for the benefits of using a game engine as the building block for the EgoSim framework. There are many available game engines which could fit our design, just to mention a few:

- Source[1], Half Life's game engine, was used to build the TATUS 2.5 ubiquitous computing simulator
- Quake III Arena's game engine from ID Software[2], was used to build the UbiWise 2.4 device-centric simulator
- jMonkey Engine version 2.0 [15], was used to build the simulation in the SIMACT 2.2 smart home infrastructure simulator
- Unity[3] is a new cross-platform game engine, having a large number of game developers adopting it as a platform of choice when developing a game.

Previous to starting this project we had no knowledge of game development and game engines whatsoever. We have ran across Source, Quake III Arena and jMonkey Engine (JME) during research of related work, while we have heard of Unity due to all the buzz that formed around it lately. Source, Quake III Arena and Unity require strong knowledge of game development and of the game engine at hand in order to become proactive. JME on the other hand has been implemented as a "3D game engine for adventurous Java developers". Moreover, "it's open source, cross platform and cutting edge. And it is all beautifully documented" [15]. One can become proactive in JME while learning it.

We have decided to use jMonkey Engine in the EgoSim's framework implementation. It is tailored for experience Java developers in need to perform complex 3D interactive simulation tasks. It has a

1 https://developer.valvesoftware.com/wiki/Source
2 http://www.idsoftware.com/gate.php
3 http://unity3d.com/

well written documentation and has formed a big and helpful community around it, which is really important to help developers get up to speed with game development. The community helped us to find answers to various problems we have encountered. Finally, it is open source, helping us achieve this aspect for our system as well, as described in goal 4..

SIMACT 2.2 has used an old version of the jMonkey Engine (v 2.0). This version has stopped being developed and supported in 2008. In 2009 the community started up with new ideas, rewriting the framework from scratch, resulting on the current jMonkey Engine version 3.0.

JME is a community-centric open source project, especially built for modern 3D development. It is written purely in the Java programming language and it uses the Lightweight Java Game Library (LWJGL)[4] as its default renderer. JME by itself is a collection of libraries, but to provide an integrated development environment, the community made available the JME SDK/platform. The SDK is based on the NetBeans Platform[5], providing all the necessary tools needed to develop a game in JME. It is designed to be easily used by Java developers to build games/simulation without previous game development knowledge. The learning curve is easy, compared to other game engines, teaching the developer both to build games and to learn the concepts and terminology of game development.

However, the design we came up with in Chapter 3 is universally applicable within most available modern 3D game engines. Implementing this design using various game engines should not pose a challenge, as long as the developers are familiar with the game engine of their choice.

Figure 21 illustrates the project's structure within the JME SDK. "EgoSim" represents a Java project with JME dependencies and capabilities. The items highlighted in screenshot are as follows:

1. The Project Assets holds resources to be used in the simulation. For example, the Scenes folder contains the environment models, the Sounds folder contains the sounds which can be played in a simulation, etc.

2. The Source Package folder contains the source code of the EgoSim framework.

3. The Libraries folder contains the Java ARchive (JAR) dependencies of the EgoSim project. These are both JME library de-

4 http://lwjgl.org
5 https://netbeans.org/features/platform

pendencies as well as other third party libraries we have used throughout the development process.

4. The Run button. Runs the current simulation.

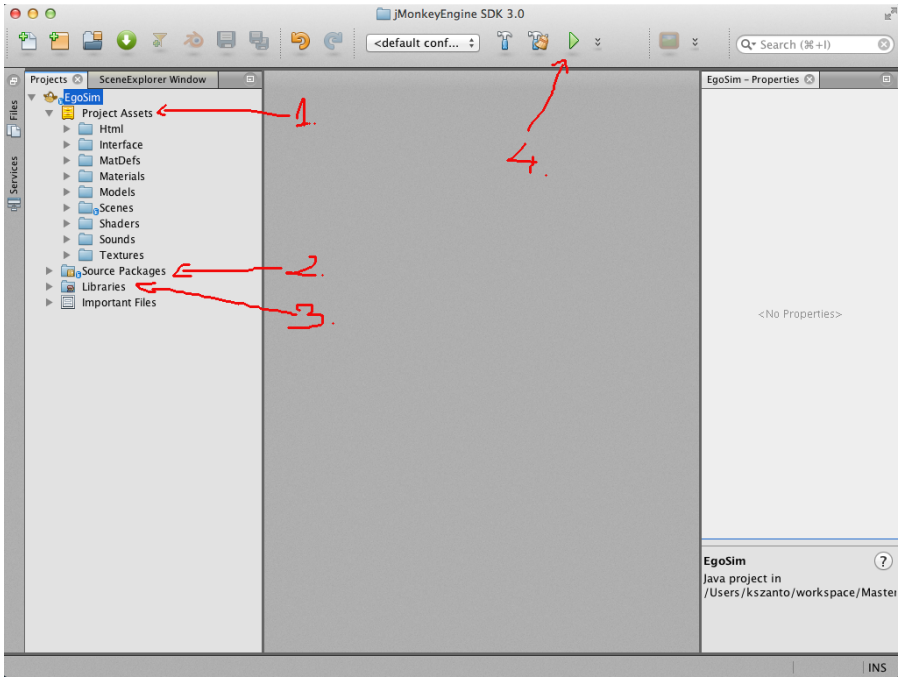


Figure 21: EgoSim: The project's structure in the jMonkey Engine SDK

To get up to speed with jMonkey Engine development, you can access the following resource:

- read the official documentation web page `http://hub.jmonkeyengine.org/wiki/doku.php/documentation` to get up to speed with JME concepts
- ask questions or search answer on the JME community forum `http://hub.jmonkeyengine.org/forum`
- consult the JME JavaDoc `http://hub.jmonkeyengine.org/javadoc`
- read the JME Beginner's Guide book [25]

In this section we will go through the implementation of each component presented in the design Chapter 3, in the exact same order. We will first describe the Simulation Designer 4.1 and how it can be used to work with the 3D models JME is compatible with and how to configure the objects with Ego Metadata. Next, we will present the Simulation Runtime 4.2 and how it helps us in bringing those 3D models to "life". We continue with the rest of the components : The Agent 4.3, Monitoring Service 4.4, The API 4.7 and finally the Context Client 4.8.

The source code of the framework is available; see Appendix C. Using the framework to set up a new simulation is described in details in Appendix A.

4.1 SIMULATION DESIGNER

As discussed in Section 3.4, the Simulation Designer helps the system designer augment the objects she/he wants categorised into SSM Sets during the simulation. It helps to identify and configure these objects with Ego Metadata.

First, we present the 3D model formats JME is compatible with. As defined in JME's feature description[6], JME is compatible with the following 3D model formats:

- *.blend* Blender[7] is the most popular open source 3D modelling application. Most major model formats can be imported into Blender, which in turn can be saved as a .blend and used in jME3.
- *.ogrexml* OGRE3D is a game rendering engine with a very stable asset pipeline. OgreXML exporter plugins have been written for all major modelling applications, including 3DS Max[8], Maya[9], Softimage[10], and Blender[11].
- *.obj* The Wavefront OBJ format is a widespread alternative export format. It doesn't support animations which makes for less complications when dealing with static assets.

As it is the most popular open source 3D modelling application, in this project we have used environment models build in Blender. They are discussed in Section 4.9. The JME SDK comes with out of the box tooling to import blender models and a scene composer to modify to modify the model. Importing and working with blender models into JME is briefly covered in Appendix A.2.

A scene is a complete model of an environment. The scene is usually made up by other object models, including terrain, walls, everyday physical objects, etc. JME provides a data structure (i.e. a map) called *user data*. This map holds instances of objects mapped to unique String identifiers. Therefore, using this mechanism, we can identify and augment object with Ego Metadata. When trying to attach an

6 http://jmonkeyengine.org/features/asset-pipeline/
7 http://www.blender.org/
8 http://www.autodesk.com/products/autodesk-3ds-max/overview
9 http://www.autodesk.com/products/autodesk-maya/overview
10 http://www.autodesk.com/products/autodesk-softimage/overview
11 http://www.blender.org/

instance of the EgocentricContextData to a certain model, the scene composer, using reflection, builds up a visual form, as illustrated in Figure 46, with all properties of the class allowing the system designer to easily configure the metadata properties.

To model the Ego metadata Properties we have implemented a class called EgocentricContextData. It contains all the properties defined in the design of the metadata. Moreover, in order to add an object in the user data map, it must be *Savable*. This means it must implement the Savable interface from the JME library. This enables the object to be serialized and de-serialized. This is actually how it ends up from the static model into the simulator. The data is saved in a file along with the 3D model. When the model is loaded in the Simulation Runtime, the user data is also read from the file it was saved in and attached to the corresponding objects.

The class diagram in Figure 22 illustrates how the EgocentricContextData is tied to the JME library. The class implements all the attributes we have described in the design of the Ego Metadata. When serialized and de-serialized, the attributes take meaningful default values. Please inspect the source code for the default value each attribute has.

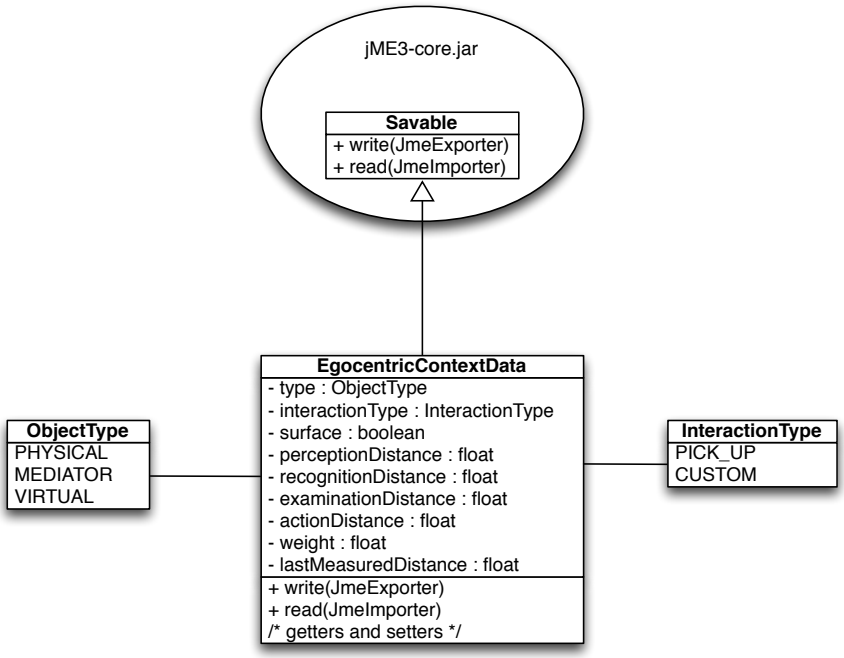


Figure 22: Egocentric Context Data Class Diagram

In JME distances are expressed in World Units (WU). This is a subjective interpretation of the distance in real-world metrics, the system

designer should decide upon at the time of building the environment.

Although we have implemented support to interact with physical objects and devices, we have designed to support virtual objects as well (see the ObjectType enumeration). Moreover, we have designed to support other types of interactions (CUSTOM), except the current pick-up/put-down interaction type (see the InteractionType).

When adding EgocentricContextData to an object, always use the "EGOCENTRIC_CONTEXT_DATA" as the identifier in the user data map. The other components of this framework take into account objects carrying data under this tag. Identifying and augmenting objects to be monitored is covered in Appendix A.4.

To conclude, Figure 23 illustrates a slightly modified design of the simulation designer in the JME context. It is the same design as presented in 20, but using components provided by the JME SDK. We have managed to entirely base our configuration process on JME components. Therefore, using the JME Scene Composer from the SDK, the system designer can load a list of supported 3D model formats. Once loaded, the model can be converted to the .j30 format, internal to JME. In the .j30 model the system designer can visually identify the objects he wants monitored augmenting them with the EgocentricContexData information.

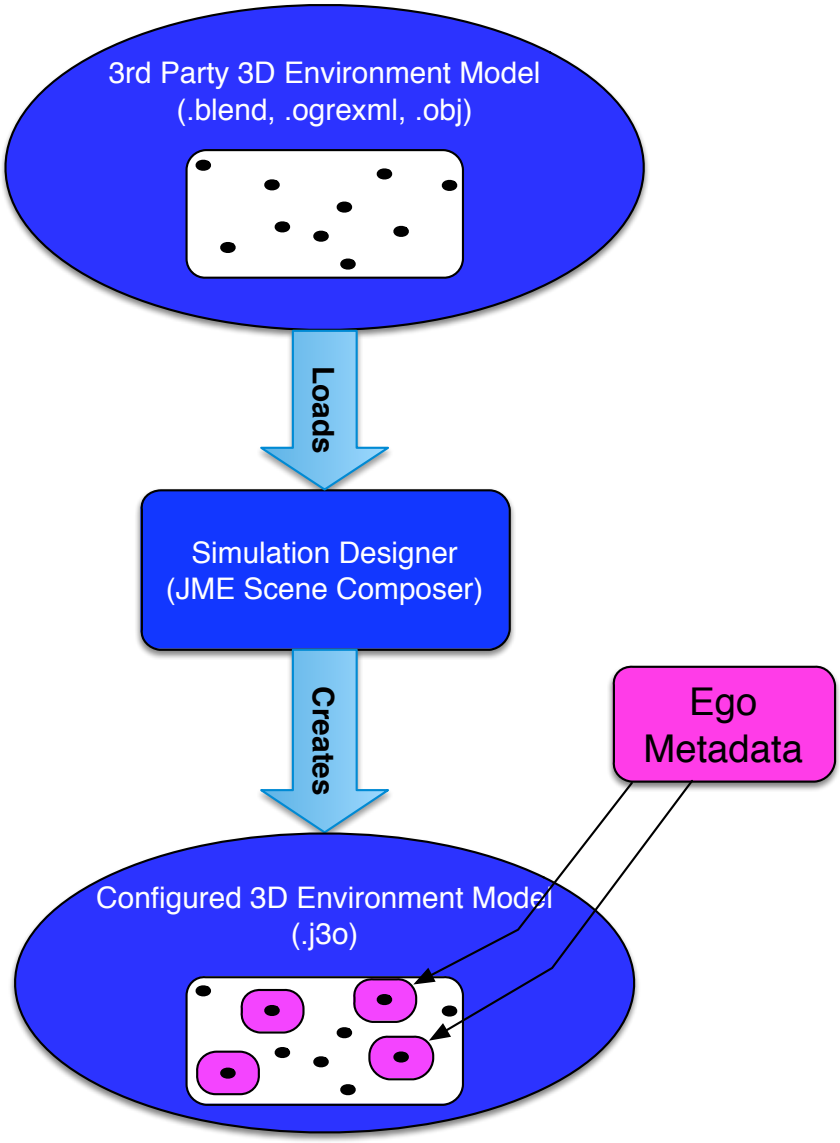


Figure 23: Simulation Designer in the jMonkey Engine context

4.2 SIMULATION RUNTIME

As depicted in Figure 20, the Simulation Runtime component is responsible for loading up the 3D model of the target environment and to initialize some of the components: the agent 4.3 and the monitoring service 4.4. It also represents the entry point of the simulation. In Java terminology this means that this is the main class of the application.

Each simulation a system designer would set up, needs to follow the exact same series of tasks. Hence, this component represents an abstraction which must be extended by each particular simulation.

Normally, the particular simulation should only provide the loaded scene and the abstraction takes care of the rest. We have implemented a few prototype simulations described in Section 4.9. These prototype simulations have been used during the evaluation process.

The diagram in Figure 24 illustrates the EgocentricApp abstract class and how it relates to some classes in the JME library. This abstraction provides the functionality needed for a concrete simulation. The diagram highlights only a few of its properties and methods.

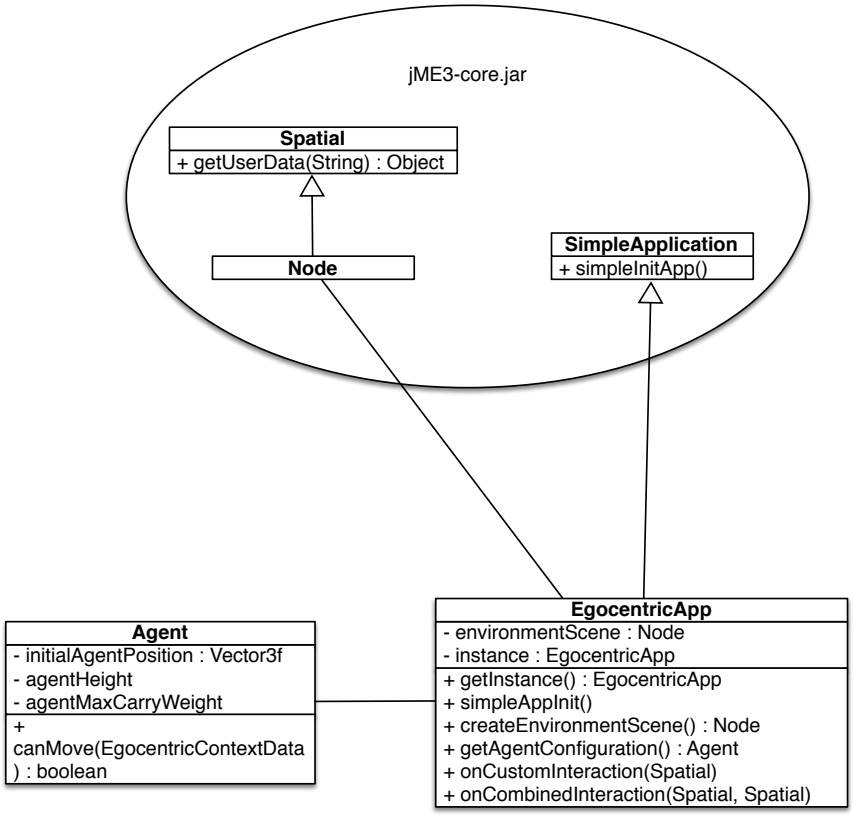


Figure 24: EgocentricApp Class Diagram

The EgocentricApp subclasses SimpleApplication from the core JME library. SimpleApplication gives us access to standard game features, such as a scene graph (or the rootNode), an asset manager (to import textures, sounds, etc), a user interface (guiNode), input manager, audio manager, and many more. We have covered most of these concepts in Sections 3.5.1 and 3.5.2. The components we have access to through SimpleApplication are implementations of these generic concepts particular to JME. To star or stop the simulation, SimpleApplication provides the *start()* and *stop()* methods.

In the diagram, we have depicted the *simpleInitApp()* method. This is called when the simulation is initialized. This is where EgocentricApp initializes of the simulation as follows:

1. calls the *createEnvironmentScene()*. This is an abstract method and must be implemented in all the subclasses. It must return an instance of a Node, meaning the subclass is not constrained to load the a certain type of 3D model. It can load any of the supported formats or even build the model pragmatically.

2. it adds a collision control component to the environment scene. This will handle automatic collision detection with other collidable entities (i.e. the agent).

3. add the environment scene to the guiNode. The environmentScene is just an in-memory data structure. To make it visible, it must be added to the rendered tree of nodes.

4. start up the monitoring service

5. create the first person agent using the Agent configuration data

Besides the createEnvironmentScene() method, all the other methods are implemented with a default behaviour. For example the getAgentConfiguration() method, provides various configuration parameters needed by the first person agent. More on the agent, in Section 4.3.

The EgocentricApp provide two callback methods:

- onCustomInteraction(Spatial) is called when the agent tries to interact with an object carrying Ego Metadata with a custom interactioType. The object is received as a parameter. This method can be overridden and a concrete simulation can implement various custom behaviours. For example, interacting with a door, should open it.

- onCombinedInteraction(Spatial, Spatial) is called when the agent tries to interact with an object while carrying a picked up object. For example, interacting with a notebook while carrying a pen, could pop up a window so the simulation user can easily input text.

Spatials are the elements of the 3D scene graph. As depicted in the class diagram they provide the *getUserData(String)* method. With the help of this method we can retrieve the Ego Metadata using the EGOCENTRIC_CONTEXT_DATA flag. In the callback methods presented above, the received spatials will always carry Ego Metadata. Otherwise, the callback methods will not be called in the first place.

The simulation logic is spread among the components which we will present next. The EgocentricApp is mean to initialize these components and provide a generic entry point for the concrete simulations extending it. To provide easy access to various game assets and the callback methods for the rest of the components, EgocentricApp is implemented as Singleton [17].

A few concrete simulation examples will be discussed in Section 4.9.

4.3 THE AGENT

As described in Section 3.6, the Agent component enables the simulation user to control the virtual agent in order to move around and interact with the environment. Moreover, it triggers the classification event and the interaction callback methods on the EgocentricApp. The agent component implements three components presented in the design 20: First Person Agent, Agent Control Trigger and Classification Trigger.

The First Person Agent component is implemented as a standalone class, having at its core the Camera and ActionListener classes from the JME3 core library. Figure 25 depicts the class diagram of the first person agent.

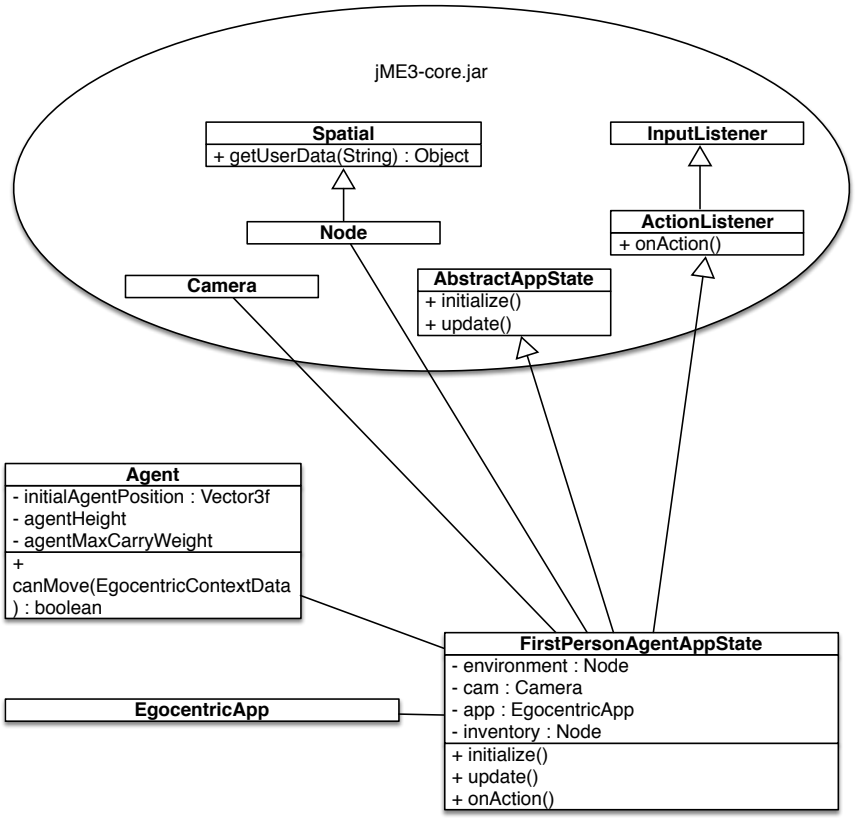


Figure 25: First Person Agent Class Diagram

To implement the requirements for the agent 4., we have isolated the implementation in an AbstractAppState, which is a jME3 abstraction that allows you to control the global game logic and the overall game mechanics. This allows to create a modular implementation of the framework, instead of having all the game logic related functionality implemented in the EgocentricApp. Just like SimpleApplication, AbstractAppState provides an initialization method which is called once, after the scene graph has been initialized and an update method, which is called on every render step of the rendering engine. The number of render steps of the rendering engine is measured in Frames Per Second (FPS) and it depends on each machines graphical processing capabilities.

4.3.1 *First Person Agent*

The first person agent is a wrapper around the Camera which is placed within the rendered 3D environment. The camera is initially positioned at the initialAgentPosition and height given by the Agent class. Also, agentMaxCarryWeight is worth mentioning – this has impact when the agent tries to pick up an object, which is possible if the

object's weight is less or equal to agentMaxCarryWeight.

To help targeting objects, we have placed a cross sign in the middle of the screen which is always visible. Figure 26 depicts a screen shot of the agent targeting the Pot object on the Stove with the help of the on-screen CrossHair.

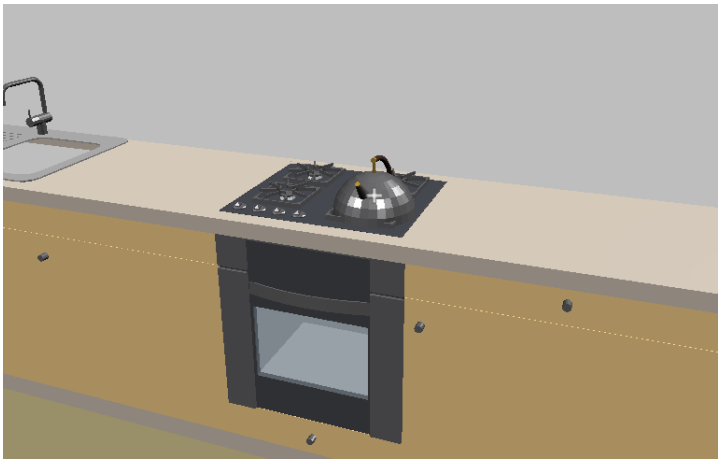


Figure 26: A cross hair always visible in the middle of the screen to help targeting objects

To represent picked-up objects, we have implemented an inventory. As illustrated in Figure 27, the inventory is a miniature representation of the picked up object and moves around together with the agent.

Figure 27: Representation of a picked up object

To allow the game engine to automatically handle collision between the agent and the rendered 3D environment, we have used a the CharacterControl entity from the core library. The control contains a cylinder shape which is placed at the initial agent position and is moved together with the agent. This shape entity is not visually rendered.

The character control together with the collision control component the scene was wrapped in by the simulation runtime, prevent the agent from passing through the rigid areas of the environment (walls, objects, etc).

Controlling the agent and interacting with the environment is detailed in Section A.5.1 of the user guide.

4.3.2 *Agent Control Trigger*

By implementing the ActionListener interface from the JME core library, we receive onAction() callbacks from the game engine whenever an input event is triggered (e.g. the mouse is moved, a key on the keyboard is pressed, etc). In the onAction() method we are listening for key presses (W, A, S, D), mouse movements and mouse clicks (left click, right click).

As already mentioned before, the view the agent has is represented by the Camera which is positioned within the 3D environment by means of a location vector and direction vector. The key presses influence the location of the camera, while the mouse movements influence its direction.

When the left mouse button is clicked, it triggers an interaction event with the environment. To determine the object the agent tries to interact with, we cast a ray of infinite length from the camera's current position along the it's direction. We intersect this ray with the *environmentScene* stored in the EgocentricApp singleton. The intersection results in a list of Spatials which intersected with the ray. We sort this list by proximity and get the closest Spatial, which is the first object displayed on screen pointed at by the cross hair. Next, if the spatial doesn't carry EgocentricContextData, the action is dismissed and warning message is displayed on screen ("Interaction possible only with egocentric entities"). Otherwise, we retrieve the EgocentricContextData as *contextData* and the adequate action is taken based on the current context. The context based decision flow can be analysed in the source code going through the FirstPersonAgentAppState.interactListener.

4.3.3 *Classification Trigger*

To start the classification process on the monitoring service, we have implemented another ActionListener, where we listen for any action that can change the field of vision of the agent. That is key presses

(W, A, S, D) and mouse movements. If any of these actions is carried out, we trigger a new classification on the Monitoring Service 4.4.

4.4 MONITORING SERVICE

The monitoring service is meant to categorise objects in the agent's surroundings into SSM Sets. Moreover, it initializes the API 4.7 and the context client 4.8 components. Following the design we have established in Section 3.7, the monitoring service is implemented into several classes as depicted in Figure 28.

The EgocentricContextManager extends the AbstractAppState class in the JME core library. On *initialize()* it goes through each node in the scene graph using the WorldSpaceVisitor and adds each Spatial containing EgocentricContextData to the World Space set in the SSMBundle; more on the SSMBundle in Section 4.6. The world space is computed only once, when the simulation is initialized. The WorldSpaceVisitor follows the visitor design pattern [17].

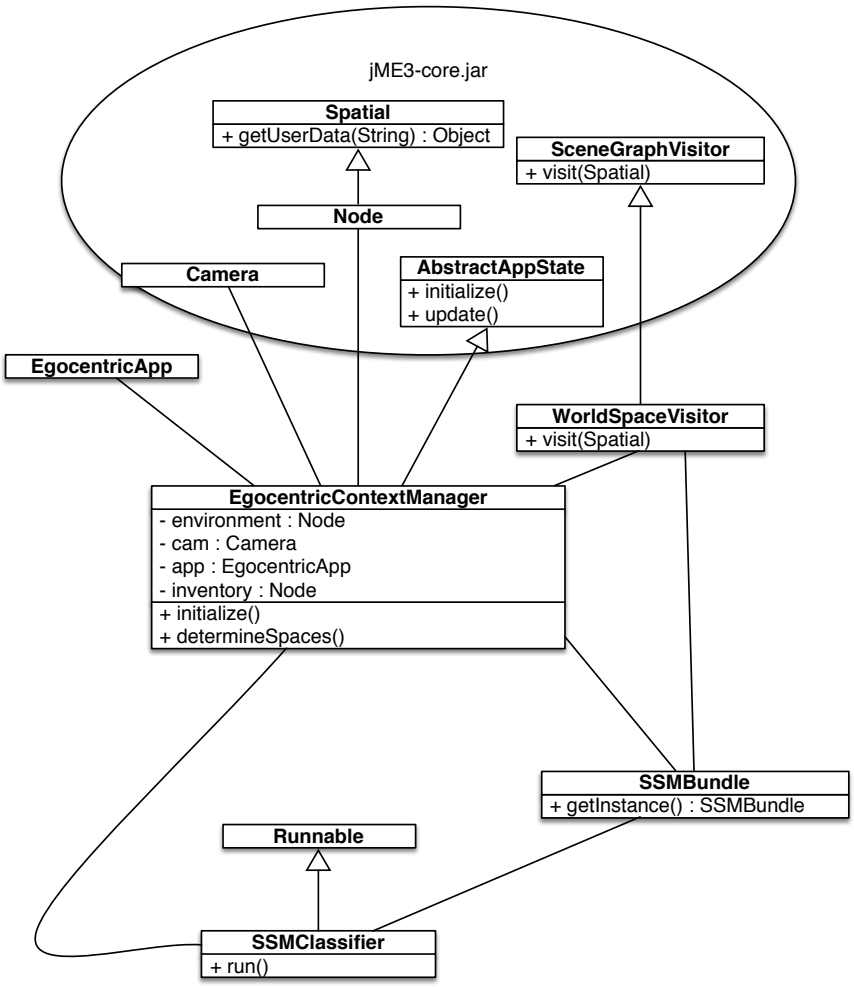


Figure 28: Context Manager class diagram

The agent starts up the classification process by calling the *determineSpaces()* method of the active EgocentricContextManager state instance. The classification process runs in a separate thread, the SSMClassifier.

4.5 THE CLASSIFIER THREAD

Initially, we have ran the classification within the state instance. This lead to performance issue as heavy computation was running on the UI thread, creating a disruptive user experience, sometimes heavily lagging the rendering process. Moreover, the agent triggers a new classification multiple times per second (this depends on the number of frames per second the simulation is running at) resulting in a classification being triggered before the previous one had finished. In the current implementation we allow only one classification to run at a time to avoid overlapping results. The process could be improved

in the future by running multiple classifications and taking into account the results from the most recent one. However, this improvement would need thorough testing in order to avoid false positives.

The SSMClassifier classifies only the objects in the world space, as those are the one carrying EgocentricContextData. This approach drastically reduces the computation time as we don't have to parse the whole scene graph every time a new classification is triggered; and performance is imperative for the classification process as it needs to happen in real-time.

The classification logic is spread among multiple classes as depicted in Figure 29. This architecture was designed to be extensible, based on the strategy design pattern [17]. Therefore, the AbstractProximitySSMSpaceComputationStrategy is a class hierarchy specialised at classifying objects in the agent's surroundings based on proximity. But the AbstractSSMSpaceComputationStrategy could be extended with other specialised hierarchies. For example, to classify objects around the agent in its audible field.

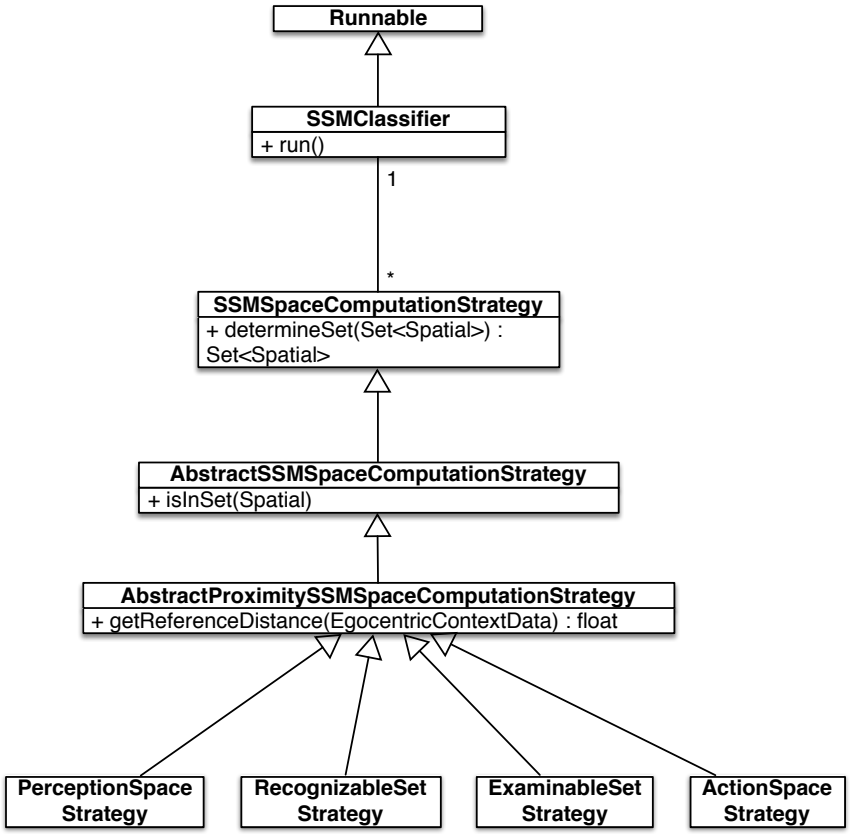


Figure 29: Computation Strategy class diagram

When the classification process is started up, it first determines the set of objects in the agent's field of vision. The algorithm of determining whether an object is currently visible to the agent or not, is embedded in the SSMClassifier.isOnScreen(Spatial) method. Having the list of currently visible objects, each concrete strategy (i.e. PerceptionSpaceStrategy) is responsible to determine which of these objects should be part of a certain space.

After the classification process is finished, all the sets in the SSMBundle are up to date.

4.6 SSM BUNDLE

The SSMBundle is a singleton containing the SSM Sets used throughout the application. It is a shared resource and all the other components have access to it. The monitoring service updates the sets as classifications are carried out, the API and context client access the sets to provide this context information to third party clients, the agent uses the information in the sets to implement the business logic for various types of interaction.

Being a resource shared among multiple components and several threads, thread safety is imperative for the SSMBundle. Therefore, only one thread can access/modify the content of this resource at a time.

4.7 THE API

In Section 3.8 we have designed a RESTful API serving JSON content for third party service to retrieve the egocentric context data in the SSM Sets. There are many ways one can implement a RESTful API in Java, ranging from using one of the many existing libraries, to building it from scratch. The EgoSim project is a simple Java application with no server-side capabilities. Meaning that in order to provide an API we would first need to create a HTTP server to intercept and parse the requests, with capabilities to serve multiple requests at the same time.

Building up the API from scratch, would have been a redundant work given a wide range of third party libraries offering these features ready made. One of these frameworks is Restlet[12]. We have chosen Restlet, as it is one of the leading open source RESTful web API frameworks for Java. Moreover, we have already worked with it in previous projects, saving us time when integrating the library into

12 http://restlet.com

our framework.

The classes that implement the functionality of our API are depicted in Figure 30. The ContextApiServer is initialized in the monitoring service component 4.4. This starts up a HTTP server accessible within the network the physical computer is connected to. If accessed from the same computer the simulator is running on, it can be accessed on `http://localhost:8182/context/api`. This endpoint accepts only GET requests, each request being handled by a new instance of the ApiContextResource class.

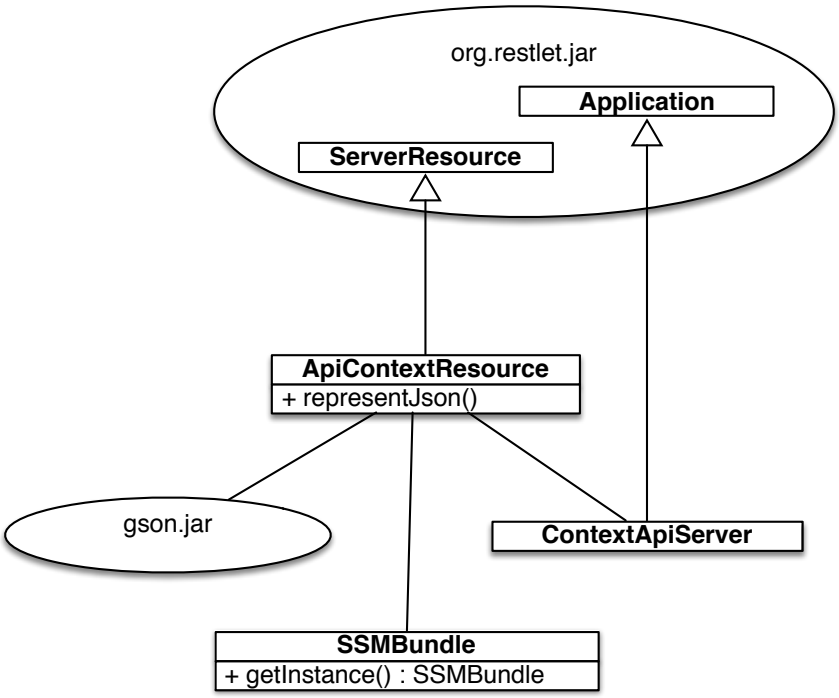


Figure 30: Class Diagram of the API classes and their third party dependencies

The endpoint accepts an URL parameter called *name* to specify the name of the set the third party service wants to retrieve. The possible values are: *worldSpace*, *perceptionSpace*, *recognizableSet*, *examinableSet*, *actionSpace*, *selectedSet*, *manipulatedSet* and *all*. For example, to access the content of the World Space, the third party service would issues the following GET request:
`http://localhost:8182/context/api?name=worldSpace`.

Whenever a new request is encountered, the ApiContextResource retrieves the data from the requested sets and serializes them to the JSON format. To convert the set contents to JSON we have used the

GSON[13] library, a Java library that can be used to convert Java Objects into their JSON representation. An example JSON output of the World Space from the API is illustrated in Figure 31.


```
{
    "worldSpace": [{
        "id": "Pen",
        "type": "PHYSICAL",
        "interactionType": "PICK_UP",
        "surface": false,
        "perceptionDistance": 200.0,
        "recognitionDistance": 80.0,
        "examinationDistance": 50.0,
        "actionDistance": 13.0,
        "weight": 0.0,
        "lastMeasuredDistance": 60.17832
    }, {
        "id": "Table1",
        "type": "PHYSICAL",
        "interactionType": "PICK_UP",
        "surface": true,
        "perceptionDistance": 500.0,
        "recognitionDistance": 500.0,
        "examinationDistance": 200.0,
        "actionDistance": 13.0,
        "weight": 30.0,
        "lastMeasuredDistance": 60.20811
    }]
}
```


Figure 31: JSON output example

4.8 THE CONTEXT CLIENT

The context client component needs to provide a user friendly representation of the SSM Sets, as the simulation unfolds. In Section 3.9 we have designed this component to work on top of the API. But, the context client also needs to represent data close to real-time. So, instead of introducing an level of indirection by connecting to the API, which in turn adds a certain delay to retrieving the response, we are using the information straight out of the SSMBundle. This means that the context client is not running on top of the API, rather it is a standalone component.

In the current version we have implement the context client as a web page served right out of the running simulation. Once loaded,

13 https://code.google.com/p/google-gson/

the web page is automatically refreshed every second. A screenshot of the context client is depicted in Figure 32.

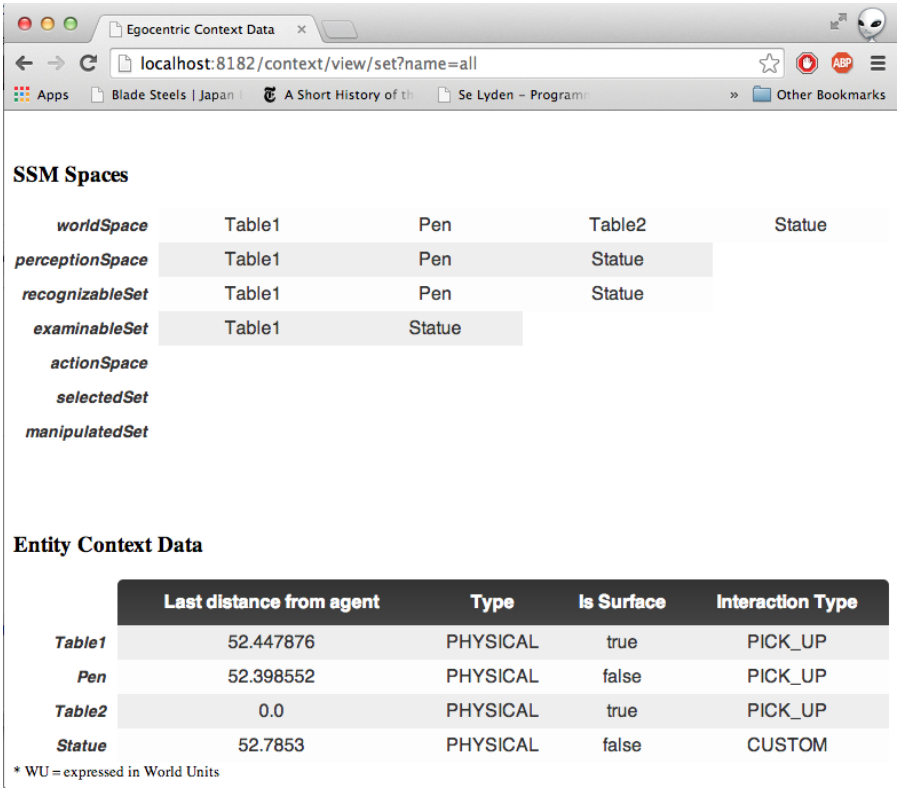


Figure 32: Screenshot of the Context Client

To implement the context client we have used the same approach as we did with the API 3.8. We have use the Restlet framework to create an endpoint serving HTML content. The class diagram in Figure 33 illustrates the classes implementing this components behaviour. The ContextViewServer is initialized in the monitoring service component 4.4. If accessed from the same computer the simulator is running on, it can be accessed on `http://localhost:8182/context/view`. This endpoint accepts only GET requests, each request being handled by a new instance of the ViewContextResource class.

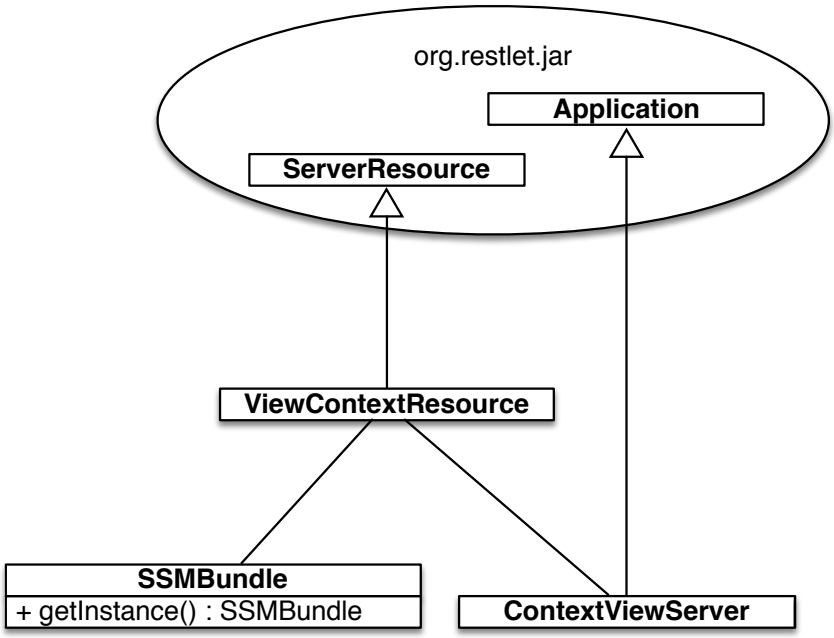


Figure 33: Class Diagram of the Context Client classes and their third party dependencies

The endpoint accepts an URL parameter called *name* to specify the name of the set the third party service wants to retrieve. The possible values are: *worldSpace*, *perceptionSpace*, *recognizableSet*, *examinableSet*, *actionSpace*, *selectedSet*, *manipulatedSet* and *all*. For example, to see the content of the World Space in the context client, the simulation user would access `http://localhost:8182/context/api?name=worldSpace` from a web browser.

The problem with the current implementation is that it does not reflect the new information immediately after a change in the context occurs. It only does so every second. To improve refresh time, we can use Web Sockets[14] in the future work. The WebSocket specification defines a full-duplex single socket connection over which messages can be sent between client and server. This means that instead of refreshing the context client every second, the ContextViewServer could push up to date information towards the context client, whenever available, through an open web socket connection.

The advantage of having implemented the context client as a web page, is that no additional software installation is required. It can be accessed from any browser, on any platform.

14 `http://www.websocket.org/`

4.9 PROTOTYPE SIMULATIONS

To prepare the framework for the evaluation processes, we have set up a few prototype simulations, using the steps described in the user guide in Appendix A. The class diagram in Figure 34 illustrates the classes implementing the two prototype simulations. The WarmUpTask and ALFTask are two individual simulations extending the EgocentricApp class. Explaining the meaning of the environments used in the simulations is out of scope for this section, this information is tightly related to the evaluation and will be addressed in Chapter 5.

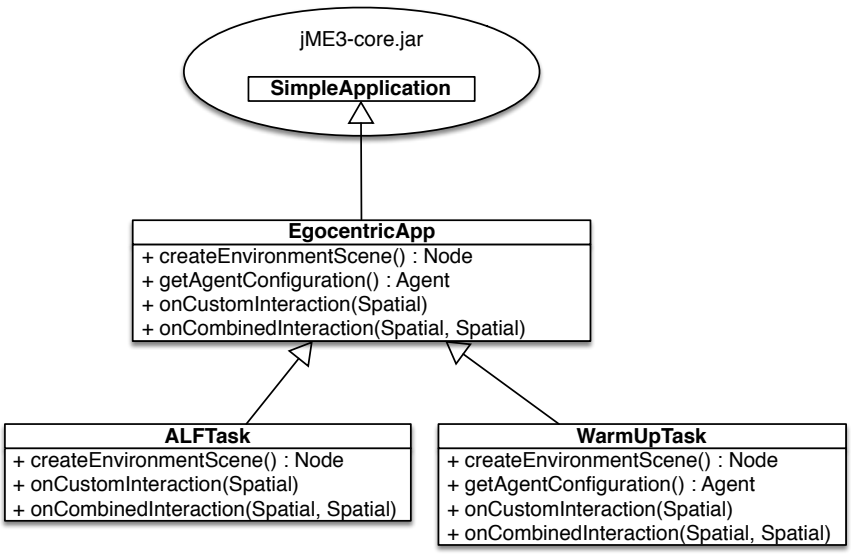


Figure 34: Prototype Simulations class diagram

As it is the most popular open source 3D modelling application, we have decided to implement some of the 3D model in Blender. To learn about 3D modelling in Blender please refer to the on-line manual[15]. JME3 supports direct import of .blend files. No intermediary formats are needed.

Next, we will shortly address each of the prototypes.

4.9.1 *WarmUpTask*

This simulation was used during the development of the framework to validate the ongoing work. For this simulation we have set up a 3D environment pragmatically. When the createEnvironmentScene() is called, it builds up a simple environment using standard shapes in the JME SDK and a few available models. The screenshot in Figure 35 illustrate a view over prototype one's 3D model. We have named

15 http://wiki.blender.org/index.php/Doc:2.6/Manual

it WarmUpTask because it was used as the first task in the evaluation process, helping the subjects of the evaluation familiar with simulator. The 3D model is made up by two tables, a sphere and a cylinder. All of them are configured with EgocentricContextData.

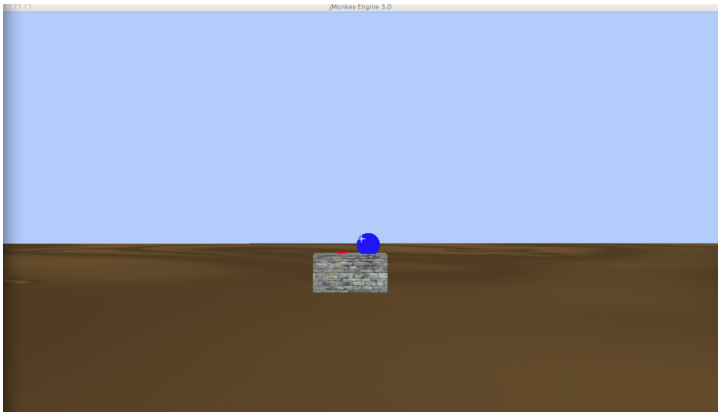


Figure 35: WarmUp Prototype

A detailed description of the scenario this simulation is involved in can be found in Appendix B.1.

4.9.2 *ALFTask*

The ALFTask represents the simulations used as the second evaluation task. For this simulation we have create a 3D model in Blender. The model resembles a one level home interior with kitchen, bathroom, living room and bathroom. The environment is populated with appliances and everyday physical objects and devices. Some of the objects have been configured with EgocentricContextData. The 3D model is available in the framework's source code in the *Assets/Scenes/alf* folder; for information regarding the source code see Appendix C. Also, a detailed description of the scenario this simulation is involved in can be found in Appendix B.2.

Interesting to note in the ALF simulations is that we have implemented a custom interaction action with the piano depicted in Figure 36. The piano's EgocentricContextData is configured with interactionType CUSTOM. When the agent interacts with the piano, the simulation plays a series of musical notes.

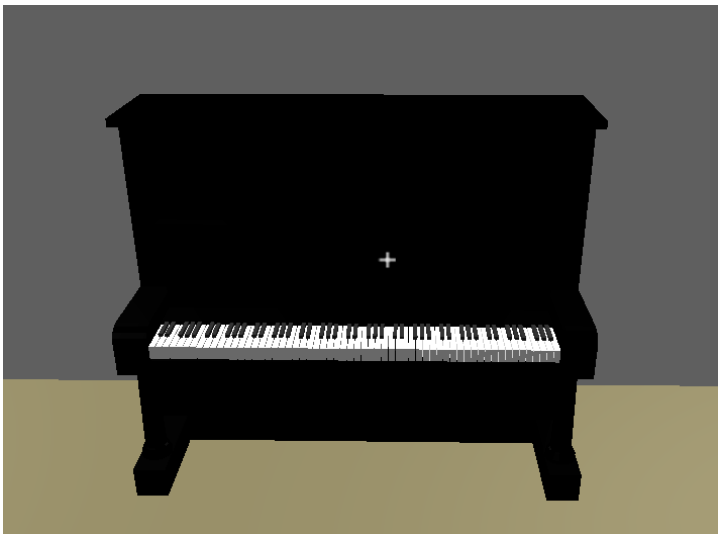


Figure 36: Example of custom interaction with the piano in the Assisted Living Facility environment

To implement the custom interaction behaviour, we have overridden the onCustomInteraction() method as illustrated in the following Listing. This is an example of why it is important to choose good and unique IDs for the entities while augmenting them with EgocentricContextData – because they are used to implement business logic further on.

Listing 1: Snippet of code illustrating how to implement a CUSTOM interaction with an object

```
@Override
public void onCustomInteraction(Spatial spatial) {

    final EgocentricContextData data = spatial.getUserData(
        EgocentricContextData.TAG);
    if ("Piano".equals(data.getId())) {

        /* play notes */
    } else {
        super.onCustomInteraction(spatial);
    }
}
```

Moreover, we have also implemented an example of combined interaction. The screenshot from Figure 37, presents an example of combined interaction when the agents tries to interact with the Cup while holding the Pot. This triggers causes the combined interaction callback to be called on the concrete simulation instance.

Figure 37: Example of combined interaction within the Assisted Living Facility environment

To implement the combined interaction behaviour, we have overridden the onCombinedInteraction() method as illustrated in the following Listing. In this version we simply play a pouring liquid sound. But, from the ALFTask we have access to all the components and assets of the simulations, so more complex actions could be taken (e.g. animating liquid pouring from the Pot into the Cup, etc).

Listing 2: Snippet of code illustrating how to implement a COMBINED interaction between two objects (when the agent acts upon an object while holding another object)

```
@Override
public void onCombinedInteraction(Spatial pickedUpObject, Spatial
     withObject) {
    super.onCombinedInteraction(pickedUpObject, withObject);

    final EgocentricContextData data1 = pickedUpObject.
        getUserData(EgocentricContextData.TAG);
    final EgocentricContextData data2 = withObject.getUserData(
        EgocentricContextData.TAG);

    if("Pot".equals(data1.getId()) && "Cup".equals(data2.getId())
        ) {
        /* combined interaction action */
    }
}
```

4.9.3 *Childproof model*

For the last evaluation task we have not implemented a simulation as that is the task the subjects of the evaluation has to carry out. But, in order to allow them to focus on working with the framework, we have provided a 3D Blender model. The 3D model is available in the framework's source code in the *Assets/Scenes/alf* folder; for information regarding the source code see Appendix C.

The model resembles a living room with a semi-integrated kitchen. The environment has been populated with appliances and everyday physical objects and devices. A detailed description of the scenario this simulation is involved in can be found in Appendix B.3.

5 EVALUATION

We have recruited a total of 17 participants, out of which 12 male and 5 female. All of them experienced Software Engineers, 15 working in the industry, 1 Post-Doctoral Researcher and 1 Associate Professor. Their average experience in software engineering is 6.41 years, with an average of 1.56 year of experience in mobile development. 47.1% are familiar with concepts of context-aware computing, while only 29.4% have actually worked on context-aware systems. Finally, one participant was familiar with concepts of the egocentric interaction paradigm or the SSM.

To evaluate to what extent we have met the goals defined in Section 1.4, we conducted an evaluation in two parts. The whole evaluation session, including reading the documentation and providing feedback, lasted between 90 and 120 minutes.

First we have evaluated a simulation created with the EgoSim framework from the simulation user's perspective. The users were able to explore a ready made simulation while observing contextual changes in the context client. Second, we have evaluated the EgoSim framework from the system designer's perspective, as the users built a simulation using our framework. Moreover, to determine how our system is different from existing system, we have compared EgoSim to two of the most relevant systems in the related work.

The entire documentation the participants were exposed to can be accessed at `http://karolyszanto.ro/MastersThesis/evaluation/index.html`.

5.1 PARTICIPANTS

In order to collect relevant technical feedback, we have aimed at recruiting participant having a fair amount of experience in software development. We have recruited a total of 17 participants. 15 of them are Software Engineers, 1 Post-Doctoral Researcher and 1 Associate Professor. In the beginning of the study they were asked to assess their experience in software engineering and, particularly, in mobile development. Moreover, we gathered information on their experience with context-aware computing and the egocentric interaction paradigm. We briefly present the statistics of our participants profile bellow:

- Years of experience in Software Engineering – an average of 6.41 (ranging from 3 - 15)
- Years of experience in Mobile Development – an average of 1.56 (ranging from 0 - 5)
- Familiar with concepts Context-Aware Computing – Yes: 47.1% (8), No: 52.9% (9)
- Has worked on context-aware system (e.g. a mobile APP using the device's location to perform implicit actions) – Yes: 29.4% (5), No: 70.6% (12)
- Familiar with the egocentric interaction paradigm or the SSM – Yes: 5.9% (1), No: 94.1% (16)

Analysing the profile of our participants, we can deduct that they have a good seniority level in software engineering and a fairly good knowledge of mobile development concepts. Almost non of them were aware of the egocentric interaction paradigm, but we were able to easily explain the concepts. Learning about the egocentric-interaction paradigm was made easy due to the vast experience the participants have in the field of software engineering.

5.2 PROCEDURE

We have developed a web page the participants were able access, read through the introduction to get up to speed with the terminology and where they could find the scenarios and tasks to finish the evaluation, as well as the feedback forms to help them get back to us with answers. This documentation can be accessed live on the project's evaluation web page [1] or in raw format in the source code repository [2].

Going through the introduction, the participants were first introduced to concepts of context-aware computing, followed by an introduction to the egocentric-interaction paradigm and the situative space model. In the last part of the introduction we have presented the goals of our work and the features of the EgoSim framework.

5.2.1 *Evaluation Tasks*

Next, the participants were given three scenarios. The first step was the WarmUp task, meant to make the participant familiar with the simulator's concepts and ways of interacting with the environment. This step did not require any feedback. The WarmUp task is detailed

1 http://karolyszanto.ro/MastersThesis/evaluation/index.html
2 https://github.com/ksza/MasterThesis_Report/tree/master/evaluation_html

in Appendix B.1.

In the second scenario we have developed a simulation for an Assisted Living Facility where objects around the human agent are categorised according to the SSM. For this task the participants have evaluated the system from the simulation user's point of view. They had to test a ready made simulation with the EgoSim framework. The second task is detailed in Appendix B.2. In Figure 38 a participant is evaluating the Assisted Living Facility (ALF) simulation.

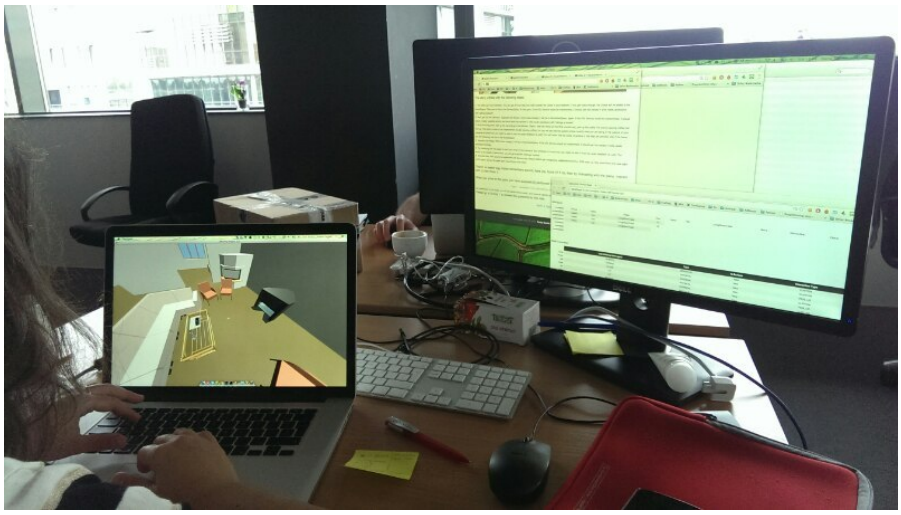


Figure 38: A participant in our user-testing trying out the Assisted Living Facility Scenario

In the third scenario, the participants had to develop a new simulation using the EgoSim framework. The hypothetical problem they were given as part of this task is that families cannot make their homes secure enough for their children. Therefore, the participants were asked to imagine themselves being a system designer solving this problem using the EgoSim framework. This third task is detailed in Appendix B.3. Setting up 3D models for EgoSim is done in third party software, therefore it is out of scope for its evaluation. We have provided all the 3D models needed by the participants to finalize the tasks. In Figure 39 a participant is using EgoSim to build the Childproof simulation.

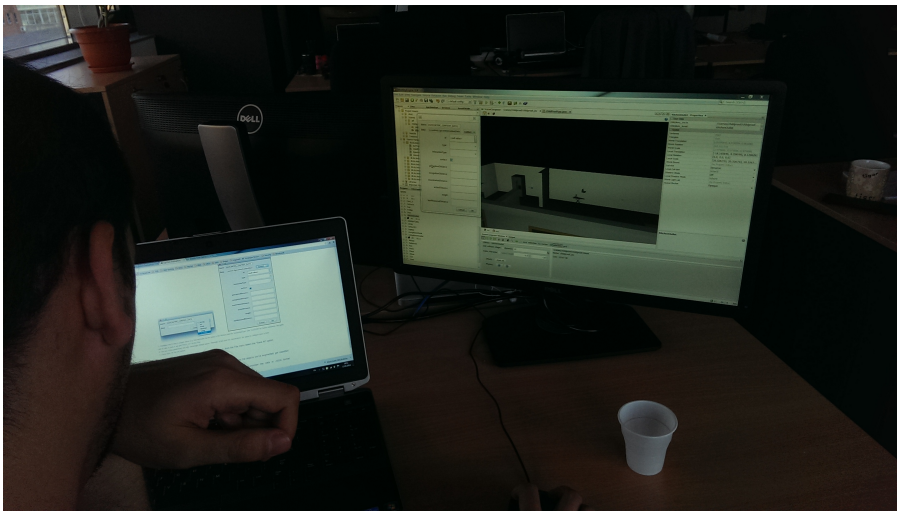


Figure 39: A participant in our user-testing building the simulation for the Childproof scenario

Except for the warmup task, the participants were asked to fill in the feedback forms provided at the end of each evaluation scenario. The forms resemble a semi-structured interview which include questions regarding the usefulness of the system, usability of the framework and of the resulting simulations. We have asked many open questions encouraging the participants to provide detailed feedback that came to their mind about various aspects of the system (software bugs they might have found, what they liked, what they didn't, etc).

5.2.2 *Feedback & Discussion*

We would have liked to do a comparative study where the participants would create similar simulations using one of the systems presented in the related work chapter 2. However, those systems do not support some of the features our system making it impossible to implement a replica of the same simulation. Moreover, the systems in the related work do not provide classification of objects around the agent according to the SSM, which makes the results of a comparison between a simulation built with EgoSim and one of the systems in the related work irrelevant. We therefore proceed with a user-test of our system with no comparison to other methods.

In the feedback form we have asked the participants a series of questions. The first set of question was about aspects of simulations built with EgoSim, asked from the simulation user's perspective as enumerated in Appendix E.1. The second set of question was about the framework itself asked from the perspective of a system designer,

as enumerated in Appendix E.2.

For a complete assessment of the framework, we should have covered the third role as well, evaluating the system from a third party system's point of view. Normally, we would have asked the participants to build a piece of software which connects to our API and implements some business login on top of the retrieved data. Instead, to reduce the evaluation time, in the second set of questions we gathered feedback about the API from the participants. Moreover, in the last question the participants were asked to describe in plain words or pseudo-code language the business logic to solve a given problem using the SSM Sets available through the API.

The whole evaluation session, including reading the documentation and providing feedback, lasted between 90 and 120 minutes. Videos of running the three scenarios are available; see Appendix D. Next, in Section 5.3 we will discuss the results of the evaluation.

5.3 DISCUSSION

The evaluation results can be found in Appendix E.1 and Appendix E.2. Before we carry on with the discussion, we would like to point out one more time that the level of experience in the field of software engineering of all of our participants was high. We believe that the results of the evaluation provide good guidelines for future work on the system.

For the ALF evaluation scenario we came up with a hypothetical problem and provided a solution based on the SSM. We were concerned that the participants might not agree that designing the solution based on the egocentric interaction paradigm is the right approach. Therefore, we have asked them E.1.7 whether the SSM is the right context model for the ALF system and for the third party services developed around it. All the participants agreed that this was the right approach, assuring that we have chosen a strong and relevant scenario for this evaluation.

5.3.1 *Characteristics*

The tool we have built is meant to be used by system designers to create simulations. The resulting systems are then used by simulation users and third party services. To meaningfully assess the quality of the EgoSim framework and of the simulations which can be built with it, we have to take a step back to inspect the goals 1.4 and and

requirements 3.1. The requirements have been identified based on the goals of the system and guided by an end-to-end scenario 1.6 we have imagined as the intended way to use our system. Given that the result of our work is an interactive system, we have evaluated the system from the end user's point of view.

To quantify the user feedback into a meaningful discussion, we have extracted four characteristics of interest which emerged from the goals and requirements of our system. To evaluate how useful our requirements are and to what extent our system have met them, we will look into the *usefulness* and *usability* of both the framework and simulations.

Given that simulations created with EgoSim are interactive systems, we are interested about how *responsive* they were found by simulation users. Moreover, the ContextClient is meant to be a close to real-time context visualization tool, therefore we are assessing its responsiveness as well.

The core of a simulation is the classification process and the SSM context model. We have found it imperative to assess the quality and relevance of how objects around the virtual agent got classified during the simulation.

To conclude, in the remainder of this section we will discuss the evaluation results in the light of four characteristics: usefulness, usability, responsiveness and classification.

5.3.2 *Usefulness*

One of the main objectives of our evaluation was to investigate if the users found our system useful. On one hand we were interested if the participants found simulations built using EgoSim as being useful; on the other hand if they have found the system as a whole useful to simulate systems designed around the SSM context model.

5.3.2.1 *The Framework*

All the participants have seen the value of simulating a system using EgoSim, over directly implementing the system into a real set-up. This strengthens the hypothesis of our work, that there is a real need for the system we have built. As one of the users stated in the feedback for question E.1.8: "The simulation could provide a cost effective way of gaining valuable insight into the design of an Assisted Living

Facility before such a facility is constructed". Moreover, some participants pointed out some benefits of the simulator we have not stated:

- lower costs in case of trial and error
- gives you a perception of how people interact with the objects
- faster feedback loop
- a simulation is a reliable and cheap proof of concept
- it allows the discovery of edge cases

To further assess the usefulness of EgoSim, all the users found it useful to simulate systems build with the SSM context model at their core. One of the participants stated in the feedback for question E.2.3 "Objects can be easily decorated with context information and interacting with them can be tested quickly because the framework takes care of basic interaction and movement in the space". The fact that we have inferred functionality from based on the Ego Metadata configuration, makes the framework a lot more useful for users than we though in the first place. None but one of the participants were previously aware of the SSM context model, but they have managed to successfully build a simulation in the second task: "with a few easy steps I was able to model the desired environment".

Our system proved to be a useful educational tool as well. Before staring the evaluation, only one participants reported as being familiar with the egocentric interaction paradigm E.1.1. After the evaluation the participants have reported an average of 4.5 level of understanding about the SSM, on a scale from 1 to 7 E.2.1. This means using the EgoSim framework, participants got rather good insights into what egocentric integration paradigm means and how the SSM works. In the future, it might prove to be a useful tool for students studying ubiquitous computing to learn about these concepts.

In one of the feedback questions we have asked the participants to imagine various system they could build using based on the SSM Sets and the outcome of our classification algorithms. The have came up with a list of interesting ideas E.2.11; we have listed some of them bellow:

- Assistance for people with disabilities (especially the visually impaired), to offer an augmented representation of their environment.
- A tourist map that is context aware and can recommend places to visit that are near the current location, that are for example still open (maybe it's night, you don't want recommendations for a zoo), etc.

- Security application: auto-lock my computer screen when it is in my Action Space.
- Automatic light switching based on the position of the inhabitant of a home (not motion-based). This can be generalized to a system that defines custom operation modes of devices in a home, depending on what the user is doing.

All of these systems/services are useful systems and they can easily be simulated with EgoSim. The fact that so many participants were able to come up with ideas of systems build on top of the SSM context model, means the context model has a lot of potential for context aware system development. Therefore, the support EgoSim offers for building simulations with this context model as a cornerstone, based on the user feedback, turns out to be useful based on the user feedback.

5.3.2.2 *The Simulation*

When creating a simulation of a real life environment, immersion of the user into the virtual environment is rather important. The user must identify herself/himself with the agent and the simulated environment must resemble the real environment it actually represents. This has two implications: on one hand the 3D model of the environment must be good enough to resemble the real life setup of the simulated environment; on the other hand, when the simulation is running and the user is interacting with the environment, she/he must feel absorbed to a certain extent. All the participants found the ALF simulation a good simulation for a home E.1.2. We have build both the ALF and the Childproof 3D models with a third party software and used them with our framework. Moreover, the participants have experienced the virtual environment by means of the running the simulation. The fact that all participants found it a good simulation of a home, assures that third party software can supply useful 3D models. The feedback from our participants makes us believe that we made the right decision to use a game engine to render the 3D models and provide interaction capabilities.

We believed the level of object details for this evaluation was not relevant. One of the participants proved us wrong saying "given the high system requirements I would have expected a better graphics/game engine". Reflecting upon the issues this is not a game engine problem. The jMonkey Engine has all the capabilities of rendering high quality models. We have simply implemented models with low details. However, developing high detail models requires much more time invested in the modelling process.

5.3.3 *Usability*

5.3.3.1 *The Framework*

All participants were positive about the usability of the system. Creating a simulation using EgoSim comes down to do some configuration work on top of a 3D model. Using EgoSim to design a new simulations, the participants have reported that identifying and configuring the objects to be classified during simulation, on a scale from 1 to 7 (Very easy - Very hard) was an average of 2.1 E.2.4. This means that almost everyone has found the process of designing a new simulation easy. We have based this process on the jMonkey SDK's components. We were sceptical at first and thought it might be a laborious and hard to understand process, but based on the user feedback, we conclude it was the right decision.

Although, one of the participants was not able to finish the second task E.2.2 due to not being able to augment the desired object models with EgocentricContextData. This was most probably due to the fact that when configuring an object in the jMonkey scene composer, the name of the custom user data must be EGOCENTRIC_CONTEXT_DATA. If it is not exactly this, even if the name has an extra space, the framework will fail to identify object to be classified. A user feedback to improve this feature was to "trim the context variable for space EGOCENTRIC_CONTEXT_DATA". We could indeed rephrase the name into something shorter, but this is a task for future work.

5.3.3.2 *The Simulation*

When running the simulation, the participants felt that interacting with the environment was very intuitive. On a scale from 1 to 7 (Not intuitive at all – Highly intuitive), they have reported an average of 5.5 E.1.4. This translates into the interaction with the environment being very intuitive. This is of course valid only for moving around the environment and for the default interaction we have provided for objects – pick-up/drop-down. Some users felt that a generic USE action would be more appropriate "A generic USE action would help is many situations as many objects that are interacted with might have a main USE method. pickup/putdown could be specific implementations of use". Moreover, other kind of interactions have been requests like turn off/on for electronics, shower, etc. All these aspects will be discussed in Section 6.1.

The collision detection (the fact the agent could not pass through objects) was especially well received "I think moving around was very close to reality and I liked the fact that it was aware of the obstacles.

For example, you could not just move forward through the couch to get to the coffee table".

5.3.3.3 *Context Data and Contextual Interactions*

The custom interactions we have implemented in the fist scenario were also well received: "The Piano playing actions were nice". Some participants noted that support for building custom interaction would have been useful: "Would be nice if people could declare their own type of interactions". This is actually possible with the current implementation, unfortunately we could not add a tasks for the participants to try this out; the evaluation took a fair amount of time as it is. The fact that it is a required feature, means we have well anticipated this need by implementing support for both CUSTOM interaction with objects, as detailed in Listing 1, and COMBINED interaction between two objects, as detailed in Listing 1.

Most context aware systems need to access the historical information of changes in context of various entities. We have not implemented this aspect, but we should plan for this in future work. One of the participants even mentioned the lack of this feature stating that it would be useful to have "Historical information about the agent: what actions has he performed so far, objects with which he has interacted with so far, etc".

A potential weakness of the framework was pointed out by one of the participants saying "a potential weakness of this framework would be the lack of information regarding the relationship between objects". In the current implementation we monitor the context of surrounding objects only from the agent's perspective. But context information that is affected by relationship between objects might also be relevant. This could be solved by adding support for sensors to our framework, which we lack at the moment.

5.3.3.4 *The API*

To evaluate the framework from a third party service's point of view, in the second task we have given the participants to solve a hypothetical problem in the context of the scenario, as a developer of a third party service based on data available through the API. 16 out of the 17 participants were able to provide the correct solution E.2.11. This makes us believe the data in the SSM sets provided through the API is explicit enough to build service aware of the current context in the ongoing simulation.

In the current implementation context changes occur only as the agent interacts with the environment. Some participants pointed out that it would be useful to modify some context data through the API. We will reconsider this idea in future work.

Overall, the framework as a whole was perceived by the participants as being highly usable E.2.5. One of the participants even said that "it is very easy to import a scene and augment the objects with different information and interaction types".

5.3.4 *Responsiveness*

Target of our evaluation was also to assess how responsive the simulation and the context client turned out to be. The simulation was found to be responsive and running smoothly by 88.2% of the participants E.1.9. We are especially happy about this outcome because the classification process implies heavy computation under the hood, which means we have done the right thing implementing the classification as a standalone process. One of the users reported that "there was a glitch when I could not move forward when getting out of bed. I had to move in a different direction and after that come back and approach the closet". This is most probably because we have used a deprecated CharacterControl[3] class from the jMonkey framework to help us in with the agent's movement. In future work we should address this issues by upgrading to the BetterCharacterControl[4] class.

When asked about the responsiveness of the context client E.1.6, 76.5% of the participants reported it being responsive and updating the context changes close to real time. Some of the comments from the other 23.5% made us realised that there's place for improvement for the context client. Currently the context client is implement as a web page served from within the running simulation. The page automatically updates every second. This approach has some drawbacks as it will not display the context changes instantaneously. Moreover, it queries all the SSM sets over and over again without some of them modifying at all. This creates an unnecessary overhead as the context client is accessing the same shared resource the API and classification modules are using. Therefore, a more suitable implementation for the context client would be based on web sockets. This way, we would reverse the flow of information; the server would send notifications to the opened web page every time a change in context occurs. Anyhow,

3 http://hub.jmonkeyengine.org/javadoc/com/jme3/bullet/control/CharacterControl.html

4 http://hub.jmonkeyengine.org/javadoc/com/jme3/bullet/control/BetterCharacterControl.html

for the purpose of this evaluation the context client played out well and it's responsiveness was well received.

5.3.5 *Classification*

A little more than half of the users have reported that the monitored objects were correctly classified into SSM sets E.1.3. This means our classification algorithm performed decently. The rest of the participants reported there were cases when the classification did not work as expected. There were two problems the users have noticed which prevented the classification to work as expected.

The first problem was that some objects were not being classified at all. These objects were actually never augmented with egocentric data, therefore with the current implementation they cannot be classified. We though this was clear from the description in the evaluation, but we might have had to stress more on this matter. However, if all the objects in a scene were to be augmented with Ego metadata, during simulation they would all be taken into account and classified.

The second problem was that although some objects were visible, they were not classified as being perceived by the agent. This is a problem we are aware of and plan on addressing in future work. At the heart of our classification algorithm lies the way we determine if an object is currently visible to the agent or not. To do this, we cast a ray from the agent's current location to the centre point of the object's bounding box (think of it a the object's centre of gravity). If the ray intersects any other object before hitting the middle of the object, we consider the object as not being visible to the agent. This works in most cases, as the user feedback shows. But it has a lot of edge cases. For example the centre of a table can be occluded by a coin; this does not make the table unrecognisable, it just make our algorithm generate a false positive. Likewise, the agent could see through a hole in the wall the only centre of a drawer. Without other details, that object might be unrecognisable.

We have foreseen this limitation in Section 3.7 and we have provided a solution for the future work. To overcome these kind of edge cases, we could cast a large number of rays to various points on the object and based on a more advanced algorithm, determine if the object is being occluded. We have decided to leave the design of a more advanced computation for future work, as the current solution is good enough to cover most of the cases, while developing a more advanced and correct algorithm needs more research in the field of 3D modelling and rendering.

One of the participants provided a very helpful feedback in this matter "I consider the condition for an object to enter the perception space to be too weak. A user cannot identify an object if only a pixel is within his field of vision. Humans are built to recognize patterns, therefore a pattern-oriented metric could be used. For instance, the introduction of a diversity factor would help the simulation. If enough diversity of the object is within the perception space, it could trigger its presence in the perception space". We entirely share this opinion and will be taken into consideration in further improvements of the classification algorithm.

For the agent to be able to interact with an object, that objects must be in the ActionSpace, therefore within reach, or within the ActionDistance. We have tailored a default value for this property. The system designer is free to modify this value at design time, but the participants did not modify it. Most of them (88.2%) have felt that the default value was close to realistic E.1.3. One participant felt it was too close and another that it was too far. The problem discussed above has impact when classifying into each sets, therefore affecting the interaction between the agent and the surrounding objects.

Taking a step back to Section 5.3.3, the fact that such a large number of participants have found interacting with the surrounding objects highly intuitive, means that they were able to easily pickup/drop down objects. This means the classifications are running close to real time. Why is that? Because every time the field of vision of the agent changes, we run a new classification. Even the slightest change to user's field of vision would trigger a new classification. Within the framework, decision are taken based on the latest available result of the classifications. As the simulation was perceived to run smoothly and interacting with the environment seemed natural, we conclude that decisions during the simulation were made according to the current context, meaning the classifications were execute almost instantaneously. In future work it would be interesting to evaluate how a very large number of tracked objects affect the performance of our classification algorithm.

5.4 COMPARISON TO EXISTING SYSTEMS

We have designed and implemented a framework that allows system designers to create simulations of mobile context-aware systems. The simulator empowers users to intuitively interact with the simulated environment, while classifying the objects around the agent according to the SSM context model.

UbiWise 2.4 is a device-centric simulator, it focuses on user manipulation of devices and the interactions between devices.

TATUS 2.5 targets the intelligent technology controlling ubiquitous computing environments through the use of embedded sensors and actuators.

In Section 3.1.1 we have identified three roles for the EgoSim system. For two of them we have found similar roles in all the systems from the relate work described in Chapter 2: the system designer and the simulation user. Some systems might not have used the same terminology. For system designer some of the related work have used the term *researcher*, while for the simulation user has been named in some related works simply as *user*.

We conclude this chapter with a comparison on how some of the simulators presented in the related work differ from EgoSim, with respect to each of these roles. We have chosen only UbiWise and TATUS as their simulator are closest to ours, meaning that they represent the target environment as a 3D model. Moreover, they use a game engine to render the environment and to allow users to interact with it.

5.4.1 *Role1: The System Designer*

The system designer is a researcher or a developer designing and implementing a simulation of a context-aware system. This role shares a common working methodology throughout all three systems:

1. using an editor, she/he creates a 3D model of the simulate environment
2. using the framework and the underlying game engine, she/he develops the simulation's business logic. This step might need thorough understanding of the game engine, in case the system designer has to implement more than what the framework has to offer

5.4.1.1 *UbiWise*

"The building blocks of UbiWise are the simulated devices, the virtual representations of the physical devices"[19]. Using a designer written in Java, the system designer creates a model of the simulated device. This model will be represented both in the 2D and the 3D view. Next, the system designer uses GtkRadiant[5] to design the 3D model of the simulated physical environment where the device will be used. GtkRadiant is the official level design tool chain for the Q3A

5 http://icculus.org/gtkradiant

game engine. The drawback of this environment designer, as stated in the UbiWise paper is "Operation of this editor is complex and it takes a couple of days to understand it; online resources for models often require format conversions to work in the editor. Thus the effort for the first simulation could be a week's time;"[19]. The last step is to provide functionality for the device. This is done by means of an XML configuration file. It configures various actions to be handled by custom Java classes. These actions will be triggered from the 2D view.

5.4.1.2 *TATUS*

In TATUS, the system designer uses the Hammer[6] map editor to model an environment. It is a drawing tool for building maps. Hammer compiles maps a format used by the Half-Life game engine, which TATUS was built upon. To add sensors/actuators to the environment, the system designer would place *triggers* at various locations in the map. Triggers are specific concepts from the Half-Life game engine. For clarity, imagine them as invisible entities within a map, which generate events based on a player's movements and location. So, if a player enters a region of a map occupied by a trigger the associated event is activated (e.g. a door is opened).

Next, the TATUS framework handles the aggregation of the generated messages; basically manages the context of the simulation. The context information is available through an API. To develop the simulation's business logic, the system designer implements a new System Under Test (SUT), in Java, which connects to the API and implements business logic based on the available context information.

5.4.1.3 *EgoSim*

For EgoSim, the system designer can use one of the many 3D modelling software to generate compatible 3D models with the underlying game engine, one of them being Blender[7]. The model can than be imported into the scene composer where the system designer identifies the objects he wants classified according to the SSM context model. To finalise the simulation, the system designer has to write some Java code by extending an abstraction provided by the framework. The abstraction will only need the model to use and some initial configuration data. If desired, the abstraction provide callback methods to implement functionality for custom interactions. During the simulation, the framework makes the context data available through

6 https://developer.valvesoftware.com/wiki/Valve_Hammer_Editor
7 http://www.blender.org/

an API available to third party services.

5.4.1.4 *Discussion*

Because of the game engines they have used, developing the model of an environment for UbiWise or TATUS is bound to their specific map editors. EgoSim, on the other hand, is based on jMonkey Engine which works with various third party 3D model formats. We consider this being a plus. For example Blender, on of the software producing compatible formats, is the most popular open source 3D modelling software. The community produces a lot of free model which can be reused. Moreover, it is well documented which together with the active community helps a system designer get up to speed fairly quick.

If the system designer is in need to develop custom functionality and needs to go to the game engine level, TATUS and UbiWise impose quite a challenge. Both game engines were written in C/C++. As stated in TATUS "Following the experiences encountered while working with the HL SDK, it has become apparent that its complexity requires the presence of an experienced developer. In order to tackle the SDK, a developer must at least have a good working knowledge of C/C++". EgoSim on the other hand is based on jMonkey Engine. It is written in Java, a language widely embraced by the research community. Moreover, it's open source and well documented. A system designer experienced in Java should be able to get up to speed and become proactive within a few day with jMonkey Engine.

5.4.2 *Role2: The Simulation User*

This role is filled up by a person running a simulation build with one of the frameworks. In all three cases, the user interacts from a first-person view with the simulated environment.

5.4.2.1 *UbiWise*

UbiWise offers two views to the user, the 3D physical view (game environment) and the 2D device view (Java Swing Interface). A user in such an instance does all manipulation of the devices through the device view only. The 3D view is used solely to see the effects of device manipulation on the 3D environment. For example, in the example scenario the agent carries a digital wireless camera and in the simulated environment there is a wireless picture frame, both connected to the same content service. When the user uploads a picture from the 2D view, the wireless picture frame reflects the picture that has

been uploaded.

5.4.2.2 *TATUS*

TATUS offers a 3D view. The simulation user can experience the physical environment fully immersed throughout the lifetime of the simulation. The environment will be modified by the SUT based on the business logic implemented for a specific situation (in a certain context). For example, if the agent is close enough to a door, the SUT might decide to open that door. Therefore, TATUS specifically targets predicting user intentions based on notifications from sensors and actuators embedded in the environment. The simulation user experiences changes in the environment while interacting with it, as effect of the SUT's business logic controlling the ubiquitous computing environments through the use of embedded sensors and actuators.

5.4.2.3 *EgoSim*

EgoSim offers a 3D view. The simulation user can interact with the simulated environment by moving around and performing basic interactions with everyday objects (i.e. pick-up/drop-down). As the simulation unfolds, the underlying monitoring service classifies the objects in the agent's surroundings according to the SSM context model. This context information can be accessed from a third party service through an API. Applications build on top of this context model can be always aware of the agent's surroundings, being able to take appropriate actions. Moreover, the simulation user has access to a ContextClient which displays the current status of the SSM context model. This can be used to observe if the simulation was correctly set up.

5.4.2.4 *Discussion*

In all cases the user interacts with a 3D environment, but each environment is meant for a certain type of interaction: UbiWise is device centric, TATUS is sensor centric while EgoSim is body centric.

5.5 CONCLUSION

The results presented in this chapter, conclude that we have managed to develop a usable open-source simulation tool for applications designed based on the egocentric interaction paradigm. We have managed to satisfy the objectives laid out in Section 1.4. By fulfilling the current work we have successfully implemented a first version the EgoSim framework, with many possibilities for future work

as detailed in Section 6.1. The feedback from our evaluation participants have strengthened the assumption that there is a real need for this tool, as they have provided concrete examples of real-life problems solve using the SSM context model. Finally, the comparison between UbiWise, TATUS and EgoSim confirms that this framework contributes with another dimension to the worlds of 3D ubiquitous computing simulators.

6 CONCLUSION & FUTURE WORK

This chapter presents our goals for future work followed by conclusions about the results of this project.

6.1 FUTURE WORK

We have received a fair amount of feedback from our participants, as detailed in Appendix E. We have collected all this user feedback into issues and enhancements in our source code repository[1]. GitHub, the hosting service we use for our source code repository, offers the "issues" feature. This way issues can be managed on the same repository we keep our code in. Being open source, if anyone is willing to contribute to the project, immediate work items are already created and can be addressed.

In the remaining of this section we will detail the proposed future development for the EgoSim framework. Some of the proposed ideas were mentioned in previous sections of this thesis, others come from the user feedback.

6.1.1 *Interaction with virtual objects*

We have taken the compromise to limit the current implementation to physical objects and devices. But to fully support the SSM context model, virtual objects must be classified as well. A virtual object can be a window displayed on display, or a song played on a mobile device. Interaction with these objects is done through mediators (devices).

6.1.2 *Third-person perspective*

Avatar based games and simulations usually provide both first-person and third-person views, with an easy way to switch perspectives during runtime. For this project, a third-person perspective could be useful if we are to represent wearable devices. Also, it would the simulation user to observe the various body gestures the agent is doing. Moreover, in a third-person perspective the simulation user could get a better understanding of the agent's immediate surroundings.

1 https://github.com/ksza/EgoSim/issues

6.1.3 *Improved objects detection*

This goal is about improving the detection of objects in the agent's field of view. At the heart of our classification algorithm lies the way we determine if an object is currently visible to the agent or not. To do this, we cast a ray from the agent's current location to the centre point of the object's bounding box (think of it a the object's centre of gravity). If the ray intersects any other object before hitting the middle of the object, we consider the object as not being visible to the agent. This work in most cases, as the user feedback shows. But it has a lot of edge cases. For example the centre of a table can be occluded by a coin; this does not make the table unrecognisable, it just make our algorithm generate a false positive. Likewise, the agent could see through a hole in the wall the only centre of a drawer. Without other details, that object might be unrecognisable.

One of the participants provided a very helpful feedback in this matter "I consider the condition for an object to enter the perception space to be too weak. A user cannot identify an object if only a pixel is within his field of vision. Humans are built to recognize patterns, therefore a pattern-oriented metric could be used. For instance, the introduction of a diversity factor would help the simulation. If enough diversity of the object is within the perception space, it could trigger its presence in the perception space". We entirely share this opinion and we will look into improving the classification algorithm as such.

6.1.4 *Improved computation of distance*

This goal is about improving the computation of distance to objects. When computing the distance to a certain object, we measure the distance from the agent's current location to the centre point of the object's bounding box. Therefore the distance might not always seem accurate because we are not taking into consideration the shape of the object. A possible solution would be to cast a ray from the agent's location to the centre of the object's bounding box and compute the distance to the closest intersection point (where the ray intersects the object). This way we could obtain a more accurate distance.

6.1.5 *A more granular perception of entities*

In the current implementation we have used a less generic definition of the SSM spaces. Future work could broaden them up to provide a classification closer to the ones defined in the theoretical framework. For example, an object in the Perception Space can be perceived as different entities depending on the distance to the agent. Given a hammer, from the furthest perception distance (X1) the agent would

perceive it as an object; from a closer distance (X2), the agent would perceive it as an object certain kind of shape while from a more closer distance (X3), the agent would perceive it as hammer. So, an object should be perceived in different ways based on the agent's proximity.

6.1.6 *Improved API*

In this implementation to retrieve data from the API a third party service has to query the endpoint every time it needs updated data. This creates unnecessary overhead both for the framework and for the service. To improve this, we plan to look into alternatives, for instance an API based on publish/subscribe. With this communication pattern we would allow service to express their interest in contextual changes based on some rules they define. For example context changes of a certain entity with a given ID, context changes within a certain set, etc. Whenever one of the registered events occurs, the API would broadcast the event and every service registered for that type of event would receive the notification. With this pattern we can minimize the overhead on both the API's and the third party service's side. Moreover, the delivery of the notification would happen real-time, the service being able to take action at the same moment a certain context change has occurred.

We are also interested in evaluating if there would be value for third party service to dynamically manipulate some attribute through the API. For example, an external service might automatically turn up the light if the agent walks into a certain room.

6.1.7 *Improved ContextClient*

The advantage of having implemented the context client as a web page, is that no additional software installation is required. It can be accessed from any browser, on any platform. The problem with the current implementation is that it does not reflect the new information immediately after a change in the context occurs. It only does so every second. To improve refresh time, we though of using Web Sockets[2]. The WebSocket specification defines a full-duplex single socket connection over which messages can be sent between client and server. This means that instead of refreshing the context client every second, the ContextViewServer could push up to date information towards the context client, whenever available, through an open web socket connection.

2 http://www.websocket.org

6.1.8 *Improved context information*

Many participants have pointed out that more context data should be monitored, besides visibility and proximity. To mention some of them: temperature, on/off state, orientation of certain objects (i.e. a computer screen might be close but you might be viewing it from behind), etc. We agree that for a complete context aware system some of these attributes have to be monitored by the framework. Another attribute useful context information we are interested in monitoring is the distance between objects. At the moment we only monitor the distance of objects relative to the agent's position. Knowing the distance between certain objects might be an important information for building business logic. Of course, adding more context information means to have it available through the API as well.

6.1.9 *Evaluate performance of the classification algorithm under heavy load*

This goal is about evaluating how a very large number of tracked objects affect the performance of our classification algorithm. During the evaluation process we have built a couple of simulations using the EgoSim framework. Within a simulation we have tracked around 20 objects at a time. With this number of tracked objects, the simulations ran smoothly providing up to date SSM sets throughout the lifetime of the simulation. We plan on setting up a simulation where we monitor a large number of objects to see how that affects the performance of the classification algorithms, hence the simulation overall. We think there should not be much of a difference as we only take into account the objects currently in the agents field of vision which are usually constant in number through the lifetime of a simulation. But we leave this conclusion to be drawn once this research has been conducted in future work.

6.1.10 *Improved interaction between the agent and the environment*

We have received some interesting improvement ideas from the user feedback. To give a better feedback for the simulation user, we might display a relevant icon when the agent is targeting a certain objects which is in the ActionSpace to denote that the object can be interacted with. This enhancement could improve the simulation user's experience.

Next, we are interested designing a generic USE action when interacting with objects. Therefore, instead of taking a default action when the agent tries to interact with an objects, the simulation could display a list of actions available for that specific object; the list of

actions would depend on the current context. For example, trying to interact with a mobile device could provide two actions: *pick-up* and *interact*. The *pick-up* action would simply pick the object object up, while the *interact* action would allow interacting with the virtual objects within the mobile device.

In this version of the framework the only default action we have provided is pick-up/drop-down. Some of the participants have mentioned other useful default actions like pull/push for drawers, open/-close for doors, drop object at current location, turn on/off for switches, lamps, etc. All of these actions broaden the interactions the agent is enable do with the environment. We will plan on implementing these default actions.

In the current implementation we have used a deprecated CharacterControl[3] class from the jMonkey framework to help us in with the agent's movement. As the CharacterControll class became deprecated, we plan on upgrading to the BetterCharacterControl[4] class. This will solve some of the agent navigation issues encountered during the evaluation process.

6.1.11 *Integration with a Virtual Reality (VR) kit*

This is an idea we have got from one of the participants. Integrating the simulation with a VR kit would create allow for maximum immersion of the simulation user into the simulated environment. On such VR kit is the Oculus Rift[5] VR kit. The community of jMonkey Engine has already made some advances in integrating it with the Oculus Rift SDK. We are really interested looking into this aspect in future work, as we believe it has to offer a lot of benefits for how the simulation user experiences the simulated environment.

6.1.12 *Create an EgoSim IDE*

The jMonkey Engine SDK is free and open source. To create an integrated development experience for the system designer, we have the option extending the SDK and creating an EgoSim IDE of our own, making the configuration process easier and providing EgoSim specific design options. This would require a large amount of work. First because we need to dig deep into understanding how jMonkey Engine is implemented. Moreover, we need experience in developing

3 http://hub.jmonkeyengine.org/javadoc/com/jme3/bullet/control/CharacterControl.html

4 http://hub.jmonkeyengine.org/javadoc/com/jme3/bullet/control/BetterCharacterControl.html

5 http://www.oculusvr.com/

plug-ins for the NetBeans platform, as the JME SDK is developed on top of NetBeans. At the moment this is not our biggest priority, but we are not excluding the possibility of looking into this aspect in the future.

6.2 CONCLUSION

In this thesis, we have designed and developed EgoSim, an open-source simulation environment for mobile context-aware system design. The framework is meant to be used by system designers to produce simulations of their target egocentric system. The resulting simulations are body-centric, meaning that the simulation user interacts with the simulation environment by controlling a virtual avatar. As the agent interacts with the environment, physical objects and devices in its surroundings are classified according to the SSM context model. This data is available through an API to be accessed by third party service and to be visualised in the ContextClient.

To help gather the requirements of our system, we have imagined an end-to-end scenario of how the finished system would be used by end users. This scenario was approved of by Thomas Pederson, one of the experts in the egocentric interaction paradigm. Aided by our goals 1.4 and requirements 3.1, we have decided that a modern game engine is the best technological choice to base our design upon. Therefore, the final architecture of our system is based on the architecture of modern game engines. Although we would have liked to follow an iterative design process, due to the high amount of research work and technical tasks required by this project we were constrained on trying to build the right system from the begging; hence, we took on a high risk of failure.

As a retrospect of the design process, if we have had more resource available, we would have had followed an iterative design process with 2 iterations as follows. For the first iteration, using the requirements we came up with, design an initial prototype. Evaluate the first prototype with a group of experienced ubiquitous system designers and developers. Using this feedback design a second prototype, the final design of our system.

We have implemented our design on top of the jMonkey game engine, which we have found ideal for research purposes as it is: open-source with a large and active community, well documented, purely written in Java and works with 3D models produced by third party modelling tools (i.e. Blender, one of the most popular open source 3D modelling application). Using EgoSim we have developed two prototypes, used in the evaluation process. The first one had a 3D model

built pragmatically, for the initial testing phase of our system; the second one was a simulation of an assisted living facility, using a 3D model created with Blender.

To evaluate our work, we have recruited 17 participants, all with strong software engineering skills. First they have evaluated the two aforementioned prototypes, followed by the implementation of a simulation from scratch. All participants have found both the framework and the simulations useful, highly usable and responsive. Only a little more than half the participants found the classification process to be reliable. The classification performed well in most cases, but it gives inaccurate results in edge cases. We have provided details on how to improve the classification process in the future work 6.1. This should be the first improvement taken care of in the near future of the project.

We have captured the list of goals in the future work as "issues" in the source code repository of our project `https://github.com/ksza/EgoSim`. This way, if anyone is willing to contribute to the project, immediate work items are already created and can be addressed.

The evaluation process has concluded that we have successfully met the four goals (1., 2., 3., 4.) we have set out in the begging of this project and that we came up with useful requirements (3.1.2, 3.1.3) which we have successfully implemented into a highly usable system.

Part I

APPENDIX

A

USER GUIDE

In this chapter we will walk the system designer through the steps required to set up a simulation from scratch. It also covers how to interact with the environment from the simulation user's point of view. The user guide is written by example, using a concrete environment model – the childrpoof.blend model which was used in the evaluation). To set up a simulation based on a different model, simply use the other model and take particular configuration actions as required by your use case.

A.1 PREREQUISITES

To get started, you will need to get your hands on 3 gems:

1. The jMonkey Engine 3.0[1] development platform, a NetBeans based IDE.
2. The EgoSim[2] framework from the project's git repository.
3. The example simulated environment's model[3].

Unzip the JMonkleyEngine Platform and run it. On the first run you will be asked to configure the working directory. Please do so!

From the File menu, select the "Import Project > From ZIP..." option. Point to the location of the EgoSim zip you've recently downloaded. This will import the BodyCentricSim project in your workspace.

A.2 IMPORTING THE ENVIRONMENT MODEL

Next, you will import the environment model into the project. Expand the "Assets/Scene" folder. Create a new folder, "Scenes/childproof". To create a new folder, from the "File" menu select the "New File..." option. In the dialog, select "Other" and "New Folder", as depicted in Figure 40.

1 http://hub.jmonkeyengine.org/downloads
2 https://github.com/ksza/EgoSim/archive/master.zip
3 http://karolyszanto.ro/MastersThesis/evaluation/media/models/childproof.blend

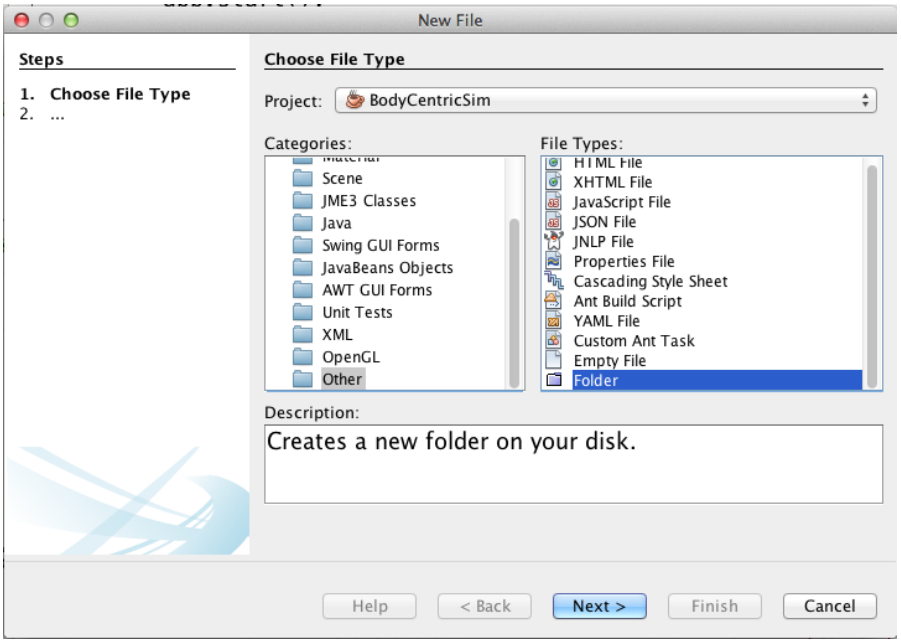


Figure 40: Create a new folder within the Scenes folder

Copy the "childproof.blend" file into this folder. Once you see the file in the IDE, right click it and "Convert to j30 Binary". This convert it to jMonkey format.

A.3 SETTING UP THE SIMULATION

In the "Source Packages" folder, under the "dk.itu.bodysim" package, create a new class, ChildProofApp. Make this class extend EgocentricApp. In the implementation of the "createEnvironmentScene", simply write:

Listing 3: Loading a j30 environment

```
return new GenericEnvironment("Scenes/childproof/childproof.j30",
    "ChildProof", assetManager);
```

Next, create a main method for your class:

Listing 4: Starting the app

```
ChildProofApp app = new ChildProofApp();
app.start();
```

The ChildProofApp class should look similar to:

Listing 5: Full class implementation of a new simulation created with EgoSim

```
package dk.itu.bodysim;
```

```
import com.jme3.scene.Node;
import dk.itu.bodysim.environment.GenericEnvironment;

public class ChildProofApp extends EgocentricApp {

    @Override
    protected Node createEnvironmentScene() {
        return new GenericEnvironment("Scenes/childproof/
            childproof.j30", "ChildProof", assetManager);
    }

    public static void main(String[] args) {
        ChildProofApp app = new ChildProofApp();
        app.start();
    }
}
```

Now your set! Inside the ChildproofApp hit the SHIFT+F6 key combination. In the settings window just start it up with the default configuration. This will start up the simulation! In here you can control the avatar, move around the environment, NOT walk through walls and objects. The EgoSim took care of this for you.

You will notice that you see the environment from a child's perspective: the height of the agent is smaller.

At this point, you notice the agent can't interact with any objects. That's because none of them have been augmented with context data! You can notice this by opening, on your second display, the ContextClient[4] in a browser.

A.4 IDENTIFYING OBJECTS TO MONITOR

In the assets, right click the "Scenes/childproof/childproof.j30" file and select "Edit in SceneComposer" as illustrated in Figure 41.

4 http://localhost:8182/context/view/set?name=all

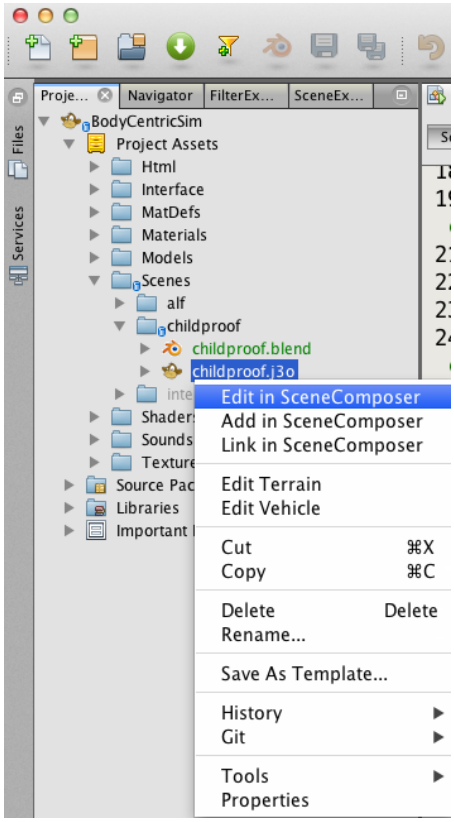


Figure 41: Edit in Scene Composer

To see the environment, you'll have to "turn on the lights", as depicted in Figure 42.

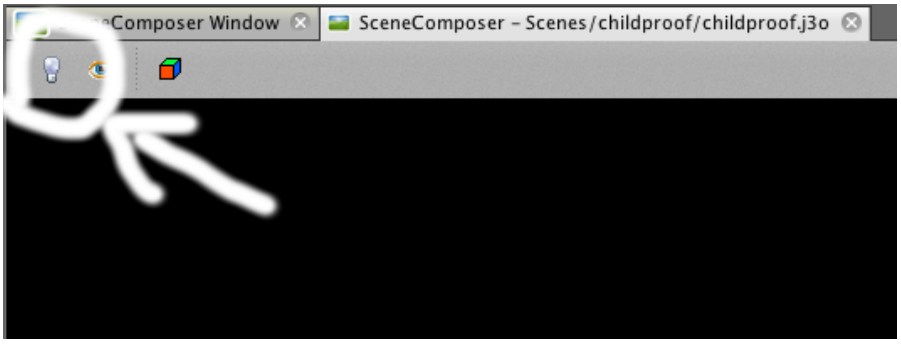


Figure 42: Turning on the light in Scene Composer

Suppose that the objects you want to track in this scenario are the Outlets and all the small objects that might be inserted into them; in this case, the pen on the living room table. These objects are highlighted in the screen-shot from Figure 43.

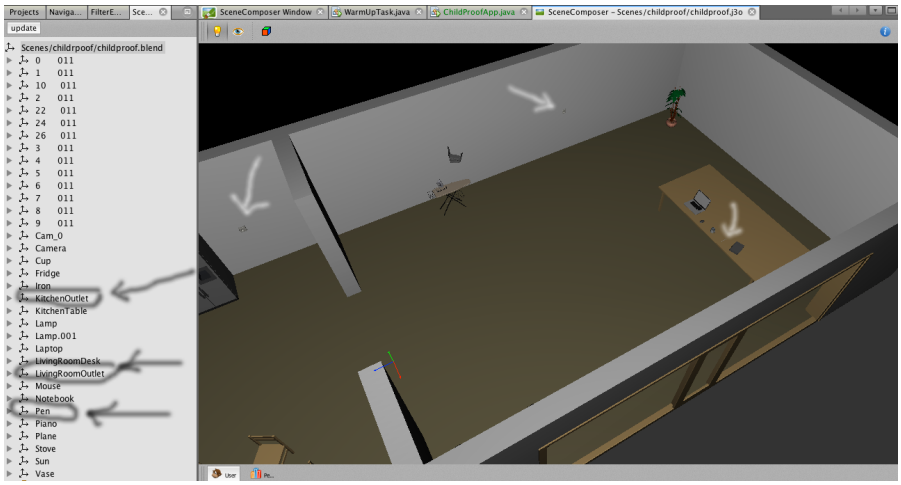


Figure 43: Example objects to identify in the Scene Composer

Identify the objects in the "SceneExplorer Window" on the left or by RIGHT-CLICKing on them in the "SceneComposer". Either way, as you identify them, they get highlighted in the "SceneComposer" on the right, as depicted in Figure 44.

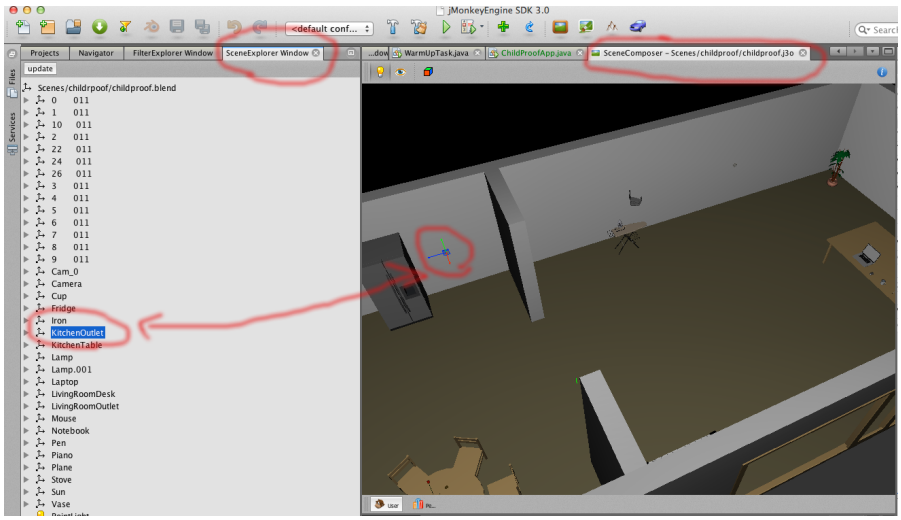


Figure 44: Scene Composer

To augment the objects with context data:

1. Right click the object in the "SceneComposer Window" and select the "Add User Data" as illustrated in Figure 45.

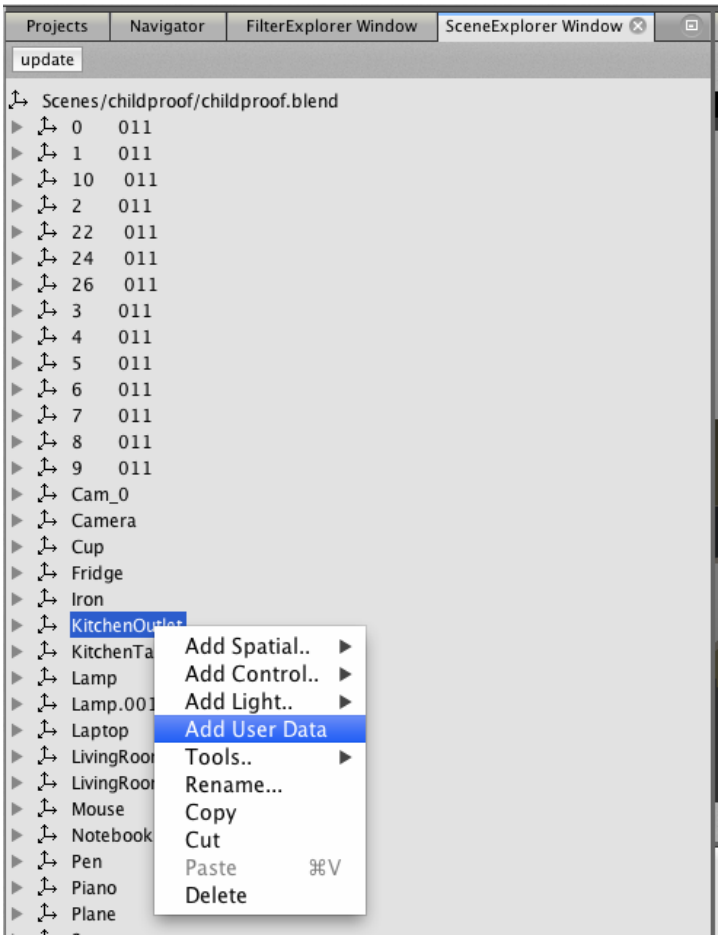


Figure 45: Add User Data action

2. In the name field enter enter EGOCENTRIC_CONTEXT_DATA. From the drop-down select "Custom" and choose "dk.itu.bodysim.context.EgocentricContextData". This will bring up the configuration form depicted in Figure 46.

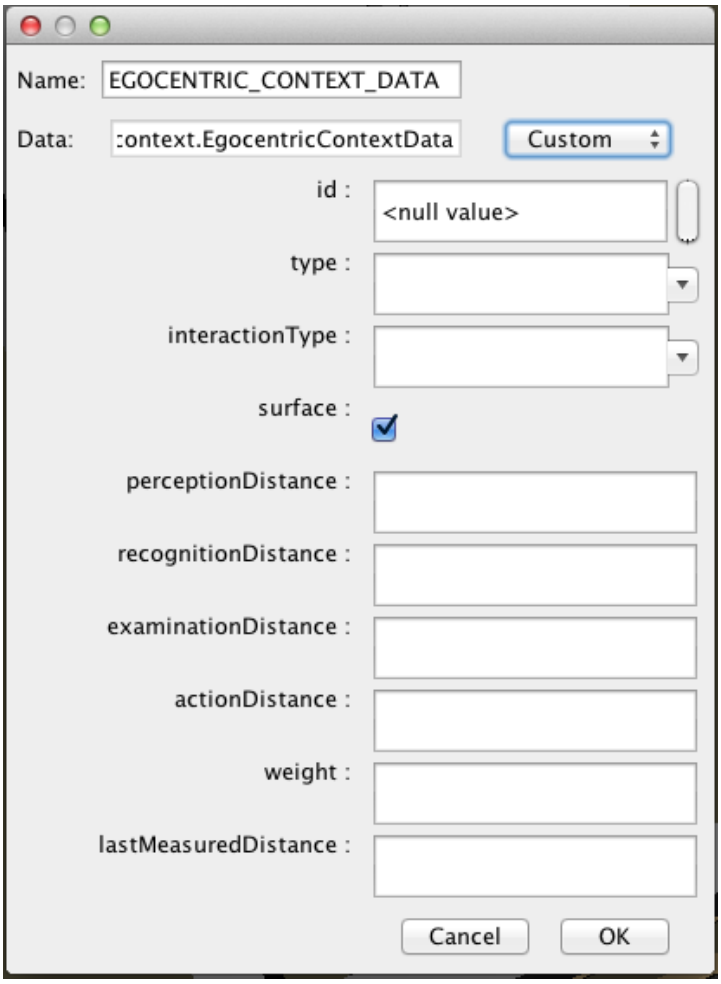


Figure 46: Egocentric Context Data Config Form

3. Configure the ID with a unique name (e.g. KitchenOutlet for the outlet in the kitchen) and the "interactionType" with "CUSTOM" for Outlets (otherwise, the agent will be able to pick it up) and "PICK_UP" for the Pen.

4. The rest of the parameters will take meaningful default values. Although, to fine tune the classification, the option to configure them is there.

5. Once done, hit the OK button.

After you have configured all the objects you want to track, from the File menu select the "Save All" option.

You can close the SceneComposer now.

A.5 RUNNING THE SIMULATION

Run the simulation once again. You can notice in the ContextClient[5] how the objects you've augmented, get classified in the SSM sets.

During the simulation, you can also access the API[6] endpoint which provides the data in JSON format.

Now you are ready the explore the simulated environment! You can move around the environment, pick up the Pen and interact with the Outlets.

A.5.1 *Controlling the agent*

You should see the world from a first-person perspective. Whatever entities you see in front of you are categorised to be "visible" entities. This context information together with the agent's distance to each object, helps in building up the other sets. The distance is measured in World Units (WU). This is a subjective interpretation of the distance in real-world metrics, the researcher decides upon at the time of building the environment.

To move around please use the standard first-person key mappings:

- W -> step forward
- S -> step backward
- A -> step left
- D -> step right

You can also move the mouse to look around you. The middle of the screen contains a cross, this is used to point at objects. When you wan to interact with an object, point at it and hit the LEFT MOUSE BUTTON. In order to be able to interact with the object, the entity representing it must have the following characteristics:

- must be part of the World Space (must carry egocentric context information)
- the agent must be within reach of the object (it must be in the Action Space)

At the moment the framework only supports picking up objects, and does not support custom interactions. To pick up an object the agent must be able to lift it up; the weight of the object has to be at

5 http://localhost:8182/context/view/set?name=all
6 http://localhost:8182/context/api/set?name=all

most the maximum weight the agent can carry.

Once the object has been picked up, you can carry it around the environment. To use the object to interact with other objects, you can either hit the RIGHT MOUSE BUTTON and the object will be automatically put back to its initial position (no matter how far you are) OR you can point to another monitored object and click the LEFT MOUSE BUTTON. Again, the object you want to interact with must be in the Action Space. In this iteration, picked up objects can interact only with surfaces, onto which they can be placed onto. For instance you can move a pen from one table to another, but you won't be able to pour water from a glass into a glass – you will get a system message.

At the moment, carrying objects around is the only way the agent can interact with the environment and modelled entities. Future development targets to implement interaction with mediators and virtual objects as well.

B

EVALUATION SCENARIOS

In this chapter we detail the scenarios we have imagined for our test subjects to guide them in evaluating the EgoSim framework. The first scenario has the purpose of making the user familiar with the framework. The second scenario was imagined for the simulation user role, where the subject is evaluating a simulation built using our framework. In the third scenario we have imagined a problem which a system designer can solve using the EgoSim framework.

We have built the first two prototypes into executable binaries for three major operating systems: Mac OS X, Windows and Linux. The builds are included with the thesis in the *EvaluationPrototypes.zip* archive. They are also available at `http://karolyszanto.ro/MastersThesis/evaluation/prototypes/`.

Videos of running the three scenarios are available; see Appendix D.

B.1 SCENARIO 1: WARMUPTASK

This first scenario is meant to make you familiar with the simulator's concepts and interactions. The environment is made up by two tables; the first table has two items on it: a pen and a small statue. The other table is empty. All entities in this scenario can be interacted with. A screenshot of the running simulation is depicted in Figure 47.

A recording of the running scenario is available; see Appendix D.

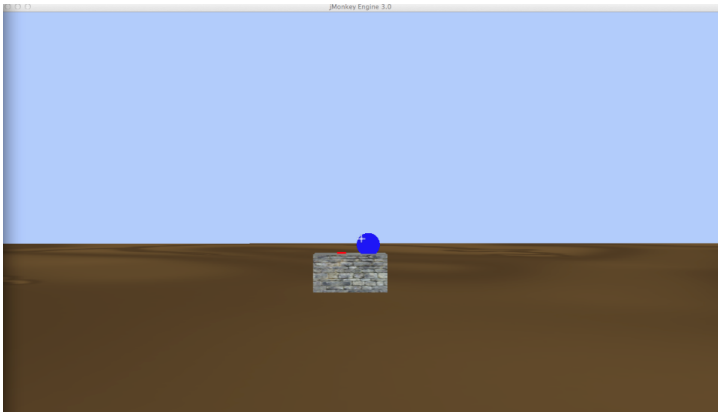


Figure 47: Sceenshot of the Warm-up simulation

Your task is to walk around and interact with the environment while observing the SSM spaces in the ContextClient. N.B.: The client is automatically refreshed every second. As part of the actions you carry out, please try out the followings:

1. Move far away from the first table (facing the table), until in the perception space contains only Table1 (it might contain Table2 as well if it's in your visual range). Slowly move towards Table1. At certain distances the PerceptionSpace, RecognizableSet and ExaminableSet will be populated.

2. Try interacting with the pen while it IS NOT part of the Action-Space. You are informed that the agent is too far.

3. Try interacting with the pen while it IS part of the ActionSpace. The pen gets picked up.

4. Move with the pen towards Table2. While Table2 IS NOT in the ActionSpace, try to interact with it. You are informed that the agent is too far.

5. While Table2 IS in the ActionSpace and with the pen picked up, try to interact with the table. Table2 is a surfaces and the pen can be placed onto it.

6. While not holding anything in your hands and while Table2 IS in the ActionSpace, try to it up. Table2 is too heavy for the agent carry.

7. Move towards the Statue until it's in the ActionSpace and try to interact with it. Interacting with the Statue requires a different implementation than the standard piking up object which is not implemented in this iteration.

8. Pick up the pen. With the pen picked up and the statue in the ActionSpace, try to interact with the Statue. You can't, because

Statue is not a surface. Future releases will handled this more complex scenario as well.

When you arrive to this point, you have successfully performed the WarmUp Task.

B.2 SCENARIO 2: ASSISTED LIVING FACILITY

An assisted living facility (ALF) is a housing facility for people with disabilities. These facilities provide supervision or assistance with activities of daily living (ADLs) and monitoring of resident activities to help ensure their health, safety, and well-being. Basic ADLs consist of self-care tasks, including: dressing, eating and feeding, bathing and showering etc. Constantly monitoring and predicting the activities of ALF residents is imperative for software services in the ALF.

For this task, please imagine yourself as the researcher of a monitoring system for ALFs. The system has to monitor the surroundings of the resident in real-time and provide this information to third party software service. One example of such a service is the "ADL Assistance Service" (ADL Service). This service detects whenever the resident is close enough to everyday objects to start interacting with them. If the object is usually used in a daily living activity, the service asks the resident, by means of the smart watch she/he's wearing, if she/he needs assistance with this activity. In case of a positive answer, a caregiver is immediately sent over. Please note that the ADL Service is purely conceptual; it is not implemented neither is its implementation target of this task! I am referring to it to exemplify the usability of the SSM the in various situations throughout the simulation.

You decided to design your system based on the egocentric interaction paradigm, as it can categorizes all the objects of interest around the human agent. Based on these sets, you can further develop software services to assist the residents in their ADLs.

In this task you are given a ready made simulation of the system for such an environment. You will be simulating a resident's interaction with the ALF by means of the virtual agent, completing a series of activities the resident does throughout his morning routine.You will be evaluating the usability of the simulator, the responsiveness of the ContextClient, the correctness and usability of the SSM sets, as well as how intuitive is the interaction with the environment. The screenshots in Figure 48 depict the object you will be asked to interact with.

A recording of the running scenario is available; see Appendix D.

The story unfolds with the following steps:

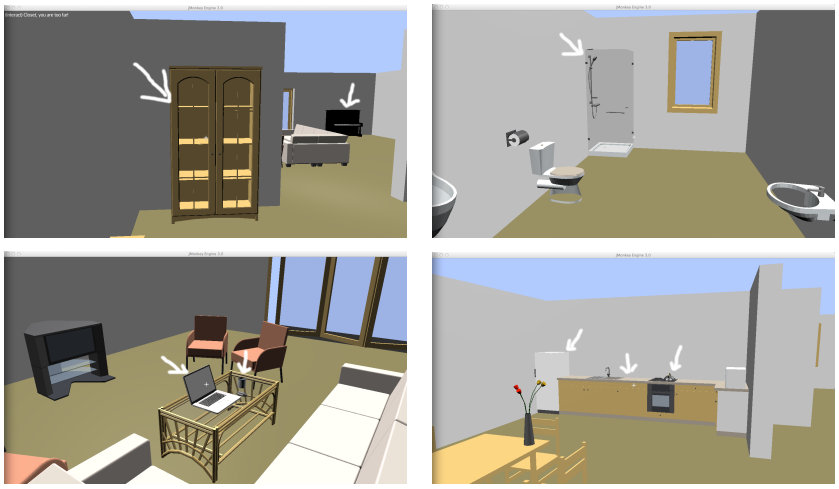


Figure 48: Screenshots of the running Assisted Living Facility simulation, highlighting objects the simulation user can interact with.

1. You wake up in your bedroom. You just got off your bed; now walk towards the Closet in your bedroom. If you get close enough, the Closet will be added to the ActionSpace. Take note of this in the ContextClient. At this point, the ADL Service should ask the resided if she/he needs assistance with "getting dressed".

2. Next, got into the bathroom. Approach the Shower, when close enough, it will be in the ActionSpace. Again, the ADL Service would detect a highly possible activity and would asks the resident if she/he needs assistance with "takings a shower".

3. Go to the living room, pick up the cup and go to the kitchen. Place it near the Stove (on the Sink counter-top), pick up the coffee Pot and try pouring coffee into the cup. This kind of action is not implemented visually (pouring coffee), but you will see that the system knows exactly what you are doing (if the volume of your computer is turned one, you might be able to hear the audio feedback as well)! You will notice that the series of actions in this step are possible only if the items you are interacting with are in the ActionSpace.

4. Approach the Fridge. When close enough, it will be in the ActionSpace. The ADL Service would ask the resident if she/he needs assistance feeding.

5. Try interacting with the laptop to read your email (if the volume of your computer is turned one, you might be able to hear the audio feedback as well). The action is not visually implemented, you will get a system message instead!

6. As a last step, walk around the apartment and observe how various objects get categorized, added/removed from SSM sets, as they enter/leave the view sight oh the agent, and as the agent gets closer/further from them.

There's an easter egg hidden somewhere around, have you found it? If no, then try interacting with the piano. Interact with it a few times :).

When you arrive to this point, you have successfully performed the task.

B.3 SCENARIO 3: CHILDPROOF

The hypothetical problem we are facing as part of this task is that families cannot make their homes secure enough for their children. To provide a solution for this problem, there's a need for a system to constantly monitoring the objects a child should keep away from and should not be interacting with. Based on this context information, we can further build various software service to secure the house from the child's actions. One example of such service is the "Secure Outlets Service" (SOS). This service has to detect whenever a child is approaching an outlet with a small object, in which case it should shut down the electricity switch for outlets, preventing the child from the possibly of getting electrocuted.

For this task, please imagine yourself as being the researcher in charge to develop this system. Once again, you decide to design your system based on the egocentric interaction paradigm, as it can categorizes all the objects of interest around the human agent. In order to validate your design, you decide to simulate the system first.

In this exercise, you will be provided with the virtual model of the simulated environment in the Blender[1] format. Blender is a free and open-source software for creating 3D virtual models, animations and more.

Using the EgoSim framework, you will augment this model to monitor the objects required by your system. Afterwards, based on the frameworks API, you will write the condition for the SOS to shut the electricity down in the outlets.

A recording of sequentially executing the steps in this scenario is available; see Appendix D.

1 http://www.blender.org/

Next, please follow the steps in the User Guide A to set up a new simulation.

After you have successfully set up and ran the simulation, as part of the interactions you do with the environment, pick up the Pen from the table and move with it close to one of the outlets, so that the Outlet is in the ActionSpace. When you arrive to this point, you have successfully performed the task.

C

SOURCE CODE

The source code is included with the thesis in the *EgoSim_SourceCode.zip* archive. You can also find the source code of this project on GitHub at `https://github.com/ksza/EgoSim`. The on-line repository also has an active list of issues `https://github.com/ksza/EgoSim/issues` representing available and active tasks.

D

VIDEOS

Three videos are included with the thesis in the *Videos.zip* archive. They are also available at http://karolyszanto.ro/MastersThesis/videos/. There is one video for each evaluation scenario, sequentially following the steps each participant had to complete. In each video, on the right side we have recorded the running simulator while in the left side we have recorded the ContextClient reflecting the current SSM context model of the system.

You can closely follow how entities get classified as the agent is interacts with the environment. Moreover, you can notice what the agent can and cannot do according to the current context: cannot interact with an item that is to far (the simulation gives a message), is an object is in the Action Space it can be interacted with, there is no default implementation for custom or combined interactions but you can notice that the system knows exactly what the agent is doing and displays proper messages.

WARMUPTASK.MOV is a recording of running the simulation in the first evaluation prototype as described in Appendix B.1. On the right side you can see the running simulation where the agent is interacting with the environment, while on the left side you can see the changes in the SSM context model as reflected in the ContextClient.

ALFTASK.MOV is a recording of running the simulation in the second evaluation prototype as described in B.2. On the right side you can see the running simulation where the agent is interacting with the environment, while on the left side you can see the changes in the SSM context model as reflected in the ContextClient.

CHILDPROOFTASK.MOV is a recording of following the steps described in the third evaluation task, in order to create a new simulation using EgoSim, as described in Appendix B.3. We have recorded the actual steps of implementing a simulation using the EgoSim framework. Besides following the steps in the description, at the end of the video we have demonstrated how you can extend the embedded support for combined interaction. We have simply displayed a message using EgoSim's notification manager.

E

EVALUATION RESULTS

The raw evaluation results are included with the thesis in the *EvaluationResults.zip* archive, in various formats (.csv, .xls, .pdf). They are also available at http://karolyszanto.ro/MastersThesis/evaluation/results. This appendix present a formatted version of the raw evaluation results. We have promised our participants keep their identity undisclosed: "As a participant to this study, you will be treated anonymously, your personal identity and personal data will remain undisclosed!". Hence, we have removed the questions and answer about personal data.

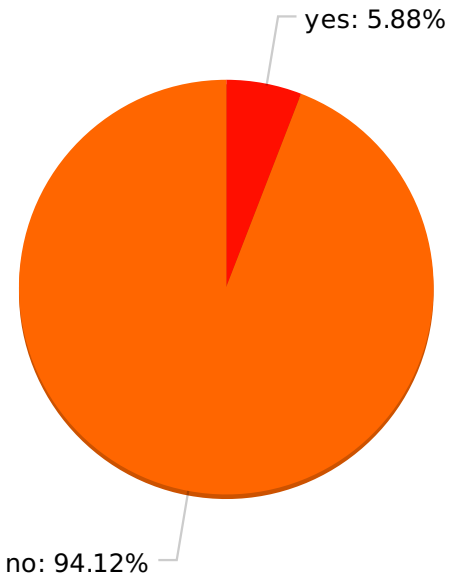
The entire documentation the participants were exposed to can be accessed at http://karolyszanto.ro/MastersThesis/evaluation/index.html.

E.1 TASK 1: ASSISTED LIVING FACILITY

E.1.1 *Question 0*

Prior this evaluation, were you familiar with concepts of the egocentric interaction paradigm or the SSM?

Yes	No
5.9% (1)	94.1% (16)



E.1.2 *Question 1*

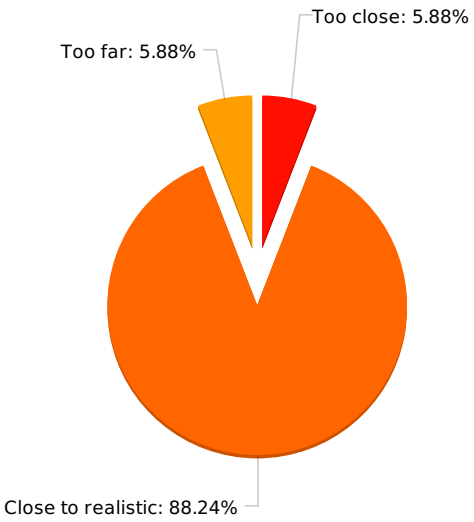
Did you find this environment a good simulation of a home?

Yes	No
100.0% (17)	0.0% (0)

E.1.3 *Question 2*

How did you find the ActionDistance – the distance from which the agent is able to interact with objects?

Too close	Close to realistic	Too far
5.9% (1)	88.2% (15)	5.9% (1)

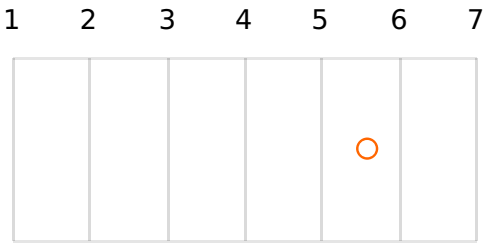


E.1.4 *Question 3*

How intuitive did you find interacting with objects?

The scale goes from 1 *(Not intuitive at all)* to 7 *(Highly intuitive)*.

(1)	(2)	(3)	(4)	(5)	(6)	(7)
0.0% (0)	0.0% (0)	11.76% (2)	11.76% (2)	23.53% (4)	11.76% (2)	41.18% (7)



E.1.5 *Question 4*

Were all the objects correctly classified in the perception space?

Yes	No
58.8% (10)	41.2% (7)

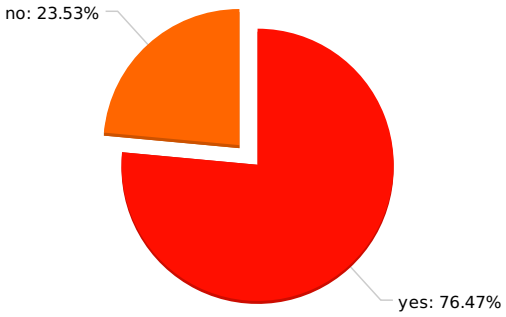
If No, describe at least one in which one or more objects were not correctly classified

- If I am in the corner of the kitchen facing the bedroom I can see the corner of the closet but it does not appear in any other space apart from the world space. And I can use the mouse to try and interact with it and it says ”Closet, you are too far!” so maybe it should have appeared in the perception space at least.
- The living room table while standing in the bedroom. It could ”see” the cup and the laptop, but not the table.
- Standing behind the couch, it only identified the laptop, not the closet, even though it was visible. If I moved a little back, it identified the closet too.
- the action space is sometimes too determined by where the cursor cross is, and not whether the object is in front of you or not. e.g., placing the cursor just on the side of the laptop makes the laptop be outside the action space. moving the cursor over the laptop makes the laptop enter the action space. the laptop should be in the action space also when the cursor is just beside it.
- I was standing in the kitchen door. The laptop and cup were in the perceptive space although they were kind of hidden by the sofa and the TV although more visible was not.
- The table and chairs in the kitchen – they are not classified in the perception space.
- There were objects in the apartment which were not classified at all: toilet, window, chairs...
- When standing next to the piano I think the TV should have been in the perception space.

E.1.6 *Question 5*

Was the ContextClient displaying the updated context information of the system fast enough to reflect the surroundings of the agent?

Yes	No
76.5% (13)	23.5% (4)



If No, why?

- The ContextClient update rate was decent for the ADL service but might require some improvement for other applications.
- A tricky situation would be objects suddenly appearing in the perception space and being used shortly thereafter. For example, a phone may ring behind the user and the user instinctively turns and picks it up (use). This example would be solved if the perception space is not limited to the visual stimulus, however it illustrates some exceptions that might cause the low update rate to be a problem.

E.1.7 *Question 6*

Do you think the SSM would be the right model for the ADL Service?

Yes	No
100.0% (17)	0.0% (0)

E.1.8 *Question 7*

Do you find this simulation useful over implementing the system directly into a real system?

Yes	No
100.0% (17)	0.0% (0)

Why?

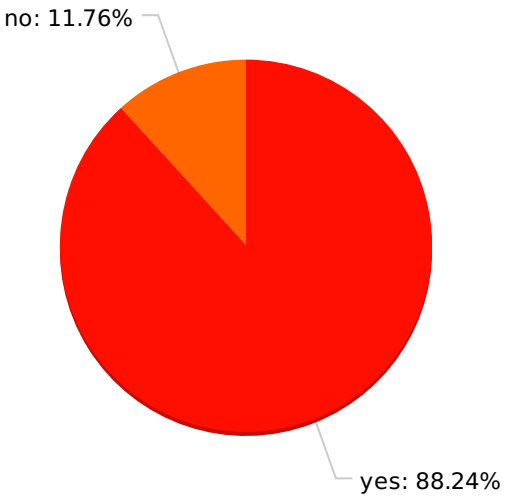
- The simulation could provide a cost effective way of gaining valuable insight into the design of an Assisted Living Facility before such a facility is constructed. The location of various objects around the room and distance between them could be determined using this simulation so as to maximize the efficiency of an ADL Service
- Simulations are always good for better understanding of the real systems. In this particular case, a simulation is a must. A every day life activity has a lot of interactions, and some are necessary (taking pills), and other are harmful (forgetting the stove turned on).
- I think it is very useful especially for testing the implementation of a ADL Service or other similar services for other environments. It is not needed to implement the real system to know how the service works if you can simulate it, one can detect bugs and malfunctions in the system before implementing it in a real environment. It is also cost effective.
- Because it offers a controllable environment to test the concepts and calibrate the initial assumptions of how the system should be designed before creating the actual devices.
- Because of the cost. You can implement it basically for free. Afterwards, you can use the simulation to do a cost estimate of the implementation into a real system.
- I think such a system must be validated before being deployed to ensure the safety of the people involved. Also, I think it would be a lot more cost-effective as it would help in discovering faults early on, before something is implemented (therefore significantly cheaper to correct)
- Gives you a perception of how people interact with the objects.
- Lower costs in case of trial and error.
- Helps in the real implementation, like finding the best location for sensors.
- Faster feedback loop.

- A simulation is a reliable and cheap proof of concept for me.
- It allows the discovery of corner cases.
- As most simulations, it helps in better understanding of the system.
- Same reasons as the ones highlighted in the "Introduction" section.

E.1.9 *Question 8*

Did the simulation run smoothly?

Yes	No
88.2% (15)	11.8% (2)



If No, describe a situation it did no perform well.

- There was a glitch when I couldn't move forward when getting out of bed. I had to move in a different direction and after that come back and approach the closet.
- the simulation window run on the external screen was placed in the middle of the screen as default and could not be moved. in order to save screen space (enable to have the task description on the side for instance) it would be good to be able to move the simulation window around.

E.1.10 *Question 9*

Please leave in this last field any feedback that comes to your mind! (ideas for improvements, bugs you might have found, things that you liked, etc)

- It was not clear to me if the elements in the action space are influenced only by the proximity of the agent to them or also by what elements are in the selected space. The reason I am thinking of this is because I can imagine objects which should be only handled if the agent is currently holding another object. For example, the ADL service could generate a warning when the agent tries to handle a dangerous object (like a hot plate) without having some protecting item in the selected space like some gloves (this is not a very good example but you get the idea :))
- One more thing I thought about was taking into memory of the agent for the recognizable space. I assume in time, the dynamics of the recognizable space change and objects which at the beginning are only in perception space, once they have moved into the recognizable space, they will stay there as the agent can remember them and their position.
- Nice playing with the application :).
- I could not use the MyGame.app and had to run it from jMonkeyPlatform (system info: MOAC OSX, version 10.7.4).
- I think moving around was very close to reality and I liked the fact that it was aware of the obstacles. For example, you could not just move forward through the couch to get to the coffee table.
- The Piano playing actions were nice.
- integrate with a Virtual Reality kit
- create a MMO EgoSim app where users (not just programmers) can create their own environments => you get feedback on a larger scale, see what other kinds of problems could be solved using the egocentric paradigm
- I really like the interaction, but what it was not quite clear to me was what is the ”Last distance from agent” useful for. Some of the objects displayed there were not even in the perception space and still they had a distance. It did not know what I was supposed to do with the info or how to check it. Maybe a ”Current distance from the agent” would be more useful?

- there seems to be a discrepancy between how the action space is modelled when you drop an object vs. when you pick up an object: dropping an object on another object is possible from afar, but picking it up isn't. Example: while standing in front of the sink, holding the coffee pot, I could drop the coffee put to the left of the sink but then not pick it up directly afterwards because it was regarded to be too far away. picking and dropping should demand the same distances, or?
- You can pick up the stove
- When you put the coffee pot down, it looks like it's in the stove not above it
- The refresh is not very accurate, the application runs smooth, good interaction.
- I liked the interaction with the piano
- I like the idea, but I would not implement it in a home where the elderly lives on its own. I think the ADL Service is more suitable for a retiring home where there is also human personnel which can intervene in exceptional cases.
- Change the cursor to reflect when an object is in the Action-Space.
- I consider the condition for an object to enter the perception space to be too weak. A user cannot identify an object if only a "pixel" is within his field of vision. Humans are built to recognize patterns, therefore a pattern-oriented metric could be used. For instance, the introduction of a diversity factor would help the simulation. If enough "diversity" of the object is within the perception space, it could trigger its presence in the perception space.
- Given the high system requirements I would have expected a better graphics/game engine.

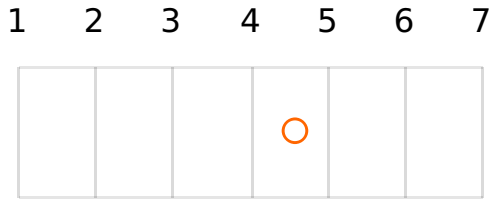
E.2 TASK 2: CHILDPROOF

E.2.1 *Question 0*

What level of understanding do you have at the moment about the SSM?

The scale goes from 1 *(Not at all)* to 7 *(Expert)*.

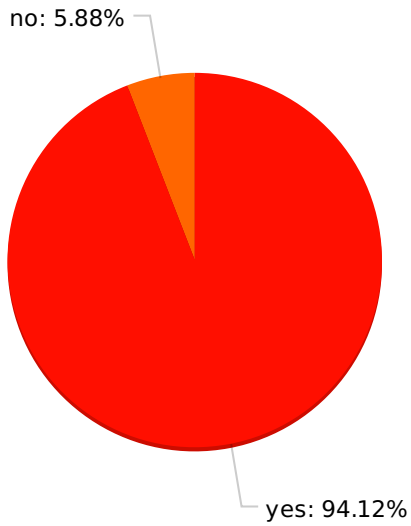
(1)	(2)	(3)	(4)	(5)	(6)	(7)
0.0% (0)	5.88% (1)	11.76% (2)	23.53% (4)	41.18% (7)	11.76% (2)	5.88% (1)



E.2.2 *Question 1*

Were you able to finish this task?

Yes	No
94.1% (16)	1 (5.9%)



If No, what difficulties have you come across that prevented you from finishing it?

- The objects I augmented with egocentric context data didn't show up in the spaces visualized in the client. I did notice some changes in the shading among the visualized spaces in the client, as I moved about, indicating that it might just be a

matter of forgetting filling in a field somewhere as to the name of the objects or something like that. An indication of that things were actually classified correctly even if I couldn't see it is that It was possible to pick up the pen.

E.2.3 *Question 2*

Did you find this framework useful for simulating the situative space model?

Yes	No
100.0% (17)	0.0% (0)

Why?

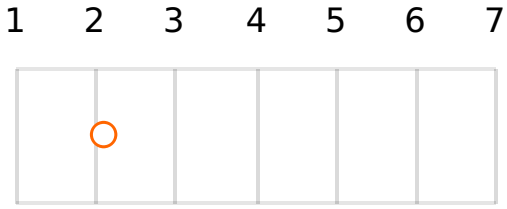
- Objects can be easily decorated with context information and interacting with them can be tested quickly because the framework takes care of basic interaction and movement in the space
- The simulation is very easy to understand and the increased usability makes it useful to prove the concept of how Context aware applications can be built.
- Very easy to set up and test a system behaviour where basic interactions are involved
- because with a few easy steps I was able to model the desired environment and got a notification api for free!
- well, just to walk around and see how objects were reclassified based on my orientation is worth something. it would have been nice to have the spaces visualized as spaces though, with animations that allow you track when objects transition from one space to another, and not as just as plain table entries.
- One can interact with designed objects.
- It was easy to use and It has a lot of possible utilities
- They can be close to realistic.
- Ease of naming objects and performing basic pick_up interaction.
- Because I was able to complete the task and it involved setting up an unfamiliar to me. Besides, the presentation made me curious to try and implement functionality.
- It is very easy to import a "scene" and augment the objects with different information and interaction types.

E.2.4 *Question 3*

How easy/hard did you find the process of augmenting the objects to monitor?

The scale goes from 1 *(Very easy)* to 7 *(Very hard)*.

(1)	(2)	(3)	(4)	(5)	(6)	(7)
6 (35.29%)	41.18% (7)	11.76% (2)	0.0% (0)	11.76% (2)	0.0% (0)	0.0% (0)

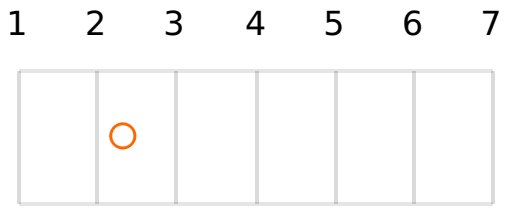


E.2.5 *Question 4*

How easy/hard did you find the usage of the framework as a whole?

The scale goes from 1 *(Very easy)* to 7 *(Very hard)*.

(1)	(2)	(3)	(4)	(5)	(6)	(7)
23.53% (4)	41.18% (7)	23.53% (4)	0.0% (0)	11.76% (2)	0.0% (0)	0.0% (0)



E.2.6 *Question 5*

What other parameters, if any, would you find useful for the system to monitor, besides proximity and visibility?

- Historical information about the agent: what actions has he performed so far, objects with which he has interacted with so far, etc
- How long an action can take. For example, a person can stay on the bad for ten hours. If the time is exceeded, a nurse should be informed.
- Depending on the application, it might be useful to monitor temperature, weight, etc.
- Previous interactions
- temperature, on/off state (outlet voltage), age (number of days the milk has)
- audio aspects of for instance the perception space.
- temperature, humidity, light.
- Temperature
- Orientation of certain objects. A computer screen might be close but you might be viewing it from behind.
- A useful parameter would be something linked to the user's perception of the existence of an object. For example, when you sit next to an open fire, you might not have a visual stimulus (let's assume you have your eyes closed) that confirms that objects presence; but rather you perceive its warmth. Thus, "perceptibility" could be defined as a measure of the subjects awareness of an object as any combination of feedback from his or her senses, in regard to said object.
- In the case of the childproof system: the height difference between the child and the outlet.
- weight, size, shape

E.2.7 *Question 6*

What information, besides what's already served, do you think that should be available through the API?

- List of objects the agent has interacted with... or maybe last X objects
- I'd make available all the parameters, see previous answer.
- the distance between all objects, so that we can have more than a visual map. Distance from edges.

- some constraints that can be set for a specific object
- Depends on who is using the API. If it is a software programmer, no. If it is a user with technical abilities who has to simulate different scenarios it is close to enough.
- On/Off state for objects; for example a door can be locked or unlocked, a faucet might be on/off.

E.2.8 *Question 7*

What types of interaction, besides pick up/put down, do you find absolutely necessary to make the simulator more realistic?

- Dropping an object at current location
- If a person takes his pills, it can not be simulated right now. The fact that an electronic device is turned on/off.
- Push, pull action (for outlets for example).
- Custom actions depending on the type of object.
- Possibility to increase speed for interactions
- There could be state change notifications. Or even surroundings aware interaction: light goes on in a room you are entering, etc.
- different types of usages of the objects (for the piano, the laptop,...)
- touch / press / push / pull
- turn off, on (electronics, water tap, shower)
- simulation of mediators (keyboards, displays, etc.)
- for the simulation that was enough
- Drag, rotate
- for me it's realistic enough
- I think pull, push for objects
- if a standard on/off state could be added, a turn on / turn off action might be useful
- A generic USE action would help is many situations as many objects that are interacted with might have a main USE method. pickup/putdown could be specific implementations of use.

- An interaction leads to an operation. A possible redesign of this feature would be adding an intermediary abstraction that is related to the scope of the operation on the object. An internal operation could be implemented in the object that is manipulated (use the pen to write), while an external interaction would result in an operation that changes the object's context in relation to other objects (moving the pen around).
- Any form of custom action that depends on the object you interact with. Some of the objects in Task1 had this sort of behaviour: you were able to play the piano, you were able to read mail on the laptop, you were able to pour tea

E.2.9 *Question 8*

Please provide any ideas to improve this framework! (also point out bugs you might have found, things that you liked, etc)

- Not sure how it is implemented right now but I can imagine such a simulation being a multiple step process: 1. the simulation being built by some developers. 2. After it has been developed, it is given to some ”experts” which calibrate the simulation so that it better simulate the reality of the situation it was designed for. For this they would need to easily change (possibly even at runtime) some attributes of the EgocentricContextData like: perception distance, recognition distance, examination distance, for any of the augmented objects. The framework should allow for this... I think.
- I liked the framework. Easy to use.
- Extend it to provide a more complete range of actions.
- It is very easy to use and identify the monitored objects.
- No bugs found in the evaluation.
- I really liked the idea and other than the fact that I did not find how I can use lastMeasuredDistance meaningfully, I have no other observations. Really nice work!
- physics model, no gravity,
- there are some SSM entity type naming issues that could be improved. e.g. the term ”surface” might not be appropriate for all objects that can accept an object (e.g. the sink).
- regarding bugs, I couldn't complete task 2. it is probably not a bug but the lack of providing some info that was needed. Maybe the instructions for task 2 could be a notch clearer (e.g. with

examples for how the field should look when filled in for the sockets and for the pen).

- first impression is that it's easy to use and friendly. The Add User Data part without help would have been a bit tricky, if that can be made easier. SceneExplorer Window a bit too heavy. Other that that, great.
- Trim the context variable for space EGOCENTRIC_CONTEXT_DATA
- Didn't noticed any bugs and I liked the fact that I had a different perception when I was a child than when I was an adult
- No bugs! As a first time usage it went fine!
- A potential weakness of this framework would be the lack of information regarding the relationship between objects. In this example, the child may pick up and pen and attempt to stick it the outlet. Is is trivial to conceive a logic that may cause the power in the outlet to shut down, preventing the kid from getting electrocuted (pen is selected, outlet is in action space). However, if the kid releases its grip on the pen, the system will register that both objects are in the action space, thus turning the power back on. If the kid then touches the pen, it will be too late to react as the child is already in danger of electrocution. A possible solution to this scenario could be adding a space that describes the interaction between two objects. A flower pot and the table it is placed on interact, though not in a way that is relevant to a child's safety. However, a pen stuck in an outlet may cause an interaction where a specific handling is necessary. The situation where the outlet is not in the action space as some electrical appliance may already be plugged in should be considered. The system should not react to the pen holding child's proximity to the already used outlet. The presence of an object in the action space can thus depend on how the said object interacts with other objects. Various object states may be needed in order to correctly simulate the above mentioned scenario
- It is very easy to augment objects with data. Would be nice if people could declare their own type of interactions.

E.2.10 *Question 9*

Given the SSM sets, what should be the condition for the SOS service to shut the electricity down in the outlets? Feel free to write the condition in simple words, pseudo-code or any language of your choice.

- if (pen is in selected space) and (outlet is in examinable set) then shut-down electricity for current outlet

- If outlet in actionSet and I am a kid, shut electricity down. But a kid runs a lot, the electricity may be shut down very often for no good reason, but to change the condition to selectedSet may be too late.
- Height of the agent should be under a limit (to consider a child) and the proximity to the outlet is less than 5 cm. Turn them on when proximity is greater than 1 m.
- if ((actionSpace.contains(outlet)) and (selectedSpace.contains(pen)) outlet.shutdown()
- If a child is approaching the outlet and has entered the Action Space with an item in the Selected Space the power should be cut off. Of course it also might raise the question if it is necessary to have something in hand. If the Outlet is not used, I think that power can be cut off just if the child is in the Action Space.
- the child holds has picked up the pen and the proximity to the outlet is small
- distance to pen <= actionable distance of the outlet (pen is inside of the outlet's ActionSpace)
- if (pen in kid_human.hand) outlets.shut_down
- if pen is in the manipulated set and socket is in examinable set then turn off electricity
- shut down the power if the child is at 2 m from the socket
- If the user has something in hand (define small object) and moves towards the outlet, when reaching the limit of 1 meter and has the pen at hand, disable the outlet.
- When the child is in the Action Space with an object from the Manipulated Set, show an alert, play a sound and shut the electricity.
- assuming we can set callbacks on any set modification:
- The pen has to be in the selection space and the outlet in the use space.
- in case the distance between the child and the outlet is smaller than a given delta, the electricity should be shut down.
- dangeous_objects = set(has_sharp_edges(obj) for obj in all_objects).
  If any(dangerous_objects.intersection(selectedSet)) and any(outlets.intersection(actionSpace)):
  shut_down_electricity()

E.2.11 *Question 10*

Can you think of any system that could use the SSM sets and classification algorithms as building blocks? Please describe it shortly.

- A friendship network with several kind of relationships and different possible interactions accordingly.
- Any home to help out with certain defined tasks e.g,: elder's home, persons with disabilities to help with daily activities or in case of urgencies. Hospitals. Public institutions guide a person from entrance to the needed assitance
- Assistance for the visually impaired.
- This system could be implemented for people with disabilities, especially visually impaired persons, to offer a augmented representation of their environment.
- A tourist map that is context aware and can recommend places to visit that are near the current location, that are for example still open (maybe it's night, you don't want recommendations for a zoo), etc...
- It would be fun to model the International Space Station with SSM. Think no gravity (physics engine for jMonkey?), resource management.
- future bodyworn computer systems, starting with today's cellular phones.
- provide help for people with disabilities
- Any monitoring system, learning systems.
- A kindergarten construction. It can be used to simulate all the possible places a kid can explore and possible things he can do.
- security application: auto-lock my computer screen when it's disappears from the intersection of my Examinable Set and Action Space (or perhaps just action space)
- Automatic light switching based on the position of the inhabitant of a home (not motion-based). This can be generalized to a system that defines custom operation modes of devices in a home, depending on what the user is doing.

BIBLIOGRAPHY

[1] V. Reynolds, V. Cahill, and A. 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*. ACM, 2006, p. 1.

[2] D. E. Knuth, "Computer Programming as an Art," *Communications of the ACM*, vol. 17, no. 12, pp. 667–673, December 1974.

[3] T. Pederson, L.-E. Janlert, and D. Surie, "A situative space model for mobile mixed-reality computing," *Pervasive Computing, IEEE*, vol. 10, no. 4, pp. 73–83, 2011.

[4] M. Lewis and J. Jacobson, "Game engines in scientific research," *Communications of the ACM*, vol. 45, no. 1, p. 27, 2002.

[5] M. Shantz, "Designing a pc game engine," 1998.

[6] [Online]. Available: http://hub.jmonkeyengine.org/wiki/doku.php/jme3:terminology

[7] M. Weiser, "The computer for the 21st century," *Scientific american*, vol. 265, no. 3, pp. 94–104, 1991.

[8] [Online]. Available: http://www.cs.berkeley.edu/Weiser/bio.shtml

[9] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a better understanding of context and context-awareness," in *Handheld and ubiquitous computing*. Springer, 1999, pp. 304–307.

[10] T. Pederson, L.-E. Janlert, and D. 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*. ACM, 2010, pp. 755–758.

[11] I. Armac and D. Retkowitz, "Simulation of smart environments," in *Pervasive Services, IEEE International Conference on*. IEEE, 2007, pp. 257–266.

[12] O. Alliance, "Osgi service platform core specification release 4 (august 2005)."

[13] G. Erich, H. Richard, J. Ralph, and V. John, "Design patterns: elements of reusable object-oriented software," *Reading: Addison Wesley Publishing Company*, 1995.

[14] K. Bouchard, A. Ajroud, B. Bouchard, and A. Bouzouane, "Simact: a 3d open source smart home simulator for activity recognition with open database and visual editor." *International Journal of Hybrid Information Technology*, vol. 5, no. 3, 2012.

[15] [Online]. Available: http://jmonkeyengine.org

[16] J. Bruneau and C. Consel, "Diasim: a simulator for pervasive computing applications," *Software: Practice and Experience*, vol. 43, no. 8, pp. 885–909, 2013.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Education, 1994.

[18] M. Martin and P. Nurmi, "A generic large scale simulator for ubiquitous computing," in *Mobile and Ubiquitous Systems: Networking & Services, 2006 Third Annual International Conference on*. IEEE, 2006, pp. 1–3.

[19] J. J. Barton and V. Vijayaraghavan, "Ubiwise, a simulator for ubiquitous computing systems design," *Hewlett-Packard Laboratories Palo Alto, â AI HPL-2003-93*, 2003.

[20] 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," in *Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005. First International Conference on*. IEEE, 2005, pp. 60–69.

[21] M. Cohn, *User stories applied: For agile software development*. Addison-Wesley Professional, 2004.

[22] [Online]. Available: http://agilemanifesto.org/

[23] G. F. Coulouris, J. Dollimore, and T. Kindberg, *Distributed systems: concepts and design*. pearson education, 2005.

[24] T. Kindberg and J. Barton, "A web-based nomadic computing system," *Computer Networks*, vol. 35, no. 4, pp. 443–456, 2001.

[25] R. Kusterer, *JMonkeyEngine 3.0 Beginner's Guide*. Packt Publishing Ltd, 2013.