A Distributed Context Simulation Component

Izhar Ahmed Khan
Abstract

Mobile devices with access to large numbers of sensors with internet access move forwards the development of intelligent applications towards new shape of ubiquitous applications. In order to create such applications we need to be able to do simulations to test and deploy. Current simulators do not permit this since they are centralized and the information is not shared globally. Therefore we cannot use them to test application built on distributed sensor information. I selected Siafu as the simulator component. In the next step, the simulator was customized according to the requirements of the project. There are different possibilities to achieve this task, but a simple GUI is made to control the simulator. The end result is a complete architecture for simulating context aware scenarios. The implementation is tested by running the simulator and dumping the context data into the PGRID overlay. For future work, implementing proximity estimation between the agents will be a good idea and can be interesting as well.

Keywords: context, siafu, ubiquitous computing, pervasive applications.
Foreword

I would first like to thank Almighty Allah for the blessings. I would love to thank my beloved parents who always supported me during my studies. I would also like to thank my tutor and supervisor Jamie Walters for his support during the entire thesis. And last but not the least, I like to thank Prof. Theo G. Kanter for the opportunity to contribute to the research work at Mid Sweden University.
# Table of Contents

Abstract ................................................................................................................................. ii

Foreword .................................................................................................................................. iii

Terminology ............................................................................................................................... vii

1 Introduction .......................................................................................................................... 1
  1.1 Background and Problem Motivation .............................................................................. 2
  1.2 Overall Aim ...................................................................................................................... 3
  1.3 Scope ............................................................................................................................... 4
  1.4 Concrete and Verifiable Goals ......................................................................................... 5
  1.5 Outline ............................................................................................................................ 6
  1.6 Contributions .................................................................................................................. 6

2 Theory ..................................................................................................................................... 7
  2.1 Context Definition ............................................................................................................ 7
  2.2 Context Awareness ......................................................................................................... 8
  2.3 Context-Aware Computing ............................................................................................. 10
  2.4 Connecting the Physical World with Pervasive Networks ........................................... 12
  2.5 Simulation ...................................................................................................................... 12
  2.6 Computer Simulation .................................................................................................... 13
  2.7 Context Simulation ........................................................................................................ 14
  2.8 Different Context Simulators ......................................................................................... 16
    2.8.1 DiaSim ...................................................................................................................... 16
    2.8.2 SimuContext ............................................................................................................ 17
    2.8.3 A Generic Location Event Simulator ........................................................................ 18
    2.8.4 Siafu ....................................................................................................................... 20

3 Methodology .......................................................................................................................... 24
  3.1 Requirements .................................................................................................................. 24
  3.2 Design Pattern ................................................................................................................. 25
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.3</td>
<td>Identifying the Simulator</td>
<td>25</td>
</tr>
<tr>
<td>3.4</td>
<td>Customization</td>
<td>26</td>
</tr>
<tr>
<td>3.4.1</td>
<td>Why Customize?</td>
<td>26</td>
</tr>
<tr>
<td>3.4.2</td>
<td>How it should be Customize?</td>
<td>27</td>
</tr>
<tr>
<td>3.4.3</td>
<td>Which Parts Needs to Be Customized?</td>
<td>27</td>
</tr>
<tr>
<td>3.5</td>
<td>Scenario Based Simulation</td>
<td>28</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Why Did I Choose Siafu?</td>
<td>29</td>
</tr>
<tr>
<td>4</td>
<td>Implementation</td>
<td>31</td>
</tr>
<tr>
<td>4.1</td>
<td>Simulation Modules</td>
<td>31</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Why These Modules Are Important?</td>
<td>32</td>
</tr>
<tr>
<td>4.1.2</td>
<td>How Layers are Created?</td>
<td>32</td>
</tr>
<tr>
<td>4.1.3</td>
<td>How Connect to Simulator?</td>
<td>32</td>
</tr>
<tr>
<td>4.2</td>
<td>Creating the Simulation</td>
<td>33</td>
</tr>
<tr>
<td>4.3</td>
<td>Customizing the simulator</td>
<td>38</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Why is this Needed?</td>
<td>38</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Implementation</td>
<td>38</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Adding New Layer</td>
<td>39</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Putting Layers into the Overlay</td>
<td>40</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Output Formats</td>
<td>40</td>
</tr>
<tr>
<td>4.3.6</td>
<td>Defining Agents and persistence in the Overlay</td>
<td>42</td>
</tr>
<tr>
<td>4.3.7</td>
<td>Modifications to Siafu’s Source Code</td>
<td>42</td>
</tr>
<tr>
<td>4.4</td>
<td>Distributed Overlay Implementation</td>
<td>43</td>
</tr>
<tr>
<td>4.4.1</td>
<td>How it Works?</td>
<td>43</td>
</tr>
<tr>
<td>4.4.2</td>
<td>Which Parts of Distributed Overlay Are Used?</td>
<td>43</td>
</tr>
<tr>
<td>4.4.3</td>
<td>Modifications to the Source Code</td>
<td>44</td>
</tr>
<tr>
<td>5</td>
<td>Results</td>
<td>46</td>
</tr>
<tr>
<td>5.1</td>
<td>Context Simulator</td>
<td>46</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Evaluation of the Ability to Simulate</td>
<td>46</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Test setup</td>
<td>47</td>
</tr>
<tr>
<td>5.1.3</td>
<td>How Much Can I Simulate?</td>
<td>47</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Increasing Number of Nodes</td>
<td>48</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Memory Usage</td>
<td>48</td>
</tr>
<tr>
<td>5.1.6</td>
<td>Adding New Sensor and Layers</td>
<td>49</td>
</tr>
<tr>
<td>5.1.7</td>
<td>How Does it Works with Increased Layers</td>
<td>49</td>
</tr>
<tr>
<td>5.1.8</td>
<td>Anomalies</td>
<td>49</td>
</tr>
<tr>
<td>5.2</td>
<td>Simulator Behaviour in Respect to Overlay</td>
<td>50</td>
</tr>
<tr>
<td>5.2.1</td>
<td>Adding New Context Attribute to Overlay</td>
<td>50</td>
</tr>
<tr>
<td>5.2.2</td>
<td>The Response Times</td>
<td>51</td>
</tr>
<tr>
<td>5.2.3</td>
<td>Application Crash</td>
<td>52</td>
</tr>
<tr>
<td>5.3</td>
<td>Context Attributes</td>
<td>52</td>
</tr>
<tr>
<td>5.4</td>
<td>Customization</td>
<td>53</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Performance Comparison</td>
<td>54</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Performance over time degradation</td>
<td>54</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Accuracy</td>
<td>55</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Updation in Overlay</td>
<td>56</td>
</tr>
<tr>
<td>5.5</td>
<td>Evaluation</td>
<td>56</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Reliability</td>
<td>54</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
<td>58</td>
</tr>
<tr>
<td>6.1</td>
<td>Problem statements</td>
<td>59</td>
</tr>
<tr>
<td>6.2</td>
<td>The results</td>
<td>59</td>
</tr>
<tr>
<td>6.3</td>
<td>Future work</td>
<td>60</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>61</td>
</tr>
<tr>
<td>Appendix A: Project Flow</td>
<td></td>
<td>66</td>
</tr>
<tr>
<td>Appendix B: UML diagram of the proposed structure</td>
<td></td>
<td>67</td>
</tr>
</tbody>
</table>
A Distributed Context Simulation Component

Terminology

Acronyms

GPS  Global Positioning System
JAR  Java Archive
Java SE  Java Standard Edition
SimuContext  Simply Simulate Context
GUI  Graphical User Interface
API  Application Programmer Interface
DCXP  Distributed Context eXchange Protocol
UbiWISE  A Ubiquitous Wireless Infrastructure Simulation Environment
JDK  Java Development Kit
CPU  Central Processing Unit
Config  Configuration
CA  Context Awareness
2D  Two Dimensional
GC  Global Clock
GLES  A Generic Location Event Simulator
1 Introduction

Context awareness has become one of the most researched areas over the past few years. Mobile technology is spreading so rapidly as the world moves towards globalization. This thesis is about the representation of the real world entities and objects in a contextual way. Presentities and users interact with each other in a context network. There may be some association between these users and presentities which can give us the information of the current context of the entity.

Current mobile trends are moving towards a high speed internet, which pushes the technology development towards a new form of service called pervasive services and applications. Over the last few years the expression of interest in context aware applications has become very high as the people want to communicate with each other irrespective of time and place. Applications that provide context based services are of high interest in today’s ubiquitous environment. Many platforms and model architectures have been presented to deal with such situations but there are always some constraints to each solution. This thesis investigates different simulators in order to find a suitable component for the distributed overlay.

Such a real world environment for the context infrastructure can be of different types in order to show the real world presentities. In order to support a high range context aware environment we need to model the real world entities in such a way that leads us to seamless access to context experiences. This real world context environment can be a simulator which can handle a large scale of simulations having different types of context attributes and dimensions so as to monitor the environment. The work contained in this report has its focus on the creation of the simulation environment that can be later on integrated into the
A Distributed Context Simulation Component

1 Introduction

overlay architecture presented by the MediaSense research group in the Mid-Sweden University. As a part of the MediaSense project, this report has a wider picture of the solution having its main focus on the development of an efficient and reliable context aware simulation platform for this overlay that uses a distributed architecture.

![Context Aware Simulation Architecture](image)

Figure 1.1: Context Aware Simulation Architecture

1.1 Background and Problem Motivation

The penetration of context aware applications has been mirrored down by systems that do not have the capabilities to support frequent changes in the dynamic context. In the context aware environment there should be a mechanism which presents the real world entity in a form evolving presentity that changes frequently over time. Keeping in mind the availability of the applications, the development of a context simulator is a good idea to deal with the context applications in the distributed environment.
A Distributed Context Simulation Component

1 Introduction

We can state that the context of an entity is the surroundings or the situations of the particular entity, where in this case an entity can be a person place or thing which has some kind of association with its current environment. However this simple representation of this notion can be practically implementable on computers [2]. Hence the research has been moving towards context awareness; consequently context aware and pervasive applications have been developed today. More specifically, these applications make use of the sensors that are connected to certain devices which gather context information about an entity. This process can later be helpful in creating intelligent applications as well for example search results can be sorted on the basis of user’s location, interest based advertisements and many more.

The context data is normally acquired from the sensors which is then stored into a storage area typically a centralized storage allowing the developers to secure and control the data, for example IMS. But, in result this centralized approach can also be a drawback at the same time as the centralized storage can become the bottleneck of the architecture, and in case of failure whole network can be down. This approach can also induces the notion that in this narrow system of storage each party have to control its own data flow of context information having no feature of sharing it between the other parties [37].

However, context aware and ubiquitous applications are open to the environment every time allowing the entities to communicate in a peer-to-peer way, thus implementing a pure distributed architecture. For such dedicated architecture we should have context aware simulator that can simulates the presentities in a better way [37].

1.2 Overall Aim

The aim of this thesis is to contribute to the research initiative taken by the Sensible Reality group for the development of context aware applications to
enable context awareness in a distributed architecture. Therefore, the thesis has its focus on scenarios which require large scale context base simulations to test the distributed overlay [37] with the simulation data. The scenarios can include different real world models for example presenting the daily life persons in the form of agents. These agents will then have a certain context attributes like their position and their temperature etc.

The primary focus of the thesis is to create simulator component for the distributed overlay having certain type of representation of the real world entities.

Figure 1.2: Overview of the Challenge Task

The first task was to identify a context simulator that suits distributed architecture. Siafu was finally selected as a solution to the stated problem.

1.3 Scope

This thesis has a main challenge for the identification of a context simulator for the distributed overlay. A Context simulator can be of two types: a GUI using a visualizer and a GUI without using a visualizer.
Figure 1.2 illustrates the overview of the project that the simulator will generate the context data which will be then connected to the overlay. This thesis has its primary focuses on the simulation portion while the secondary goal is to integrate this simulator with the distributed overlay.

The resulting work did not aim at issues that can arise from my work concerning the context simulator. The thesis study delivers a design and implementation of a simulator component for the distributed context architecture.

1.4 **Concrete and Verifiable Goals**

The thesis work will be carried into a series of steps to accomplish all the requirements.

- Review context simulators.
- Select a simulator that best suits the thesis requirements.
- Study and analyse the selected simulator and perform simple implementation.
- Customize the simulator according to needs.
- Implementation of the context simulator in respect to the requirements.
- Create a GUI for handling the simulator.
- Review the overlay and its implementation.
- Perform an implementation of the context simulator with different scenarios.
1 Introduction

- Test and verify the results.

1.5 Outline

This thesis report is structured as follows: Chapter 2 addresses the theory and some of the related work, Chapter 3 consists of the Methodology, Chapter 4 has its focus on the Implementation, and Chapter 5 concludes the Results. Finally, Chapter 6 summarizes the conclusions arising from the work contained in this thesis and lift up possible areas for future work.

1.6 Contributions

The distributed overlay was created by other students.
2 Theory

This chapter presented the theoretical background and the work related to this thesis. It begins with the introduction of context aware systems and their applications. It also looks at different context simulators are also discussed in this chapter. In short this chapter includes the overview of context, the detail regarding context awareness, different context simulators, and related work.

2.1 Context Definition

The term context is commonly perceived as the surroundings of an entity. For example, the spoken words context can also be understand by the situation in which they are spoken. Research has been conducted to explain context in the field of computer science. Within the field of computer science the accepted definition of context was defined by Brown [1]. However the most accepted one is the concept stated by Dey and Abowd [2], which states that:

*Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.*

There are additional studies which define the context in a different way. Though the definition is quite broad but as a matter of fact it is very difficult to enumerate information in the field of computer science as information can be perceived of interpretation. An example of this can be a current mood of a person [34]. Therefore the context is defined as the information related to an entity,
while the entity can be a person, place or thing. For example, context information can be the current temperature, date of birth, geographical location, etc. Context information can be presented and modeled in many ways, like taking location as an example we can define this both in terms of a position in the form of longitude and latitude and in terms of a address as well. So this becomes a trivial problem when dealing with the context information, as this has to be generally presented in a universal form. The concept of this kind of context modeling has been studied in detail by [3], which states that two approaches can be of great benefit. One approach is based on the ontology and the second is an object-oriented approach. However there are many more approaches that can be helpful.

Context information has the property that it cannot come out by itself, but it has to be collected and gathered from someplace. Many approaches exists for this, for example a manual oriented approach in which the users defines preferences of their own choice. However common approaches that are used today for the context detection in an automated way is by making use of the sensor technology and the sensor networks. So because of the involvement of sensor technology in the context system sensor networks are of great importance in gathering the context data [33].

2.2 Context Awareness

Context awareness, or more precisely context-aware behaviour, can be defined as the consequence obtained by the use of context information. This can be applies to applications as the ability to settle in the behaviour depending on the existing state of the users. First application that was accepted as a context aware system was the Active badge presented by [4]. However in recent years there has been a significant development in the context aware study, resulting in
many pervasive and intelligent applications. Many definitions of context awareness exists but the most acceptable one is defined by Dey [2], which states that

A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task.

From the above definition we can conclude that context-aware computing is the development of clever and intelligent applications, which can modify their state and behavior according to the current situation or context. Over the last few years, mobile technology has spread very rapidly. Therefore we can define a mobile device as a truly pervasive as it moves and travel with the user. Many approaches and solutions exist that define context awareness through the mobile phones, for example GPS [5], Google Latitude which make use of the several sources of context i.e. both from available sensors, user profiles and existing hardware.

Figure 2.1: Context Aware Application [13]
An application is said to be context aware that alter its state according to the current context information. These applications can assist the user in many environments. According to Dey [2] the assistance can be of three categories, which are: tagging, presentation and execution.

An example of the presentation assistance is searches offer by Google\textsuperscript{2} on the basis of location [6]. It has the ability to show the search results to the user based on the current location.

One example of the category of execution assistance is an application that does an operation on the basis of user’s context information, like heater of the house turned on.

A tagging can be defined as the information acquired to be manifest in a manner that later retrieval can be simplifies. For example in addition to date and time, modern digital cameras have the ability to add GPS coordinates to the photographs. By this facility it gave the user a better way to sort the photographs on the basis of both location and date [34].

### 2.3 Context-Aware Computing

It has been a challenge for the distributed mobile computing to take advantage of the varying environment with new dimension of applications that are awake of the current context in which they are sprint. Such kind of context-aware applications become accustomed according to the need of use, the set of people nearby, devices that are accessible, as well as to other things that changes over time.

\textsuperscript{1}Google Latitude: www.google.com/latitude
\textsuperscript{2}Google: www.google.com
A system which holds such type of capabilities has the potential to inspect the computing atmosphere and respond to changes in the environment. Aspects of context that are important are: where you are, who you with, and what resources are in close proximity [9].

Context-aware computing is usually linked with the elements of the ubiquitous computing. However, context issues have always been important in the field of human computer interaction. Context based computing scale should be absolute to other applications as well not only to the ubiquitous computing [31]. Though location is a principal aptitude, applications that are location-aware it does not mean that they essentially detain things of interest that are portable or varying. In general, context-aware is more related to take account of nearby devices, lighting, people, noise level and many more [9].

Figure 2.2: A Context-Aware Computing System [9]
2.4 Connecting the Physical World with Pervasive Networks

The physical world presents an extraordinarily wealthy set of input modalities, motion, image, light intensity, heat, atmospheric pressure, vibration and many more. Conventionally, sensing and managing physical world means the deployment of profound set of engineered instruments and infrastructure to gather certain inputs [10]. Ubiquitous computing testbeds have gathered much of such kind of engineered data, but it needs some more practice to observe the physical phenomenon. System challenges are the most severe hurdles to ubiquitous computing. The elements of distributed system in large amount, limited physical connectivity to them, and the tremendous dynamic environmental conditions make this real world very difficult to plot on the contextual map. As the system is closely coupled to extreme dynamic physical world, these systems will suffer heavy dynamism [10].

2.5 Simulation

Simulation is used when we want to examine a branch of the real world which is unreachable, or which is hard to test with, or which changes over time very frequently. A simulation is the designing of a system’s model, and then the execution of that model on a computer to analyze the output of execution [11].
Simulation is the replication of some actual thing. Simulation can be used in many ways and contexts, like technology simulation to check the performance, testing, etc. Simulation is used to represent the eventful real variations and effects of situation and the courses of action. Simulation can also be used when the real system can be judged, because of several reasons like inaccessibility or some other reason.

### 2.6 Computer Simulation

In computer science, simulation has special meanings, the term “simulation” was used by Alan Turing [12] to describe the transition of a state when a universal machines has been executed. A computer simulation is an effort to replicate a situation or real life on a computer, so that the output can be analyzed to observe how the structure works [11]. By varying the variables of the system, assumptions and predictions can be made about the performance of the model. We can say that it is a tool that helps in creating an environment in
virtual of the real world or system. Computer simulation is a valuable part of the modelling process. For example, a network traffic simulator is a good example of the usefulness of computer simulations. There can be several different kinds of computer simulation, but the all of them have a common feature which is to present a model that closely resembles to the real world scenarios in which different real world parameters can be manipulated.

2.7 Context Simulation

The invention of well automatic mechanized aircraft with the use of glass cockpits has extensively broadened the potential capabilities of the aircraft but moreover alter the type of job done by the aircraft crew. In result of such advancements, various scenarios of the real world are become part of the simulation process now [32]. Demonstration and testing of the context-aware application is a very challenging task. Acquiring contextual information for such reason requires extra development effort. Context is gathered from the physical environment of the user or sensed from the computing environment of the user. Context-aware application’s logic is more than the traditional logic, as it also deals with the logic that acquires and process the information of the context which is coming from the context source. Therefore, facing the context sources has also been a challenge for the application developers. Presently, ubiquitous application or context-aware technologies are still in development and are not deployed widely yet [32]. Therefore, in order to develop context-aware applications, one may have to put some extra development effort for the implementation. The common context source structure (see Figure 2) is that it gathers unprocessed data from a source (for example sensor), then if needed does some manipulation or processing and then convert it to proper format for the context-aware application. [13]
Demonstration and testing context-aware application in a managed and controllable manner with respect to physical context sources can be a difficult task. Naturally, context information is very dynamic in nature as it changes over time very frequently. Therefore, fetching the similar context in related state is tricky. For example, having GPS position at the same mark can give different context data due to accuracy changing over time [13]. Software application testing is a hard and time costly process. In case of ubiquitous computing, the problems are worsening. Repeatedly exploration of multiple environments is not practical for the developer while running their applications. Modelling the context of the user in software is solution to this problem. However, if the complexity of the scenarios has to be modelled, a traditional approach will be not enough. So an intelligent model has to be implemented in practice to model the real world entities into contextual framework.

Thus the context simulator can be a Multi-Agent System where the agent suites collaborate to draw a contextual model. The system of Multi-Agent System is collected of intelligent agents that interact with each other [35]. These systems can be used to overcome the problems that impossible in practical or difficult to implement for a single agent or autonomous system.
2.8 Different Context Simulators

Pervasive computing applications include both software and integration anxiety. Several pervasive applications manage a diversity of entities gathering context information from the sensors and by triggering of the actuator reacting. Actuator triggering is presumed to alter the environment’s state or condition. Thus such kind of pervasive application development needs to address several issues for example, physical constraints, heterogeneity etc. Numerous programming frameworks and middleware have been suggested to make the pervasive computing application development easier [16,17,18]. However, they need a pervasive environment which is fully equipped to be tested and run for the application. Following are some of the simulators that are developed for pervasive computing applications.

2.8.1 DiaSim

DiaSim is a pervasive computing applications simulator, which is based on actuators and sensors [15]. The simulator is enriched with respect to the description of the high level target environment of pervasive computing. Such explanation represents the entity’s classes, related to an intended pervasive computing area whether software or hardware. Moreover, an emulation layer is generated by making use of the description of the environment to run the simulation programming framework and pervasive computing applications for the simulation logic development. This method makes it easy to run the application in the real environment. The computing environment enables the resulting pervasive simulation to test the program concept against different scope of scenarios that corresponds to description of the environment. Also the in this phase the simulation allows the system to be polished against the logic of program or entities of the environment.
2.8.2 SimuContext

This project presents a framework for context simulation, known as SimuContext [13]. Normally, context aware application developers wish to hide the internal sources of the context and treat them as a black box. The SimuContext framework assists the application programmers in developing and testing context aware applications by giving the context source’s behavior. SimuContext framework offers the following facilities to the application developer.

- Speedy testing and debugging of context-aware applications, including demonstration in controllable way.
A Distributed Context Simulation Component

2 Theory

- Creation and development of a more widespread realistic environment by making use of a huge number SimuContext sources having particular properties.

- SimuContext sources replacement by physical context sources having a sensible practical context model.

![SimuContext System Diagram](image)

Figure 2.6: Overview of SimuContext System [13]

Figure 2.6 shows the black box view of the SimuContext system. The developer can specify a context source in the configuration phase. These are context sources are then simulated which behaves as a context source in physical and thus can be associated to the context-aware application.

2.8.3 A Generic Location Event Simulator

From the recent years several projects has emerged middleware for context-aware applications [20,21,22,23]. These projects have a designed focus to manage and monitor real world events (e.g. location events) and to supply significant and appropriate event stream levels to the context-aware applica-
tions. This simulator [19] is a hybrid simulation model which is based on a process oriented behavior in terms of sensor model, physical movement and many other models. Until an event occurs, every locatable [19] has a constant behavior and a state change occurs upon event occurrence at distinct time stamps. These techniques have not been applied so far in ubiquitous computing, however in control systems these techniques are very admired. [19]

In this simulator, vehicles have been modeled as locatables, so approaches can be adopted from theory tribulations cruise control, steer-by-wire and brake-by-wire systems.

---

Figure 2.7: Generic Location Event Simulator’s Architecture [19]

Initial positions of all the locatables have been set by a State Transition Controller, which is also responsible for assigning task to them when their present job
has been done. This controller also manages the mechanics transitions and queuing model transitions. To maintain time synchronization among all the process, a Global Clock has been used, so a consistent event generation has been occur. The induction of this global clock plays a vital role in the simulator, because it may helps to distribute the simulation’s thread to avoid scalability problems. The simulation test results show that high volume of pluggable locatable behaviours is the right plan for simulating ubiquitous computing applications. Some more behaviour can be added for example tracking of the moving target, following fluid flow etc [19].

2.8.4 Siafu

The complexity related to assembling and processing contextual information makes testing ubiquitous computing applications hard. Furthermore, short of simulation tools and typical data sets makes the process even harder [24]. At the moment, context-aware applications testing are typically a hard task. One reason of such problem is the heterogeneity that exists in the sensors and data sources. Finally, raw sensor dimensions can be meaningful to the user and as a substitute we have to draw advanced rank concept from the information. For example, GPS measurements can be use to gather places that are carrying great weight to the user. [24] However, implication of these abstractions is a hard and a challenging task to evaluate. Moreover, sensing device prices have been decreased in the recent few years, but it is still costly to equip users with multiple sensors. These problems are not only related to context-aware systems, but general to all multifarious systems. So, to overcome this, researchers have come up with simulation tools that try to give a better solution of the problem. Siafu is a generic tool for simulating ubiquitous computing systems [24].

Using simulation in context-aware computing is fairly a new concept. Current research has its focus on simulating sensors at low level. For example, [25] has been used to simulate smart space sensor measurements. One of the other
directions of the recent research is to combine the physical sensor simulations with virtual reality. This kind of approach include the examples of UbiWISE [28] and QuakeSim [26], [27] simulators. Other main direction of current research is to make use of the multi-agent simulation. For example, the Generic Location Event Simulator [29] generates events dependent on context, by which the middleware functionality can be tested. Siafu is a different kind of simulator on two characteristics. Firstly, well established techniques have been adopted from the simulation field. Secondly, it has a design focus to be generic and can be appropriate in broad variety of scenarios.

To build a flexible simulator, the information sources have been separated from each other. Firstly, agent’s behaviour is modeled to agent model. Secondly, the random events that are generated in the environment, handled in world model. Lastly, the simulated context data is defined in the context model. A detail description of these components is explained below.
• **Agent Model:** In the simulator, agent model is considered as the opening source of information; the agent’s decision logic is based on this model. This model is responsible for decision making of an agent’s behaviour in the given current context. The agent context is provided by the context model, while information regarding the surrounding entities is given by the world model [24].

• **World Model:** After the agent model, next source of information is the world model. This model includes three portions: the simulating environment, the simulating places of interest and an event model that takes care of the global events. The world model uses a collection of images. These images consist of a map for the background and an alpha map. The alpha defines the routes where the agents cannot move, for example, walls. To ease the creation of world model, places can be created by using the templates. For a place, this template identifies three things. One, place properties at the start of simulation. Secondly, an event model has been specified which generates random new events. Lastly, the template identifies when and how the properties are changed by the place [24].

• **Context Model:** This model is considered as the final source of information, which is responsible for the management of context variables used in the simulation. For every variable of context, this model identifies the variable possible values, the model where the values have been simulated and the distribution of the values across the environment. An easy mechanism has been provided in the form of bitmaps to update the context variables. For the generation of new context variable, the developer requires to supply a map and a file calibration to the simulator. This information has been used by the simulator to measure the mapping that relates values of pixel with the particular values of the context variable in the simulation environment. The mapping is stored in a file once it is calculated. For example, let’s consider the temperature simulation in the
environment. For this intention, a cloud image could be generated by the simulation developer. Then, the simulator measures the mapping and on running the simulator the output contains the value of temperature at the agent’s position. The values of the overlay can be modified and the image map can also be replaced by the simulation developer [24].
3 Methodology

The task is to create a simulator component for a distributed overlay. There are numerous problems that need a solution in order to achieve the goal. This chapter will focus on these tasks and argue for the possible solutions of the problem.

3.1 Requirements

To reply the questions defined in the problem statement some context simulators are required to compare. Therefore the first task is to identify a context simulator that best suits the current architecture of the distributed overlay. This task can be achieved in many ways, for example several simulators can be selected on the basis of their performance. So a list of jobs to do is as follows:

- Review context simulators.
- Identify the best simulator by comparing different available simulators.
- Study and select a simulator with some simple test implementations in respect to the overlay.
- Customize the simulator according to the requirements of the project.
- Develop an interface for the simulator.
3 Methodology

- Study the distributed overlay implementations.
- Take the data from simulator and store it in the distributed overlay.

Given a generic test case, testing of any framework can be extended with a nominal quantity of work. By making use of a generic test frame it can be easy to deploy the conducted work. The first step in finding the simulator is to investigate different simulators, and check whether the design pattern resolve the problem or not.

3.2 Design Pattern

A design pattern is a generic way of solving problems within the software development. It mainly consists of four portions, name, problem, solution to the problem and consequences. Design patterns are normally separated into three groups, depending upon their aim of abstraction. These groups are behavioural patterns, creational patterns and structural patterns. Facade is an example of the structural pattern. By making use of the facade one can solves the complications concerned in using a specific subsystem by giving an interface, a facade. Subsequently it protects the developer or user from the details of the complex subsystem.

3.3 Identifying the Simulator

More and more projects are running on developing context-aware applications that helps to improve the lifestyles of human being. However, these application developments normally require equipped rooms and laboratories with sensor hardware. Subsequently, the primary goal of this thesis is to find a simulation
framework for testing context aware applications. To reduce the complexity, the implementation will be limited to domestic environments, although it should have the capability to extend the idea to a large context.

### 3.4 Customization

After the selection of the simulator the next important stage is to customize the simulator according to the needs. The Pgrid overlay used by the MediaSense needs to be tested with heavy amount of data so the simulator should be able to produce such kind of frequent context data. Secondly, the simulator component has the ability to simulate using both the visualizer and without visualizer.

#### 3.4.1 Why Customize?

Generally, it is obvious that all the applications not alike. Software applications more specifically developed in an out of box manner, so that new pieces can be develop or added later to form a system according to the needs. In case of context-aware application development, customization can be sometimes so important that an application module can depend on the customization. Once the software is deployed, we realize that some of the features are missing or alternatively we may feel that there are too many features and it would be better to have some modifications to the applications to make it work according to the requirements. The selected simulator needs to be customized because the thesis work involves only the contextual information regarding the presents. So to have the desired features in the system one has to customize the application in order to get the necessary results.
3.4.2 How it should be Customize?

After we have decided to customize the simulator, a question arises in the mind that how the simulator would be customized. The answer to this question can be sometimes simple, but it is rather a tricky step as software packages normally does not allow the user to make amendments to the system. The best thing about Siafu is that it is an open source simulator. It means we can customize the simulator and type of simulations according to our requirements. In order to customize the simulator we have different alternatives but the one which I have selected is to develop an interface that controls the simulator. The interface would be able to start up the simulation process by specifying the simulation folder to the context simulator. More features like number of agents, use visualizer can also be added.

3.4.3 Which Parts Needs to Be Customized?

It is understandable that open source systems offer many features to be customized. But it is up to developer on how to go on with the modification process. Customizing the software can be very tricky as the software should be understood first. After a thorough study of the context simulator (Siafu), I suggest following modifications to the simulator.

- Changing the context attributes.
- Changing the simulation map and places.
- Changing simulation scenarios with varying number of agents on different times.
3 Methodology

- Using the simulator with and without visualizer.
- Which contextual information needs to be retrieved?

3.5 Scenario Based Simulation

Relating real world entities into virtual world is significantly a hard task. To fully customize the simulator according to the requirements the simulator will be designed to do the simulation like a real world scenario, i.e. simulating every day of week. So the final picture becomes like this:

![Diagram showing the simulation process](image-url)

**Figure 3.1: The Final Picture**
The above Figure 3.1 shows the development modules of the thesis work. The first involves the identification and reviewing of different context simulators. The next stage is the selection stage in which the context simulator is selected according to the requirements. After the selection of context simulator, next stage is to study and implement the simulation in respect to project requirements. The final stage is to test and verify the simulator results.

### 3.5.1 Why Did I Choose Siafu?

Siafu was selected for this work because of the following features:

- It is written in java so there is no compatibility issue as the distributed overlay is also written in java.
- Context data can be visualized easily and demonstration of context-aware applications becomes easy.
- Siafu can be customized easily, moreover customized places and scenario objects can be simulated.
- Siafu can create its own context as well as it is also able to incorporate the one obtain by other sensors.
- Siafu can be used to produce simulations of type as small as office and as large as a city.
- Siafu is an open source simulator that is platform independent and can run on different machines having different operating systems.
- Siafu can simulate several context attributes like position, proximity information, hotspot coverage etc.
These features of Siafu are important because they have a significant impact on the implementation process. Siafu is an open source simulator which makes the customization process easy for the developers. Siafu can be use to simulate almost any kind of context which makes it important to be a part of context-aware computing. Furthermore, Siafu is platform independent and can simulate context with and without visualizer.
4 Implementation

Chapter 4 will first explain the adopted solution for the implementation and then discuss the proposed architecture of the project flow. The computer used for the application development and testing is an IBM Lenovo core i5 running Windows 7. The development environment used for the development is Netbeans 6.9 with the latest version of Java Development Kit. The subsequent subsections will discuss a detail explanation of, firstly, simple testing of the selected simulator and then the implementation. For further details see Appendix A, which includes the complete flow of the implementation.

4.1 Simulation Modules

Before we start creating the simulation certain modules needs to be understand first. In order to create a simulation in Siafu we need to define some modules first. Following are the important modules that need to be focused first:

- **Background and Wall Image**: A background image needs to be defined in order to build a simulation. This background image is the basis for running the simulator. Another important image that needs to be focus is the walls image. This image defines the areas where the agents are allowed to walk.

- **Places**: Places are another important module that needs to be addressed before the creation of simulation. Places can be defined in the world-Model class of the simulator. These places can be defined in Siafu by making use of images.
Implementation

- **Context Variables**: Context variables can be defined in the get set methods of the Agents. Context variables can be initialized by the ContextModel class which uses images as a source to define context variables.

### 4.1.1 Why These Modules Are Important?

The modules defined in section 4.1 are of great importance because these are the basis for simulations. In order to create a full fledged simulation these modules must be defined first. The background and walls image are the basis for simulation, Siafu use these images to create a simulation environment for the simulator. Places are of great importance as well, by defining places we can tell the simulator that where the agents will go in the simulation area. Lastly defining the context variable we tell the simulator that these images will be used in order to get the contextual information of agents.

### 4.1.2 How Layers are Created?

Siafu works with different layers. So, in order to create a simulation in Siafu we need to create some layers. These layers are created by following a particular directory structure. The main layers are the overlays, maps and places. These layers are collection of images files which are used by Siafu at run time.

### 4.1.3 How Connect to Simulator?

In order to run and make a connection with simulator, a TCP port has been used. The contextual information from the Siafu is gathered through this port. This port has been set to a port number 4444, but it can be change. Whenever the simulator is started this port connects directly with simulator and we can get access to context data from here.
4.2 Creating the Simulation

Creating a simulation in Siafu is relatively a hard task. For this purpose certain steps are needed to be followed.

1. Defining the environment
2. Behaviour Programming
3. Packing the Simulation

1) Defining the Environment: There are some image files that are needed for defining the environment, which are:

- Wall image and background image
- Places maps
- Overlays (context variables maps)

**The background Image** is the main environment image for the ground where all the things happen. This image is solely beauty surface and has no control in the simulation. However, the image of the walls identifies the places where the agents are able to walk, and also identifies the places where not, for example walls. To make a visual logic in the simulation, the background and walls should match. Using gimp’s tools the background can be created easily. Once this is done, the image has to be stored as background.png (Portable Network Graphics). This is now considered as the basis for simulation. One thing which is important to take care of is that all the images must have the identical dimensions. Now the next step is to identify the portions of the background image which are walkable. This can be done by creating a new image, on the basis of the background image. This new image will be black on the areas which are
A Distributed Context Simulation Component

walkable and white on the areas which are not walkable (walls). This new image should be stored in png (Portable Network Graphics) format with the name as walls.

![Background and Wall Image](Background.png) ![Walls.png](Walls.png)

Figure 4.1: Background and Wall Image

**Defining Places:** Places can be defined in the World model, but measuring the coordinates of places can be a headache. So this step can also be made easier by making use of the images to define the places. Now to define any type of place, a white image has been created, and a single dot (black pixel) has been put for every place. These places can also be defined at runtime of the simulation. Whenever an agent has been tell manually where to go, a new place is created.

**Defining Context Variables:** Context variables can be created by two ways, one by linking to the positions of the map and second is by using get and set methods in the agent itself. The final thing called the overlays, and would identify, for example the temperature, the humidity, the light intensity at every
location in the map. Context variable value can be retrieved at map point from the value of pixel. The translation is done on the basis of the config file of the simulation, and it can be of three types: Binary, Discrete and Real overlay. So it means that we have to create an image for every overlay.

![Images of context variables](image)

Atmospheric Pressure  Humidity
Light Intensity  Temperature

**Figure 4.2: Context Variables Example**

From the above figure 4.2 it can be conclude that we have only images of clouds in our simulation. As the pixel lightness is proportional to the value of pixel, grayscale images have been recommended by Siafu. These images are used to extract the pixel value at any map point. This pixel value represents
contextual information of an agent at a certain point. These cloud images also
represents different threshold values which are translated by Siafu. The number
of overlays depends on the number of images.

2) **Behaviour Programming:** In order to run the simulation three models
needs to be programmed:

- **Agent model:** What the agents do, what’s there activity, more detail
can be found in Chapter 2, Section 2.8.4.

- **World model:** What about the places, more detail can be found in
Chapter 2, Section 2.8.4.

- **Context model:** Over time how the overlays progresses, more detail
can be found in Chapter 2, Section 2.8.4.

These models can be programmed by extending “BaseContextModel”,
“BaseAgentModel” and ”BaseWorldModel” from the Siafu package. Agents
can be created in the agent model class by using the create agents method.
These agents are controlled by a handler function called handleperson.

```java
private void handlePerson(final Agent a) {
    switch ((Activity) a.getActive()) {
        case WAITING:
            break;

        case WALKING:
            a.wander();
            break;

        default:
            throw new RuntimeException("Unable to handle activity "+ (Activity) a.getActive());
    }
}
```
Figure 4.4: Handling Agents

The agent’s normal activity is walking and they move randomly through the walkable areas but they can be stop at any particular time by exchanging the ACTIVITY to WAITING.

3) **Packing the Simulation**: Packing the simulation means to put all the simulation stuff in one place. Siafu follows a certain directory structure that is why the packing process is important. Packing is important because Siafu follows a directory structure in order to run the simulation. There are three directories to be created, one for the classes, one for the configuration [30], and one for the rest of the image files. So the directory structure is like this:

- `/etc/config.xml`: Simulation Configuration file: this describes what the simulation contains.
- `/packagename/*`: all the classes: my classes which are extended from the World, Agent and Context model.
- `/res/map`: wall and background files: PNG images contain the background and walls images.
- `/res/overlays`: overlay files: PNG images contain the overlay images.
- `/res/places`: image files identifying places: PNG images contain the place images.
- `/res/sprites`: image files defining agents: Contains Sprites images of the agents which move around in the simulation area. Cars images or any other image are also contained in this directory.
Now the simulation is ready for the run. The simulation can also be done without using the visualizer, and this can be done by changing the config file which is created upon running the simulator first time.

### 4.3 Customizing the simulator

The purpose of this thesis is to identify a simulation component for the distributed overlay. The main focus of this implementation is to make use of the simulator to test the distributed overlay for real time context data.

#### 4.3.1 Why is this Needed?

This simulation component has been done for the purpose to simulate large scale applications. The context data will be later persisted in the distributed overlay. The reason why this simulation component has been implemented is that we need to test the distributed overlay with respect to context data. So, to do this we need a simulator that represents real world entities into virtual environment.

#### 4.3.2 Implementation

In the implementation process, the Siafu simulator was customized to meet the requirements. To control the parameters of simulator a GUI was created in java that communicates with the simulator. The GUI comprises of simple functions like selecting day of week to be simulated. The number of agents is dependent on the day selected for example, on weekend the number of agents are higher as compared number of agents in other days of the week. Then there is also an option for customized entry of the agents that user can create any number of
agents. The context data is then stored after getting the storing time as an input in seconds.

This GUI allows the user to run the simulator in background. This is done by calling the `controller()` method in the main GUI class. This `controller()` class is responsible for running the simulator. So the original source code has been modified for this purpose. The `controller()` method is connected with the `start simulation` button, by clicking this button the `controller()` method is called and the simulator is started.

### 4.3.3 Adding New Layer

It is easier for the user to add new layer to the simulator. This can be done by uploading an image file to the res/overlays folder in the Siafu. There is also an interface that handles this feature. User can upload the selected image of the layer into the specified directory by using a simple java interface. After the image file has been successfully uploaded the developer needs to specify this
new layer into the config file in the etc folder of the simulation. This config file is the basis for starting the simulation. Everything the simulator needs is in this config file.

4.3.4 Putting Layers into the Overlay

The contextual information of the agents was then fetched from the simulator and dumped into the overlay. This process was done through java code available in the Siafu jar file. By calling the `getcontext()` function from the java code we can get the value of the specified context in respect to an agent. This function returns an integer value which is then stored in the distributed overlay by persisting the agent ID first and then calling the corresponding context object.

4.3.5 Output Formats

The interface also handles the output of the simulation. According to the requirements the simulation data has to be dumped into the overlay. But another feature has been added into this interface, in which user has the option to see the output in a text file. So both the options are applicable from the GUI, like if the text file output is not used then context data will be stored in overlay else it will out to a CSV file (comma separated value).

```
<table>
<thead>
<tr>
<th>G</th>
<th>H</th>
<th>I</th>
<th>J</th>
</tr>
</thead>
<tbody>
<tr>
<td>AtmosphericPressure</td>
<td>Humidity</td>
<td>Lightintensity</td>
<td>Temperature</td>
</tr>
<tr>
<td>Text:236548</td>
<td>Text:3684408</td>
<td>Text:144</td>
<td>Text:127</td>
</tr>
</tbody>
</table>
```

Figure 4.6: CSV Output
The user also has the facility to select the context attributes that they want to store. This can be done by clicking the checkboxes in the main GUI. Upon clicking the start simulation button, the simulation is started and the context data is shown in a separate frame. This interface shows the features that are customized and controlled in a specific way. The GUI shows only those features which are customized with respect to the requirements of the project. The simulation can be selected by clicking the drop-down box in the first field set of the interface frame. The numbers of agents are dependent on the selected simulation day. The context data can be stored in the distributed overlay in a separate time frame which can be controlled by the interface. It allows us to store data with different time options in seconds.

![Simulation Output](image)

*Figure 4.6: Simulation output*

The above figure 4.6 shows the simulation output in a separate window frame. This contextual information is extracted by the Siafu from the cloud images that we put in the `/res/overlays` directory. Contextual information concerning an
agent of any type can be fetched at run time. Now this context data is then stored in the distributed overlay.

4.3.6 Defining Agents and persistence in the Overlay

The available source code was modified for the implementation to get the desired output. The contextual information can be acquired by the movement of the agents. Therefore these agents need to be defined in a specific way. The GUI handles the number of agents by taking a specified value from the user.

```java
public ArrayList<Agent> createAgents() {
    System.out.println("Creating " + POPULATION + " people..";
    ArrayList<Agent> people =
        AgentGenerator.createRandomPopulation(POPULATION, world);

    for (Agent a : people) {
        a.set(Activity.WALKING);
        a.setSpeed(1 + RAND.nextInt(TOP_SPEED));
        a.setVisible(true);
    }
}
```

Figure 4.7: Creating Agents

Persisting agent’s data into the overlay can be a tricky process. Before the contextual information has been dumped into the overlay, the agents were assigned a random ID so as to differentiate each agent. Persistence is done by modifying a sample application of the overlay.

4.3.7 Modifications to Siafu’s Source Code

As Siafu is an open source context simulator so most of the source code is available [30], but some modifications were made to that source code to achieve the thesis objectives. The number of agents is allocated dynamically to the agent model class. This is done by taking the total number of agents from the interface and assigning it to the agent model class. More modification like a csv output
code has been embedded in to the simulator. To achieve the primary objective a major change has been done with the source code. The contextual information of agents is taken from the actual source code and dumped into the overlay.

4.4 Distributed Overlay Implementation

The distributed overlay was created by Sergio Quintanilla Vidal which was further enhanced by Markus Swenson [34]. The implementation uses distributed architecture to manage and store the available resources within the network [36]. It is an XML-based P2P protocol at application level. This thesis work is implemented in respect to the distributed overlay. The code of context simulator together with the GUI has been integrated into the code of distributed overlay. The code was embedded in the package named se.miu.mediasense.

4.4.1 How it Works?

The context simulator has been put on top of the distributed overlay. Upon running the simulator through the interface the contextual data is generated very frequently which is then store in the distributed overlay by making use of the sample application provided by Enriko. Agents have been assign a random ID, which is then register with the overlay by calling the register() method of the overlay. Once the agent is registered with the overlay then it becomes easy to store the contextual information of that agent.

4.4.2 Which Parts of Distributed Overlay Are Used?

In order to store data in the distributed overlay, some parts of the overlay are used. Siafu provides contextual information regarding the agents. The objective was to store this data in the overlay. To achieve this goal some of the parts of
overlay were used. In the implementation of sample application a `register()` method was used. This part of the overlay allows the developer to register an agent (a UCI) with the overlay. Then context attributes of type context are created for each agent.

### 4.4.3 Modifications to the Source Code

Context attributes are created by creating a context object from the package `se.miun.objectLayer.sampleApplication.contextObject`. Before the agent has been register with the overlay, an instance of overlay has been created. This is done by creating an object of object layer. This object layer has the responsibility to persist data in the overlay. Before the creating the context object, context attributes needs to be defined first. This can be done by creating instance of the specified context attribute. These context attributes were created in the same package where the context class resides. After creating the context objects the context data was taken from the simulator and dump into the overlay by calling the `register()` method. So the final picture becomes like this:
The above Figure 4.7 shows the overall flow of the object. There are three main components that are involved in the implementation. First, the context simulator (Siafu) that generates the contextual information. Second, the distributed overlay which persists this context data in the overlay. Lastly, the GUI that handles the simulation. The agent’s data is updated very frequently because the simulator produces very frequent data so to handle this situation a thread delay was introduced into the overlay code.
5 Results

This thesis sought to identify a simulator component for the distributed overlay. In this thesis a simulator component has been extended and then connects to the overlay to test the distributed overlay. To improve and increase the performance of the simulator it was run without the visualizer so as to decrease the processing burden. This work was carried out in numerous steps; firstly, it identifies and examines a simulator component. Secondly, it has its focus on customizing the simulator according to the requirements of the project. The third and final step gets the data from the simulator and stored it in the distributed overlay.

5.1 Context Simulator

The first step determined the simulator component. This was done after searching a lot of simulators and selecting Siafu as a simulator component for simulating context information.

5.1.1 Evaluation of the Ability to Simulate

Siafu is a large scale simulator for ubiquitous computing. It can simulate almost all kind of context-aware applications. It has the ability to generate contextual information very rapidly. To test the ability of the simulator a test case was defined. In this test case Siafu was tested for different general context attributes like temperature, humidity, atmospheric pressure and light intensity. It has been found after running the simulations that Siafu can simulate context of almost any type. More context attributes that can be simulated with Siafu are position, entertainment, hotspots, crime rate and many more.
5.1.2 Test Setup

A test setup was created in order to measure the simulating capacity of the extended simulator. The main focus this setup was to evaluate the ability of the simulator in terms of capacity, memory usage, increasing and decreasing layers. My expectations from this test are that am I able to simulate agents as compared to the original one. Secondly, is there any performance degradation upon customizing it. The computer system used in the setup was IBM Lenovo CORE i5 having 4 GB memory and running Windows7. The network interfaces delays and other delays were ignored in this thesis as these were not the primary focus of the work. The only delays considered are the end node delay that persists the data and the system delay that generates the context.

5.1.3 How Much Can I Simulate?

Siafu can handle large scale simulations as well. Upon testing the simulator for scalability, it was found that Siafu can manage a simulation of size as larger as a city. The simulator was tested for large scale simulations as well. In order to find the maximum limit of the context simulator it surprisingly found that Siafu can handle large scale simulations up to 20000 agents simultaneously, which is the same as original. Though the simulation process gets slower as we increase the attributes and number of agents but still it can handle large scale simulations.

<table>
<thead>
<tr>
<th>Siafu</th>
<th>Simulation Capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Simulator</td>
<td>20000 Approx.</td>
</tr>
<tr>
<td>After Modifications</td>
<td>Up to 20000 Approx.</td>
</tr>
</tbody>
</table>

Table 5.1: Handling Capacity of the Extended Simulator
5.1.4 **Increasing Number of Nodes**

Increasing the number of nodes can affect the performance of the simulator. To test this feature a use case was implemented in which the simulation were distributed on different computer systems, while one system was listening to all the participating nodes. The nodes connect with the receiver through a TCP port. As the number of nodes were increased the simulator becomes slower and slower. Up to 5 nodes the simulator was working fine but exceeding 5 nodes it gets slower in terms of both output and performance.

5.1.5 **Memory Usage**

The simulator did not use too much memory; even a slow system like Pentium III can run this simulator fluently. The worst thing occurs in two cases: one is when there are too many context attributes with too much number of agents and second when the simulation is run for a long time.

![CPU Usage History](image)

Figure 5.2: Simulator Memory Usage

The simulator can be run successfully for 12 hours, but after 12 hours the performance of simulator gets decreases. The reason behind this problem can be the large amount memory that the simulator consumed on running for a long period of time. Figure 5.3 shows the graph of memory consumption by the simulator for different frames.
5.1.6 Adding New Sensor and Layers

Adding a new sensor or context attribute to Siafu can be done in two steps. All we need to do is to store an image of the context attribute in the res directory and maps it to the directory structure of Siafu. Next thing we need to do is to associate this image with the overlay. This can be done in config file present in the etc directory of the current running simulation.

5.1.7 How Does it Works with Increased Layers

The simulator uses layers to calculate the sensor values. During tests it was found that it can handle up to 15 sensor layer. With maximum up to 15 sensors the simulator shows great performance both in terms of load balancing and memory usage. But when it is connected to the overlay it becomes slower and the performance decreases by increasing the number of sensors.
5.1.8 Anomalies

The extended simulator works really fine in comparison to the original simulator. The results are far better than expectations. But in some cases the results are not fairly good like for example, the simulator allows us to simulate up to 15 layers. When I tried to add the 16th layer the simulator failed to run. Moreover, I was expecting to run the simulation for a long time, but the simulator allows me to run for only 12 hours. These limitations are may be due to the constraints imposed by the Siafu developers or it can be due to the heavy memory usage of CPU by the simulator.

5.2 Simulator Behaviour in Respect to Overlay

It has been observed through testing that the contextual information from the simulator can be stored in the distributed overlay. But it is significantly noticed that the simulator creates context data too quickly for the overlay. This problem can be overcome by decreasing the speed of the agents or decreasing the simulation speed or introducing a delay in the simulation thread. There can be so many things to be expect when the simulator is attached to the overlay like, the behaviour of overlay against simulator, how fastly it stores the simulator's data, etc.

5.2.1 Adding New Context Attribute to Overlay

Adding new context attribute to the overlay is tricky process than adding new sensor to the simulator. We have to modify some code of the overlay to add a new context attribute. New context attribute can be created by creating a class in the package name se.miun.objectLayer.sampleApplication.contextObject, and
then also we have to do some manual coding in order to implement this new attribute fully.

5.2.2 The Response Times
Like the simulator response time for the overlay has also been measured.

![Graph showing response times for overlay registration](image)

Figure 5.4: Response Times for Register (Overlay delay)

The above Figure 5.4 shows the time taken by the overlay to register a agent and then persist the data. It shows that for one agent the overlay took 167 ms to register an agent in the overlay. The resultant graph shows that I can simulate it and put information into the overlay and at the same time I don’t see any loss of performance between the simulation and the overlay. Figure 5.3 shows that there is also an increase in the persistence time when we increase the number of agents. Besides these overlay delays a system delay of 3 ms was also noticed at the time of running the simulations.
5.2.3 Application Crash

The distributed overlay works very well with simulator. But after sometime the overlay crashes and there involves some uncertainty in the application. For the slow incoming data, the overlay behaves well, but if the context data is delivered to the overlay too quickly it is not able to handle it. It may be because the overlay uses such kind of mechanism that makes it heavy for handling real time data. Once the overlay is started it reacts very well against the simulator, but as the simulations goes on and frequent data is produced the overlay gets tired after 2 hours when simulating 1000 agents simultaneously.

5.3 Context Attributes

After selection of Siafu simulator in the first step, different context attributes were simulated by using the Siafu simulator. By making use of the context
attributes we can gather enough contextual information about presentity. The context attributes that were simulated are Temperature, Light intensity, Humidity and Atmospheric pressure. The information regarding these attributes can be gather from the movement’s of the agents, i.e. when the agents in simulator changes their position the contextual information is upgraded. Each agent in the simulator has unique contextual information. Normally, in these attributes the temperature variable changes very frequently. Apart from the user defined attributes Siafu has its own context dimensions as well, for example position of the agent.

![Image](image-url)

Figure 5.6: Clicking on agent shows the Agent’s context values

### 5.4 Customization

Customization of a readymade system is relatively a hard task. There were several modifications that were needed in order to build a suitable simulator component. Firstly, the visualizer of the Siafu was made invisible to the user, i.e. the simulation runs in the background. By doing this the processing burden of the CPU becomes less as compared to the method in which the visualizer was visible. Also the retrieving of contextual information becomes very fast without
using a visualizer. Secondly, an interface was created that controls different parameters of the simulator like number of agents, simulation day, data storage times. Lastly, the context simulator implemented with respect to the distributed overlay so as to persist the data easily into the overlay.

### 5.4.1 Performance Comparison

The performance of original simulator is not even affected by modifications. In fact the extended simulator component is working fine as same as the original one.

<table>
<thead>
<tr>
<th>Siafu</th>
<th>Simulation Capacity</th>
<th>Simulation time</th>
<th>Startup time</th>
<th>No of Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original Simulator</td>
<td>20000 Approx.</td>
<td>12 hours</td>
<td>4sec</td>
<td>4</td>
</tr>
<tr>
<td>Modified simulator</td>
<td>Up to 20000 Approx.</td>
<td>12 hours</td>
<td>2sec</td>
<td>4</td>
</tr>
<tr>
<td>Modified simulator</td>
<td>Up to 15000 Approx.</td>
<td>12 hours</td>
<td>2.5sec</td>
<td>15</td>
</tr>
</tbody>
</table>

Figure 5.7: Performance Comparison

From the above Figure 5.7 we can conclude that the results are better than expected. Modifying an open source simulator is relatively a hard task so achieving these kinds of results shows that the overlay is quite capable of handling real time context data.

### 5.4.2 Performance over Time degradation

It was noticed that the performance of the simulator was excellent till 12 hours of running the simulation continuously. But, after 12 hours the performance of
the simulator was affected. The main reason this degradation is may the mem-
ory usage of the simulator or the simulation time. So, the performance of the
simulator in terms of percentage is as follows:

![Figure 5.8: PerformanceDegradation in Percentage](image)

5.4.3 Accuracy

The accuracy of the context data was tested by creating a simple test. In this test
the accuracy of contextual information was tested between the overlay and the
simulator. The tests result shows that the data in overlay is matching with the
data produced by the simulator.
5.4.4 Updation in Overlay

Updation in overlay is set according to the settings of the simulator. If the speed of the simulator changes the overlay thread is set according to that speed. This is done by introducing delay in simulation thread. So by this mechanism the data in overlay is up to date with the data in simulator. But it is done up to a certain speed of simulator.

![Figure 5.9: Overlay Response to Increasing Simulator Speed](image)

The above Figure 5.9 shows that for 10 agents with different simulator speed the overlay has the same response. This means that increasing the simulator’s speed does not really affect the overlay performance.

5.5 Evaluation

Different evaluation tests were conducted to evaluate the results of simulator. The tests were performed under the same conditions.
A Distributed Context Simulation Component

5 Results

- Same computer system.

- Same number of agents but with different scenarios.

- Varying the storage time.

The evaluation results shows that the extension to the simulator works well both in terms of customization and overlay response. The results were found more than satisfactory for the implementation.

5.5.1 Reliability

The performance comparison table in Figure 5.7 shows the reliability of simulator. The large number of agents shows that no matter we have 5 agents or 500 agents the simulations is reliable up to certain aspects for example, startup time, increased layers, memory usage etc. Although the time to register an agent is more when we increase the number of agents but still we can manage to control large scale simulations. For the evaluation of nodes for distributing the simulation over different machines, the time taken is so long that ideally it will not be ideal to run the simulations on large number of nodes.
6 Conclusion

The aim of this thesis was to identify a simulation component for the overlay. The overall problem statements are:

- To find a context simulator component and then customize the simulator according to requirements.
- To store the generated context data in the Distributed overlay.

The answer was hunted by exploring different context simulators. The choices available for the solution were DiaSim, SimuContext, A Generic Location Event Listener and Siafu. The project described a simulator called Siafu as a solution to the problem. The solution covers the searching for a context simulator and then the customization process depending on the desired goals of the project.

The Theory in this thesis focuses on the context-aware applications and ubiquitous computing applications. Though it is hard to connect the physical world with virtual one, but it is possible to simulate the real world in a virtual way on computers. Different context simulators were reviewed in Chapter 2 but after studying them all, Siafu was selected for implementation.

Siafu was selected as a context simulator because it best suits the project requirements. Moreover, the type of simulation Siafu does is exactly what we were in need of. The Methodology Chapter discusses the units of the simulator and addressing different issues regarding these units. As an open source simulator, Siafu has many rich facilities open for the developers.

Some extra analysis could be achieved in order to find out whether it is likely to run the simulator as client server architecture. For example, making one ma-
chine as server and running the simulator on several machines, while the server will be listening on the data on a TCP socket from all the machines running the simulator.

The Implementation Chapter shows the practical implementation of the thesis work by specifying the creation of simulation, defining the places, maps and overlay with respect to the distributed overlay. Moreover, it also presents a GUI for controlling the simulator in respect to the distributed overlay.

### 6.1 Problem statements

The problem statements asked: find a context simulator component? In addition there was also the question regarding the customization of the simulator according to the project requirements.

There is a constructive reply to the first question as this thesis identifies a context simulator component for the distributed overlay. Siafu is relatively a large scale context simulator that can support context-aware applications by presenting the real world entities in a virtual manner. The basic goal of thesis was achieved by extending a context simulator called Siafu.

This thesis presents the work on customizing the Siafu simulator in order create a simulator component as describe in the introduction chapter. The performance of the distributed overlay gets affected with the increase of number of agents or frequent data creation at the simulation end.

### 6.2 The results

The entire tasks are conducted in framework and in matching manner. The starting up simulator takes a little time to map the overlays with agents and
maps, but once the simulator started contextual information happened to appear very rapidly. The results I obtain show that the overlay responds very well to the context data from the simulator.

Further investigations are requisite to identify whether it is easy to handle this frequent data in a distributed overlay or not. The produced graphs are taken on the same machine running the simulator on different times.

The response time’s table in the Results chapter shows that the distributed overlay has fairly a good mechanism to handle the simulation data. The Performance comparisons table in section 5.4.1 shows the overall performance of the extended simulation component, which shows that the results are quite satisfactory. So I meet all the project requirements for this thesis upon extending a simulation component for the distributed overlay.

6.3 Future work

For further future work many other possibilities could be investigate. Having considered all these things we can move forward with the future work. All the thesis work focuses on the simulator component, other issues like handling of the real time data could be discussed. As the distributed overlay takes a little time to register an agent for the first time so this delay could be minimized. Future work could identify the cause of these problems. It might be probable to restructure the proposed architecture of the simulator.

Future work could also focus the issue of scalability of the framework, by distributing the simulator over several computer systems.
References


http://trid.trb.org/view.aspx?id=690128


References


[34] ” DCXP and SIP Interworking - Improving context dissemination”, Victor Kardeby (Master thesis).


Appendix A: Project Flow

Project Flow of the implementation
Appendix B: UML diagram of the proposed structure