Universidade do Minho
Escola de Engenharia

Luís António Araújo Ferreira

Simulation of large scale Pervasive Displays Networks

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação de
Professora Doutora Helena Cristina Coutinho Duarte Rodrigues
Professora Doutora Maria João Mesquita Rodrigues da Cunha Nicolau Pinto

Outubro de 2012

Universidade do Minho
Escola de Engenharia

Luís António Araújo Ferreira

Simulation of large scale Pervasive Displays Networks

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efectuado sob a orientação de
Professora Doutora Helena Cristina Coutinho Duarte Rodrigues
Professora Doutora Maria João Mesquita Rodrigues da Cunha Nicolau Pinto

Outubro de 2012

We build too many walls and not enough bridges.

ISAAC NEWTON

Acknowledgments

It is now time to express my most profound gratitude to everyone who helped me arrive on this important step of my life.

I want to start by thank my dear supervisors, professors Maria João Nicolau and Helena Rodrigues, for their guidance and wise words along the course of this work. Without them it would not be possible to reach this stage. Other professors had also an important role on my academic life, their advise and support from the simpler questions to the harder ones were priceless. Among others, a special thank to professors Rui José, António Costa, Bruno Dias and António Nestor Ribeiro.

I also want to address some special words to all the people that constitute the Ubicomp Group of the Algoritmi Research Center at the University of Minho, for the brainstorms and the pleasant leisure times we have had during this year.

The last but not the least, to my Mother and Father that have always been by my side every second. They endured my lacks of humor, supported me even without saying anything gave me the strength needed to overcome the rocks on the road. For them my most sincere: ”Thank You!” A special thank has to be addressed to my bother, who never had problems to say the most relentless words, opening my eyes and unveiling me new possibilities, even when I thought I was correct. His strength and resilience will be forever a model that I will try to pursue throughout my existence.

To my friends, not referring to no one in special, for the patience, companionship and love given without any hesitation.

As Sir Colin R. Davis once said:
“The road to success and the road to failure are almost exactly the same”.
What we have not said in his statement is that the right side of the road most of the times is pointed, even not noticing, by those who take care of us and tirelessly bear us. The truly friendship and companionship of our beloved ones is undoubtedly the best compass that we can own throughout our lives.

Resumo

As redes de ecrãs públicos de área alargada estão-se a tornar um paradigma emergente e representam uma transformação radical em relação à maneira como encaramos a disseminação da informação em locais públicos.

Estas redes com o sua natureza ubíqua levantam alguns desafios para quem tem que as desenhar, instalar e usar. É bastante importante perceber quais são as principais compromissos a assumir quanto ao desenho das redes de ecrãs, principalmente em relação aos seus componentes e respectivos protocolos, para desta forma podermos oferecer uma rede aberta, global e sobretudo escalável.

A partir destas ideias o trabalho de caracterizar os componentes de rede é um dos pontos essenciais para alcançar um desenvolvimento fundamentado deste sistema. Também é fundamental ter uma avaliação dos desenvolvimentos respeitantes à desempenho do sistema e à forma como o aumento do numero de intervenientes no mesmo afecta o seu comportamento.

Assim este trabalho, complementando essa caracterização e classificação inicial, pretende desenvolver uma ferramenta que permita às demais equipas multidisciplinares criar cenários e modelos de simulação para confirmar se as suas decisões quanto aos padrões a implementar são os que melhor se adequam aos requisitos destas redes.

Abstract

Large-scale pervasive public displays networks are becoming an emerging paradigm and represent a radical transformation in the way we think about information dissemination in public spaces. These networks with its pervasive nature rise a number of challenges for those who have to design, test, deploy and use this kind of networks. It is imperative to understand what are the key tradeoffs in the design of pervasive displays networks, mainly on their components and respective protocols, in order to provide a fully open, global and most importantly scalable displays network.

Starting from these ideas the work of characterize the network components is a key step to accomplish a well grounded development of the system. Also the assessment of those developments regarding the performance of the system and how the increasingly number of elements changes its behavior is imperative.

Thus this work, in addition of that initial characterization and classification, tries to develop a tool to enable multidisciplinary teams create scenarios and simulation models to confirm if their design patterns are the ones that better suite the requirements of a pervasive displays network.

Contents

List of Figures

Acronyms

API	Application Programming Interface
IDE	Integrated Development Environment
IP	Internet Protocol
LAN	Local Area Network
MAN	Metropolitan Area Network
NED	Network Description
QoS	Quality of Service
WAN	Wide Area Network

Chapter 1

Introduction

Since the beginning of the computational era, resources are shifting from the centralized single processing unit to the most fully distributed architectures. Most importantly is the changing environment where the computation takes place whether at an office desk or even in our clothes. In fact, nowadays, it is almost impossible to quantify the number of computers that influence directly or indirectly our daily lives. These changes were partly encouraged by the Weiser's vision of the future of computing, as he said:

"Our computers should be like our childhood - an invisible foundation that is quickly forgotten but always with us, and effortlessly used throughout our lives."

This means that computers should be seen as a tool that does not consume our attention. As Weiser said about eyeglasses, with them *"you look at the world, not the eyeglasses"*. He and his team embodied the term Ubiquitous Computing anticipating the disruption in the use of computers, that should be spread everywhere without the need of attention [31].
Almost at the same time the term Pervasive Computing was introduced by IBM and was used to describe the research area that has focused its attention more on technologies, trying to create a seamless interaction between different devices without the need of complex configurations [22]. Firstly the

terms were used separately, one giving attention to the Human Computer Interaction problems while the other more on core technologies, but currently both terms have been merged and are used interchangeably.

1.1 Research Problem

Deriving from the previous ideas and by the increasing number of public displays, like the ones presented at the airports or at outdoor advertisement spaces, many projects have emerged. For example, the European project Pd-Net[1] appears to explore the scientific challenges and new technologies required to enable the emergence of large-scale networks of pervasive displays.

Nowadays, public displays are seen as mere passive elements that always show the same information, not taking care of those who are passing by, they suffer from a lack of content dynamics leading it to an increasingly "display blindness" in terms of perceived and remembered content [19]. So these projects idea is to explore those public displays to offer people new forms of interaction, providing the foundations for a new communication medium which offers entirely new opportunities for business and creativity, offering the opportunity to display owners to have a platform that have different kind of applications to better serve the interests of those who benefits from those displays.

The architectural elements that compose the network have to be inherently distributed and divided into multiple functional components. However, some of those components may be, in some execution patterns, more requested or have a more complex and time consuming operation, increasing systems response time. Also those components have to scale to embrace the growth of the network and to be more fault tolerant.

Taking into account the above ideas, we will concentrate our efforts on developing simulations to study and evaluate the scalability properties of different application execution patterns.

We intend to study how those application patterns will behave on this

[1]http://pd-net.org/

cooperative ubiquitous environment, mainly in the presence of an augmenting number of users and execution nodes. This corresponds to analyze the suitability of the above patterns considering the pervasive displays networks assumptions and characteristics.

Will the same web standards also apply, knowing that content and resources could be in multiple and potentially far locations? Do they scale up as the well study standard web scenarios? What will be the average response times when the number of application interactions rises or even when the sensed information floods the system? What will be the impacts on the network with the execution of multiple simultaneous applications with different rich and demanding content types or content hosts? How will different application interaction patterns or synchronization demands will affect network performance and responsiveness? Are some of the questions that almost immediately appear concerning these networks.

1.2 Motivation and Objectives

The shifting paradigm of digital public displays has undoubtedly a full spectrum of unexplored concepts. The novelty and emergence of these systems represent a challenging research area that requests ideas from multiple disciplines of the mobile and ubiquitous computing fields.

Public spaces or semi-public spaces, represent a challenging environment to deploy and test these new systems, and so we have to make the effort to bring new content to displays, turning them into active elements offering opportunities for people to use them, providing a new communication medium, instead of being passive forgotten actors, like most of the Digital Signage displays.

Information displays, first in paper and years later, with increasingly price reduction, as digital screens, were mainly meant to provide information to those who were passing by. Digital displays represented a big step in what could be presented and those who wanted to announce something could now show much more eye catching contents. Unfortunately, content was quite static and could not be remotely changed. To do that a maintenance team had to reset, locally, each new content. With continuously network ubiquity and the possibility of interconnecting all electronic devices a new reality rises. Connecting those displays to some remote server turns on the possibility to remotely produce and schedule contents in a much more dynamic way.

Currently, research on Pervasive Displays is mainly focusing on Human Computer Interaction. Teams are aiming their studies in the best way to capture people attention and in the possible interactions between people and displays. With the advances in the research it is also time to start to take care about the design of the network components. The potentially large-scale characteristics of these kind of networks rise inevitably several problems concerning scalability and performance. These systems, including Pd-Net, and their inherent innovative nature are characterized by a continuously changing number of users, display owners, content producers, display nodes, application items, application hosts, content types, interaction modalities, sensors and connections. In this way, it is needed to make a full study specifying what should do the core components and how would they will be interconnected in a pervasive network.

The outcome of this work is the study of the main tradeoffs in the design of Pervasive Displays Networks in respect to its main core components, protocols and respective interactions. To accomplish these results, simulation should be used as a tool for assessing the operation of the system under various conditions, addressing the effects of the scale in the system performance. With that it is also intended to understand the main protocol properties. To do that is fundamental to model the core architecture and the core software components and build, based on simulation techniques, a testbed for testing and determining the scalability requirements for different representative scenarios of use of the network. Components are not yet defined in a restricted way. Currently, there only exists a division in functional blocks that could be rearranged in multiple physical components. The assessment of the performance of those multiple combinations will be a major key aspect in this work, trying to point out what should be the optimal distribution or in some form have a set of metrics that give the maximum threshold that does not compromise the stakeholders satisfaction.

Chapter 2

Related Work

Lots of research are daily made worldwide aiming the development of displays systems, turning them more interactive, user-friendly and less obstructive on our lives. The outcome of those works are almost every time purely academic and does not overflows the walls of the universities. What we want is to join the pieces, take the best of the research and create a functional approach to this new communication medium.

In this section, it will be presented some pervasive displays systems and some of the new trends on the Digital Signage both with commercial and non-commercial uses. The main objective is to clarify the main differences in those systems, and current implementations, uncovering new opportunities, requirements and challenges for Pervasive Displays Networks.

2.1 Public Displays Systems

As everything, public displays have evolved from the simplest to the most complex interconnected display networks. Presently, we can divide public digital displays in two major groups, those who simple show some kind of information, most commonly called as Digital Signage Displays, with or without some interaction, and those who present some on demand rich content, like the ones suggested in Pd-Net project.

2.1.1 InstantPlaces

This project is held at University of Minho, Portugal, and tries to create a social network for place-based screen media as an instance of the concepts of Pervasive Display Networks. It takes the concept of ”place” to bring together the various stakeholders involved in the operation of public displays. A place arises as an abstraction to reflect one or more physical settings creating a meaningful context for situated social interaction. And so, a place creates a scope of execution, aggregating resources, people and interactions into a single coordination context. People are then invited to influence public displays interacting and expressing themselves through a series of sensors enabling that communication. One of the most well studied forms of interaction in this project uses Bluetooth device names to provide input that will be consumed and shown at the place display [14].

A mobile application is also provided to the users where they can manage their identity, creating personas to be used and exposed in each place, visit a place and interact with the place applications. Those applications can be created by third-party creators and then published for display owners to subscribe who can then manage and create a place history with them. The *InstantPlaces* architecture is divided into four main areas, as it can be seen in Figure 2.1. Each physical place can have a set of displays and sensors that together will provide multiple forms of interaction with the virtual place. That virtual place has associated a set of applications that can be subscribed next to the third-party services and applications component. The called InstantPlaces infrastructure interconnects all components and provides the Application Programming Interface (API) that can be used by applications and users. Users can also access the InstantPlace infrastructure via browser or mobile application. More information can be found at the project website[1].

[1]http://instantplaces.org

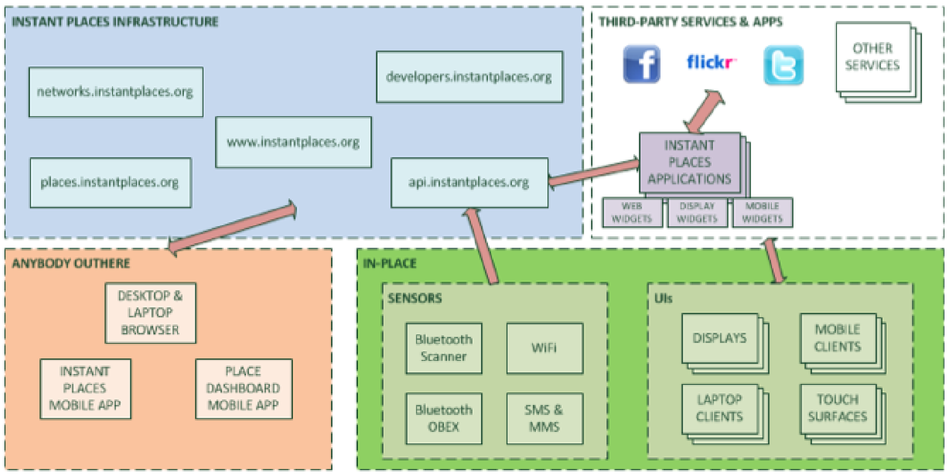


Figure 2.1: InstantPlaces architecture.

2.1.2 e-Campus

Is a network of public displays deployed at the campus of Lancaster University in the United Kingdom, and it was designed to promote an infrastructure testbed for researchers and artists as also to provide a tool to improve campus experience for those who use it. Many displays were deployed as probes around the campus and with more visibility in the sixth Workshop on Mobile Computing Systems and Applications conference, in the Brewery Arts Center exhibition and in an underground campus bus station. While the first two deployments were placed only for a few days, the last is intended to be maintained for, maybe, several years. In all of those deployments, they tried to study and develop an API to satisfy their needs about the scheduling and synchronization problems on displays, for the last one the software was rewritten in order to reflect the lessons of the first two [25].

The architecture used by them was mainly designed to embrace the problems of scheduling and synchronization between applications. After their learning process in the first two deployments, they refined their architecture and defined four main entities: displays, applications, schedulers and handlers that are coupled together as shown in Figure 2.2 [24].

In this way applications render multiple forms of content to be presented

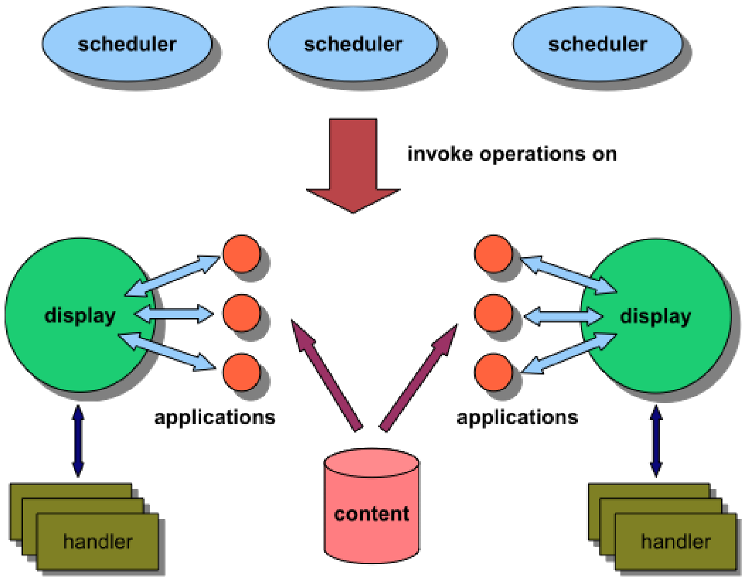


Figure 2.2: Conceptual e-Campus architecture.

on displays when invoked by the schedulers. Handlers are used to detect conflicts of resources and resolve them. Imagine a display formed by a set of screens, all of them have to be synchronized in order to show the content in a coherent way. All the communication is constructed based on an asynchronous publish-subscribe event channel.

From the deployments, they produced a set of 13 lessons, arranged into five categories: technology and deployment, monitoring and management, content creation and management, orchestrating ubiquitous computing experiences and working in public spaces.

It is also interesting to refer that from this project, it has emerged a new display type called *FireFly* [3] that consists in strings of controlled LED-based lighting elements, together they can form three dimensional displays using each LED as pixels. This is useful because it makes clear that these systems can have multiple forms of outputting their information leaving us thinking outside the box and giving an extra challenge to develop an architecture that gives support to all types of forms to outputting information and contents.

2.1.3 UBI-Hotspot

This project is probably the most interesting of all because it has actually a real large-scale long term city deployment in Oulu, Finland, since 2009, and it was motivated by the lack of these aspects in the current research systems. With their deployment, they left the campus environment and went to the real world where many details much more difficult to control, for example, monitoring services, have to be really working to provide a quick alert system when something goes wrong. In this way, they have gathered several lessons to take in consideration in the creation of these systems[11].

This project has a total of 13 large touch screen displays, six of them in indoors public facilities and seven outdoors. They have achieved many results studying the social and economic impacts of these systems in a real urban environment [21].

They also studied the impacts of providing social-networking services in their applications [13] and made several field studies that let them take some conclusions about the challenges of integration of this kind of services and present a mechanism to seamlessly integrate Facebook's account within public displays.

Its architecture, in a very high level approach, relies on a software architecture that defines its components (seen in Figure 2.3) which interacts via an existing implementation of an event-based communication overlay (FUEGO architecture [27]). This overlay takes care of the publish-subscribe communication and other aspects, like the fault tolerance in callbacks [20].

On the UBI-Hotspot, resources are managed by the Resource Manager that receives input events from the context wrappers, and informs the layout manager about the changes, so it can take care about the screen partitioning to each application. People can interact with the hotspot with their mobile devices. To bootstrap the system, users have to read a RFID tag so the UBI-mobile launchs the services associated to the hotspot. Applications can be subscribed next to the application server and be associated to each hotspot.

Actually, *UBI-Hotspot*, is the most complete study of these new pervasive networks of displays. They have made studies in multiple areas from

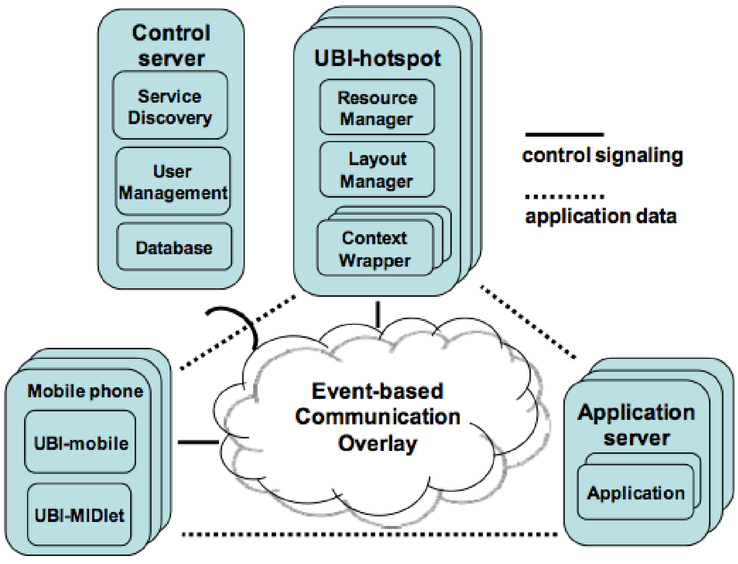


Figure 2.3: Conceptual UBI-Hotspot software architecture.

the middleware, the network components and infrastructures, to the display applications even from the interaction in several ways. However, they have much less research on the consequences of the growth of the number of those various components and about synchronization of applications and spread of contents leaving space to several research on these areas. More information about the project can be found at theirs Web Site[2].

2.1.4 InfoShare

InfoShare is a multimedia signage system developed and deployed in Keio University, Japan, and it tries to create a distributed and scalable digital signage system. They want to create a Digital Signage system that can be managed remotely contrasting with the old-style systems, that have to be updated locally next to each sign. This system is the one who had the biggest concerns about scalability issues. *Infoshare* architecture resides mainly in a client-server model to communicate with all of his components. Its archi-

[2]http://www.ubioulu.fi

tecture relies on four core components: the InfoShare DataBase Server, the InfoShare Content Repository, the InfoShare Web Server and the InfoShare Web Service. This architecture and with whom each component communicates can be seen in Figure 2.4. In this way, InfoShare DataBase stores all the information about users, access control privileges, scheduling and the screen layout. The images and videos are stored in the Content Repository. It also

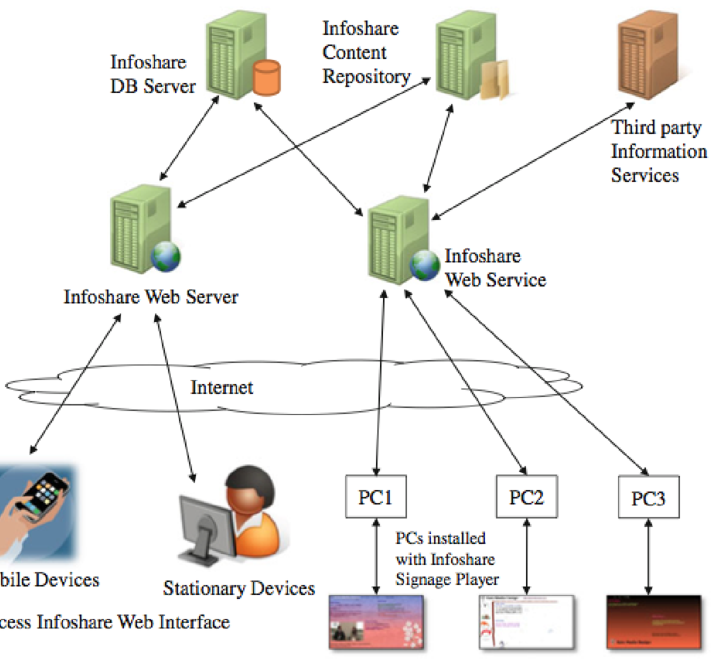


Figure 2.4: InfoShare architecture.

has on screen a Signage Player, who renders the contents. These contents are accessible by the InfoShare Web Service, that communicates with the rest of the system in order to provide the pretended resources. Multimedia resources can also be pulled from external third-party services.
The system can also be managed by is Web Interface, in which users can reschedule contents, manage accounts and monitor the system [26].

Besides this apparently complete architecture, this system does not provide rich contents as the other systems and it has its focus on the distribution of multimedia content, like news, videos or images. It does not have the concept of application to each display. However, their complete studies about the system overall performance [9] can help us a lot in constructing a more efficient and responsive system.

2.1.5 Other systems

Many other systems with commercial and non-commercial uses exist nowadays. There are many on campus studies with similar approaches to the previous systems, like iDisplays [18], ReflectiveSigns [16], or Plasma Poster Network [4] and others with completely different objectives like GAUDI [15] that is a pervasive navigation system or MobiDiC Shopfinder [17], that uses public displays as an advertising medium and as a route indicator to the shops.

Besides the clear interest of these projects, they are very focused in solve a specific scenario problem and they do not give much attention to the underlying network and the scale problems. However, they are very important to settle down some ideas and prevent us to make some mistakes. All of these systems are important to analyze how should the Pd-Net project should embrace the problem of provide a common flexible architecture to sustain the multiple approaches that may derive from the different research communities. With them we can see what components are mainly used and how they interact, helping us to create a more capable and flexible network.

2.2 Other related studies

To better conduct and inspire our work many other academic studies were revisited. Different scientific areas were covered, from the previously seen works about pervasive displays systems to more specific researches like the ones regarding Human Computer Interaction. We have also dig on the world of network simulators that will be presented further on Chapter 5.

On the work of Sarah Clinch *et. al.* [6] we were presented with the problematic of design and deploy application stores for public pervasive displays networks.

Along this work the main focus is, obviously, around application stores for public displays, where they make the task of identifying some of their design considerations and how they really differ from the standard mobile application stores. They have also made a categorization about the expectations of different types of the system stakeholders and a division between applications regarding the benefits for those stakeholders.

From the point of view of the application developers that study made some considerations about the distribution and control of their applications, the business models and at the end they identify two main interfaces (APIs) to submit an request applications next to the application stores.

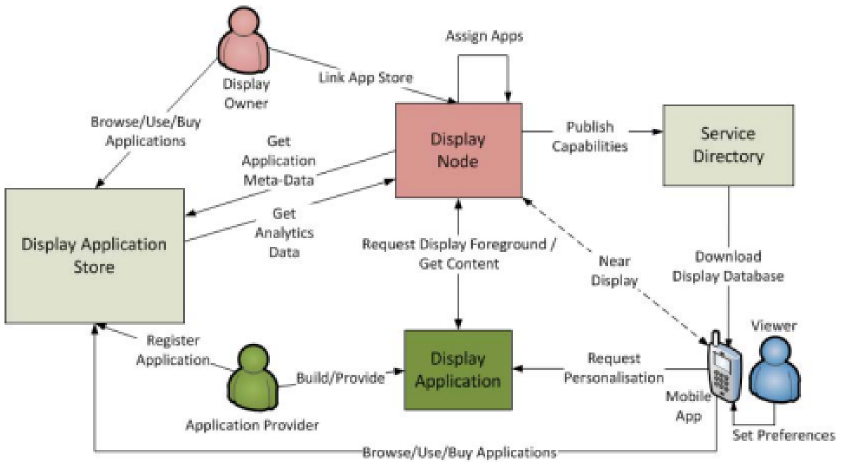


Figure 2.5: High level architecture.

But what really makes a this work so interesting is their vision about the hight level architecture, Figure 2.5, and the processes between their high level components. The architecture itself do not compromise all of the components to create a fully functional pervasive displays network, instead they concerned on those to conduce their study of creating an interface for the application developers.

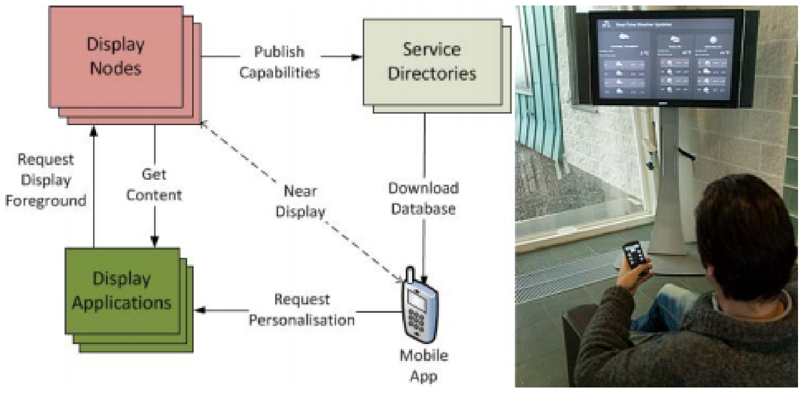


Figure 2.6: Display personalisation.

They have also created their architecture to enable display personalization (Figure 2.6) and defined the interactions for that.

Specifically those interactions, for display personalization, may have a major importance to help us define the best way to model the simulation scenarios.

Another very important contribute to settle down some of our ideas is the paper that rises the question of *"How close is close enough?"* for applications that regard display appropriation by mobile users [5].

This work tries to measure the impacts of the application location on the user experience. They try to answer to several questions like: a) *"Where should an application execute for good user experience?"*; b) *"Can it execute on a distant cloud with high network latency? Or, is it necessary to execute closer to the display and user?"*; c) *"Can we quantify the impact of latency on user experience?"*

To achieve the proposed goals they have prepared a set of cloudlets spread globally to assess if their location has really an impact on user experience. Then they measure the time between a user request, on his mobile phone, and the display update. Also their gathered a group of participants presenting them with a simple game. After the gameplay they have answered a questionnaire to collect data about their perception of the responsiveness, the usability, their sense of control and their frustration before the game on different locations.

The results were very clear, the distance of the application really affect user experience. On the paper they present the complete set of results and measurements that may be posteriorly taken into consideration.

In the end the main conclusion is that the initial question, *"How close is close enough?"*, do not have a simple answer. It depends on multiple factors: a)"the interaction-intensity of the application"; b)"end-to-end network latency (whilst loosely correlated with physical distance, measurements presented here show that the nature of the correlation is complex)"; c)"the host's hardware and software"; d)"user—some are more tolerant of delays than others. Even the same user may respond differently over time (e.g. becoming less tolerant when in a hurry)".

The set of considerations discussed on this paper are quite interesting for us because it presents some analysis regarding the user experience expectations facing interactive displays applications. It also provides some useful latency measurements on different locations that may be applied on our simulation models.

This paper, even not directly, makes a warning showing that if we want to develop a open pervasive displays network we have to give a special attention to the design of underlying system infrastructure because the high variety of users, application types and interactions have to coexist on the same global network.

Chapter 3

Pervasive Displays Networks - A Framing Essay

The problematic around the pervasive display networks is huge, different computer science disciplines have work together to overcome the inherent questions raised. The creation of an open network ready to embrace a vast number of application, sensors and content types needs to have a clear notion about all the things in stake.

In our work we try to settle down some of the concepts and ideas that round up the pervasive networks, specifically, all that concerns the PD-Net project. This chapter hopes to contribute precisely to achieve that level of eloquence, defining some of the key aspects and characteristics of this large project. We also want to present some concepts that we thought to be very important to assess and simulate these systems, like the description of the main functional components and the categorization of some existing processes. In the end we pretend that the reader have a clear idea about what is a pervasive displays network with the notion of some of the inherent problems that may emerge.

3.1 Pd-Net project

The Pd-Net project has in its foundations the goal of achieve some unique characteristics[1] that are not currently found in any digital signage display network. These characteristics, by their nature, quickly raise many challenges, each one of them full of questions that have to be studied to prevent surprises in the future. This ambitious project, if well succeeded, will change the paradigm of the information spreading throughout public spaces and will reach populations in a much more effective form. The set of topics that drive the motivation of this project are:

- Personalized Content - with the usage of multiple sensors, content may be adapted to the public passing by, creating a real and interesting communication channel that shows information according personal preferences;

- Support for Multi-Screen Applications and Content - this project tries to offer a open system that is able to support applications that can coordinate a set of displays across the network, like interactive multi-display games, and spreading contents in a coordinated and even on a synchronous form;

- Context-Aware and Situated/Mobile Content - usually digital signage systems broadcast their contents without care about the location or the interests of the receivers. Pd-Net objective is provide differentiated contents according to the users preferences, sensing the environment surrounding the displays, pleasing them, not with static and immutable contents but with some meaning and benefit;

- Global Reach - Pd-Net aims to interconnect multiple existing signage systems in a large network in order to offer new social economic experiences, expanding the possibilities of distributing applications more easily;

[1]http://pd-net.org/about/

- Public Access (Ingestion and Consumption) - multiple forms of providing and gathering contents, trying to merge distinct forms of information consumption, forming the public displays with more general contents, mobile devices with private and personal contents or other display mediums that could be connected to the network;
- Interactive - wants to offer the possibility to directly interact and influence the displays unlike most of the existing signage systems;
- Rich Media - provide support for various types of media, audio, video and interactive applications, that should contribute to radically change the digital signage panorama promoting its public usage and acceptance.

The goals are daring but, if we were successful achieving them Pd-Net will, certainly, create radical changes in information dissemination on public spaces. Hence, there exists the demand of identify and coordinate, even loosely, the different stakeholders that should enable a rich environment to sustain the network.

3.2 Identify stakeholders and actions

In any system, the identification of who are the main stakeholders [28] and what kind of actions they play [1] is one of the first and more important steps to better align the design of the system architecture, so it can have a best fitted and integrated execution between all of them. Pd-Net recognizes three major stakeholders with a different set of behaviors, requirements and needs.

Viewer Implicitly or explicitly influences the system execution, consuming and producing contents. They could have the need, in some applications, to have a previous account registration to have a full access and usage of it;

Display Owner Is responsible for the displays installation and maintenance. He has the responsibility to subscribe, to schedule and to manage applications. Contents (3rd party or user generated) regulation can also be under his scope;

Application Developer Creates applications to be used on the displays network. They have to submit and update applications to the application stores. They have also to take care about the applications charges to his costumers.

Content providers could also be seen as a system entity, however we assume that the contents provided are under the direct competence of the application developers or even the display owners. We do not have them as system stakeholders but as a sub-entity that makes part of the system but does not have a direct influence on it without the wills of the other stakeholders.

Although this set of stakeholders and actions may appear very simple, the underlying architecture has to give response to a myriad of interactions. The components have to cooperate to provide these functionalities to all the intervenients.

3.3 High level abstract Pd-Net Architecture and components

Other starting point of any work is to identify the main functional components that are needed to accomplish the proposed goals. In this section, the main functional blocks compose the Pd-Net network will be described. These components should not be seen as rigid and immutable architectural elements, they could, in fact, not exist or be coupled together in multiple combinations according the needs. Hence those components are seen as key functional services responsible for enabling an open and pervasive architecture as Pd-Net claims.

Display Node A *Display Node*, sometimes called *Pd-Net Node*, is the element responsible for, at least, generating content to one or more display mediums. To achieve this minimum goal may be imperative to have a representation of its current content subscriptions, schedulers and content renderers. However it could be capable of executing more functions as self-monitoring, sensing or cache content.

The display medium is directly controlled by a display node and it can have any kind of hardware since capable to reproduce some type of content. Often is assumed that this hardware has a visual form to display its content, however it could assume multiple forms, from the more simple to the most complex sound, smell or visual renderer.

Environment Service This service creates a meaningful representation of a physical or abstract situated context based in the concepts of, for example, place, event or activity. With this service we may manage and situate the resources associated to these environments creating scopes of usage and execution, converging people, interactions, resources, sensors and applications under the same coordination context.

It could also be responsible for some sensing and interaction information associated to each environment providing services to those who want to create environment aware applications. This service could also store information about the practices and habits taken in each environment.

Display Service It controls one or a group of displays, concerning content and display behavior, allowing the owners to orchestrate and schedule content and content sources, defining the display actions.

Directory Service This component is responsible for providing the information about the existing nodes, taking care about some location based queries to the system.

Node Registry Handles the remote management of a set of Pd-Net nodes mainly concerning about scheduling, however it will be also responsible for the node bootstraping process, allowing nodes to identify and describing themselves for the first time next to the system. This service could even enable some tasks not related with scheduling, as logging or monitoring, supporting some operations such as turning on or off or activating emergency messages.

Sensor Registry Is a service to allow local physical sensors to describe themselves to the network and initialize the sensing into specific environment services.

Application Store In this component applications are self described and distributed according requests.Display owners may then assess if applications fulfill their content needs and can make subscriptions to their displays. This service is also intended to provide other functionalities like payment and application usage logging.

Applications Are the responsible for generate content to be rendered on the displays, those applications need to be able to describe themselves and be configured to better adapt to each content scheduling scope or situatedness constraints. In this first vision, display applications should be executed remotely and just send the sufficient information to render contents near the displays.

Third-party Content Sources These sources are external to the Pd-Net project scope, they are created and managed by external entities. Hence they are out of our management we could suggest some requirements they could follow in order of provide satisfactory services to the clients, specially those who are created just to enrich signage systems.

The components described above are not yet organized in a final architectural form and, in fact, they could be coupled together in just one physical element, notwithstanding they have to operate in a predetermined way. However we suggest a minimal component division, to decentralize operation and offer a more flexible and expansible system.

In this way is expected that the Pd-Net architecture should have the components and the arrangement as seen in Figure 3.1. As figure shows there are two "regions" where we can dispose components, the *Network Segment* and the *Screen Segment*. In the screen segment should be the parts that are in direct contact with the final users of the system (display nodes with the associated sensors, the physical sensors and the mobile applications running on personal mobile equipments). On the other hand, on the network segment should be all the components that sustain the operation of the displays respective applications. We also present a subdivision of the network segment components to better understand the key functionalities and processes associated to each one of them.

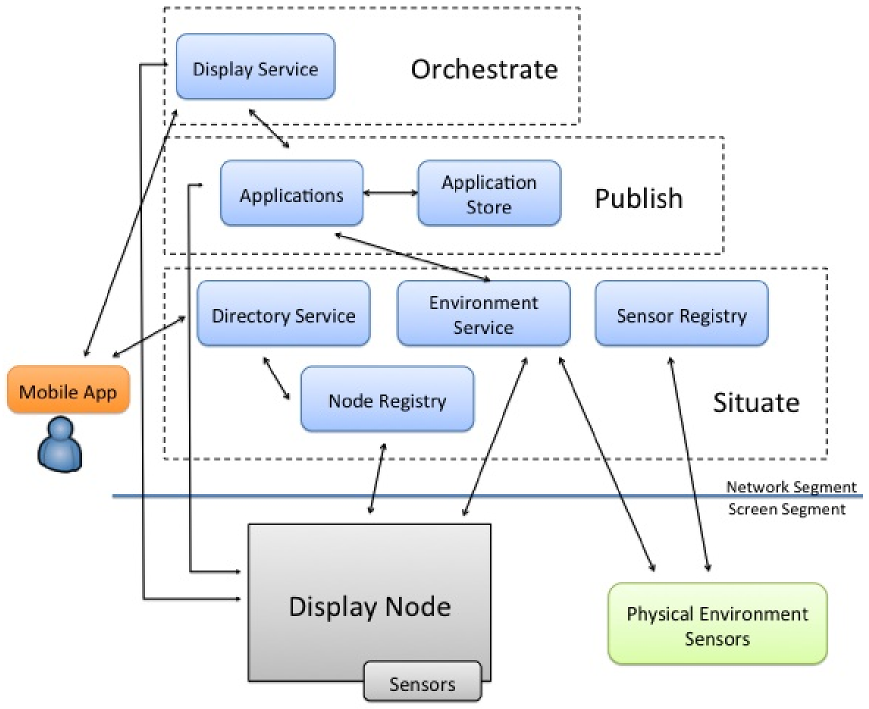


Figure 3.1: Draft of the potential Pd-Net architecture.

The *Orchestrate* layer is responsible for the arrangement of the applications under a certain expected experience of usage of a display or a set of displays, it should comprehend the scheduling definition rules and content selection rules. The *Publish* layer appears as the layer responsible for the creation and distribution of contents to be consumed by the users without having concerns about any scheduling or by who or where the contents will be shown. Finally, the *Situate* layer is focused on provide support for all context situations around displays from the information about the sensors, the on going interactions or bootstrap processes.

3.4 Identify and classify the processes of each network component

In the proposed architecture we can, at a first sight, detect and identify some of the parts where the system can have a more relaxed actuation, possibly having an asynchronous communication, or a more instantaneous synchronous communication. In fact, we will clearly assume on this study that, in a very raw vision, the processes that do not directly influence with the user interaction and feedback can be classified as non critical, having the possibility of being delayed or rescheduled to be treated after. Users quality of interaction is the key goal to achieve with the network performance, so the processes that only take care about system maintenance, for example the process of submitting a new application to the store, have more opportunities to be "slow". This happens because those processes may be delayed for later executions and eventually do not interfere with the users perception of the system. What we propose is to classify the communication processes between components as "Critical" and "Non Critical" to better understand where we the teams should focus their efforts, at middleware and the network level, maybe applying complex scalability techniques to improve the overall performance.

3.5 Non Critical processes

Looking to the Pd-Net abstract architecture we can identify the processes that are non critic by understanding where the system can have a more flexible response time, not compromising the user experience before the system. This important assumption may lead to some uncertainties classifying some processes, however we assume that, if some process do not have a component that do not directly interact with the final user it should be non critical.

Bootstrap The Pd-Net node bootstrapping process (adding one or more Pd-Net nodes to the system) and its subsequent operations, can be treated as non problematic. Firstly, this operation should occur just once for each node, then it does not have to be made in real-time. This configuration process can be made when the network load is lower, for example, at night when the usage of the system is expected to be lower. As this process does not directly influences the user experience it can have the non critical classification. The steps that have to be made to achieve the bootstrapping are:

- 1) Node registration - Each new Pd-Net node contacts a Node Registry to describe itself to the network, establishing ownership and to initialize the required procedures to drive content control.
- 1a) Environment resource registration - Local environment sensors sends its describing parameters to the Sensor Registry service allowing the sensing into the Environment Service.
- 2) Directory Service Bootstrap - Directory service will often contact one or more Node Registry services to gather information about the existing nodes, in order to provide location based services.

Application registration The communication process of registering an application by its developer should not represent a problem concerning the network performance. We assume that such process has a non urgent nature, communication can be delayed until the network has a more relaxed state, without high loads, and it should occur just a few times, while propagating the application description throughout multiple application stores. The application registration definition is:

- 1) Application Registration - Every developer who wants to deploy an application should contact directly one Application store and submit is own application, describing what can it do and what resources it needs. The spreading process throughout other stores, should be transparent to the developer, although he may limit the application usage.

Application selection (Non-Interactive) The process of selecting applications by the display owner, to be integrated next to their Pd-Net nodes also may not have to be ”instantaneously” communicated to the Display Service. This process has little influence in user feedback. We assume that, the major consequence is the impossibility of the node to provide the application while the display service has no knowledge of it. However users do not perceive this fact because they cannot have expectations about the contents of the nodes and do not play an active role in the orchestration phase. In this way these operations may be delayed to be later treated because they do not have real time constraints. To select non interactively applications we need:

- 1) Application selection - Display owners define what applications would be available on their display nodes. As consequence, display service receives the application description from the application stores and builds the orchestration accordingly.

- 2) Application Initialization - To achieve the desired application execution there has to be some initial configuration and scheduling procedures.

- 3) Describing behavior / Schedules to Nodes - Display service communicates to the display node the orchestration information describing its behavior with the references to the remote application.

3.6 Critical and repetitive processes

The identification of the processes that can suffer with more stress situations is an important step to make a more accurate study. With that pre-categorization we can focus the simulation studies mainly to this type of processes. Having more information about such processes, it may be possible to propose alternatives to their design, persecuting the improvement of the system overall performance. These processes have, by nature, to be extremely responsive because they will, very probably, be involved with users interaction, so the response times should meet those user interaction requirements.

Any process that compromises any synchronization has to be a fast and highly interactive process also any kind of user sensing and feedback has to be top priority for the network communication, hoping to achieve a good user satisfaction.

Interactive/context-aware application These types of applications have a very demanding real-time characteristics, imagine an application that depends on a sensorial information to adapt their behavior when a user is near or a interactive game between displays, the interaction opportunity is lost if information does not arrive on that exact instant, leading to a user distrust that can ultimately abandon the usage of that application. The set of steps that compound these application processes are:

- 1) Sensing / situating applications - When the sensors at the nodes detects some relevant change or interaction it contacts the Environment Service in order to describe what is happening.
- 2) Content generation by application - Applications should be able to render content accordingly a set of previous interactions with the viewers.
- 3) Environment Service feedback - After receiving the sensor information the Environment Service should provide some description to the right applications so they can adjust their execution accordingly.

Privacy-aware context aware application These applications, executing in the viewers mobile phone, have strong requirements respecting response times.Users experience can be severely damaged if a user performed interaction does not return a response at the expected moment, usually immediately, in real time. To have applications running on any mobile device, probably, several steps have to be made until the user can start his interactions. As example, an application needs to gather information about nearby displays before the user can start the expected interaction. From the beginning of that process until final user interaction the system has to be prepared to accomplish those requirements in not more than a few seconds. Failing to accomplish this goal, users may disregard the system usage, possibly, resulting in some economical damages to applications and display owners. The steps that an application of this kind has to achieve are:

- 1) Look-up near displays - Mobile applications should contact the Directory service to get the information about the near displays.
- 2) Select application in Display Service - The Display Service receives from the clients the information about what application should be selected.
- 3) Request behavior - Display service sends to the Pd-Net node the user request to change the execution pattern on the node.

Although third-party content sources are out of the spectrum of the Pd-Net network they should be sufficiently responsive to be part of the system. Users experience may be extremely affected and compromise system usage if the response times requirements were not satisfied.

3.7 Possible bottlenecks situations

The potentially large-scale nature of these pervasive networks raises lots of scalability barriers that have to be overtaken. The dynamism of content and behavior in different usage scenarios of the network had to be taken in consideration, so it is very important to point out some of the variations

that can lead to overall performance decreasing. In this way, the constantly changing number of the following items have to be taken into consideration. Bottleneck points are very likely to emerge, not only associated with one specific dimension but with the combination of other dimensions, leading to a more complex analysis.
The situations and processes that we have anticipated are:

Applications used in many places Assuming that applications running on displays are Web applications, the potentially large number of applications subscriptions could lead to server overloads that have to be managed;

Applications that generate many interactions or that heavily relies on sensing Applications that generate many interactions may generate to problems next to the Environment Service, overloading them with requests asking for sensing information;

Directory service serving mobile nodes Dealing with multiple simultaneous requests from mobile nodes may slow down Directory Service responsiveness;

Synchronization contexts with a set of nodes operating in a tight interaction relation Applications that demand a rigid synchronization between nodes may have its execution delayed by their communication;

Display nodes The increasing number of display nodes can lead to problems concerning the synchronization protocols, naming schemes among others;

Pd-Net node users The number of simultaneous users in a node or in multiple nodes may affect the system responsiveness, overloading the node with requests;

Application interactions The highly interactive applications may have to exchange, in short periods of time, large amounts of data compromising the responsiveness of the system;

Content hosts, content producers and content items The constantly augment in number of these three, can lead to registration and content discovery higher dissemination times;

Simultaneous application executions The processing, scheduling and synchronization demands on the node can be compromised by the number of simultaneous execution;

Display owners Can lead to high rates of management messages and to privacy or security models;

Content types The increasing number of different data types can degrade the system performance, for example, video streaming can lead to a high network overload;

Interaction sensors Multiple sensors providing sensing information can overload the environment servers with to much information delaying the process of create context aware scenarios.

The previous points will very probably appear during the lifetime of our pervasive system so it is very important to have them in mind. Our effort to describe them, even not very formally, has the objective to alert the developers that many problems may emerge on this pervasive system due to multiple factors. Those factors that we have tried to point out should also be, later on, on our simulations assessed if they really exist and in what circumstances they appear.

Having them in mind will certainly help us to walk around some problems in the early stages of design of the middlewares and network that will operate on the Pd-net system.

Chapter 4

Simulating Pervasive Displays Networks

Pervasive displays networks with their multitude of connections and components have inherently a high level of complexity when considering creating a simulation model.

In this context, we propose the creation of a simulation scenario as a key step to have a starting point to understand the characteristics of those networks. In this chapter will be presented and discussed the entire process of creating a case study with some alternatives and decisions. Trying in the end to have a clear idea about the requirements that have to be satisfied in order of achieve the creation of a simulation testbed for pervasive displays networks. Describing building blocks of a simulation scenario with all of its inherent characteristics, from the network topology to the present interactions, and the expected resulting data.

4.1 Simulation scenario

To have a fully functional simulation scenario various steps have to be accomplished, namely, the creation of a network topology with all the physical or abstract components that make part of the system, their connections, the communication workflow and the adjustments to the parameters that are suitable to our demands.

The communication workflow will be justified with the creation of an application scenario that considers the relevant interactions within the system. As we have seen, the categorization of processes in "critical" and "non critical" will guide us with the creation of the application scenario, by clearly showing the execution patterns to be analyzed.

4.2 Network topology

The base layer to collect data throughout simulation is the creation of the network component nodes and their respective interconnections, implementing their topology.

The ideal scenario is to obtain a modular simulation model to allow us to program our components with different functionalities and reproduce them as many times as we want. What we envisioned to design our topology was clear division of the core system components from the display nodes and the system users, as seen in Figure 4.1.

We have made a separation between sets of display nodes and mobile users of the system, clustering them, to achieve a better definition of "regions" of displays. We imagine those "regions" as different villages, cities or countries, hoping with this decision to better target the further simulation parameter values into those execution scopes. Also, as the ratio between display node users and mobile users may be significantly different in certain situations, this division seems to be a good way to focus our attention on those details.

On the other hand, the underlying infrastructure, on the *Network Segment*, pretends to be a open network that cooperates and where the components work together to provide us the expected resources, Figure 4.2. Therefore we can have functional components that could be implemented and instantiated in many ways.

Having this high level functional component modules we could hide all the inner complex decisions and define them later. This happens because we do not know yet how are the middlewares implemented and how will be the physical distribution of those components. The efforts of replacing the

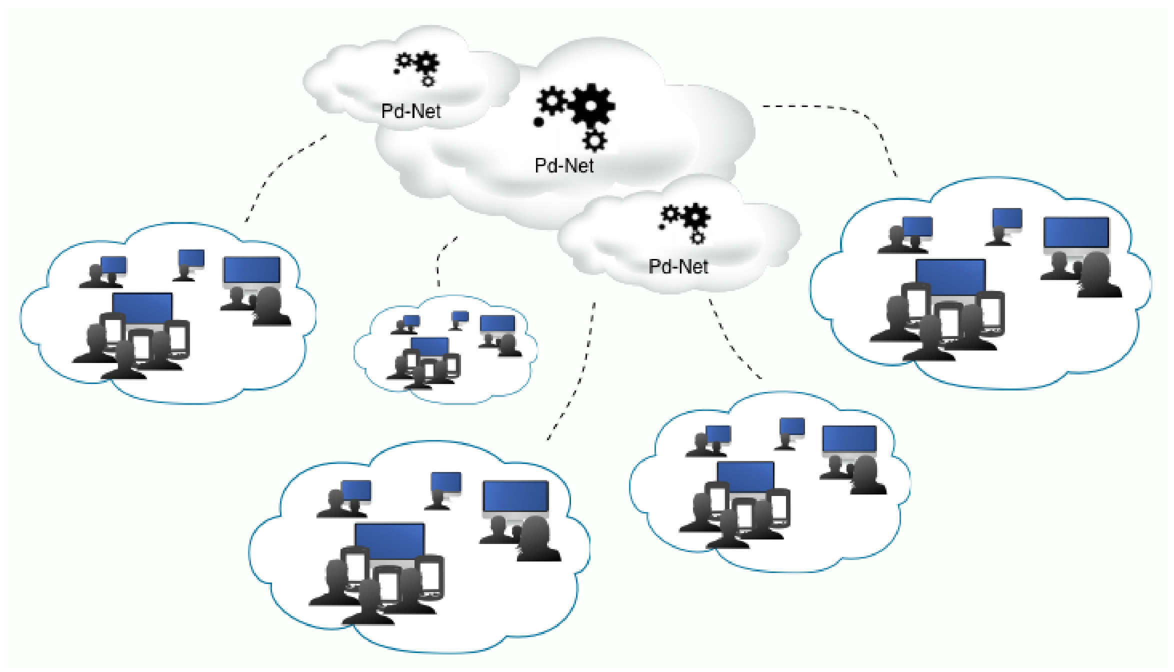


Figure 4.1: Pervasive Display Network conceptual topology.

modules with the adequate middleware decisions, to solve specific problems of those parts of the system, will be kept in stand by, waiting for the results of our first simulations and advances made by other project working teams. However with the creation of adequate modules and with the correct simulation parameters we can perfectly represent the characteristics of pervasive networks with their heavy demands of computational resources near the populations of users around the displays.

We can connect, multiple displays and mobile users clusters to one single pervasive architecture not thinking, by now, on its operational strategies to serve them with content, we just have the way of how is the communication flow between functional components. This modular approach should also provide an added value, in the future, for implement and test the best scalability technique for each particular component by just replacing its instance. Its important to emphasize that those components that make part of the *Network Segment* will only have one instance of each one, isolating, with this approach, more easily the bottlenecking points of the system.

This representation of our pervasive network have a good resemblance with the design of the traditional mobile networks [10] that is one of the best and well stablished example of a pervasive network. That is because we make the parallel between the cells division and our display and mobile users clusters having the same underlying infrastructure, maybe with different hardware capabilities, that work and cooperate together to serve us with their resources. Of course there are significant different objectives within these two networks and for now its complexity are not even close to each other, however a brief look on some used strategies may be useful to the study of our system.

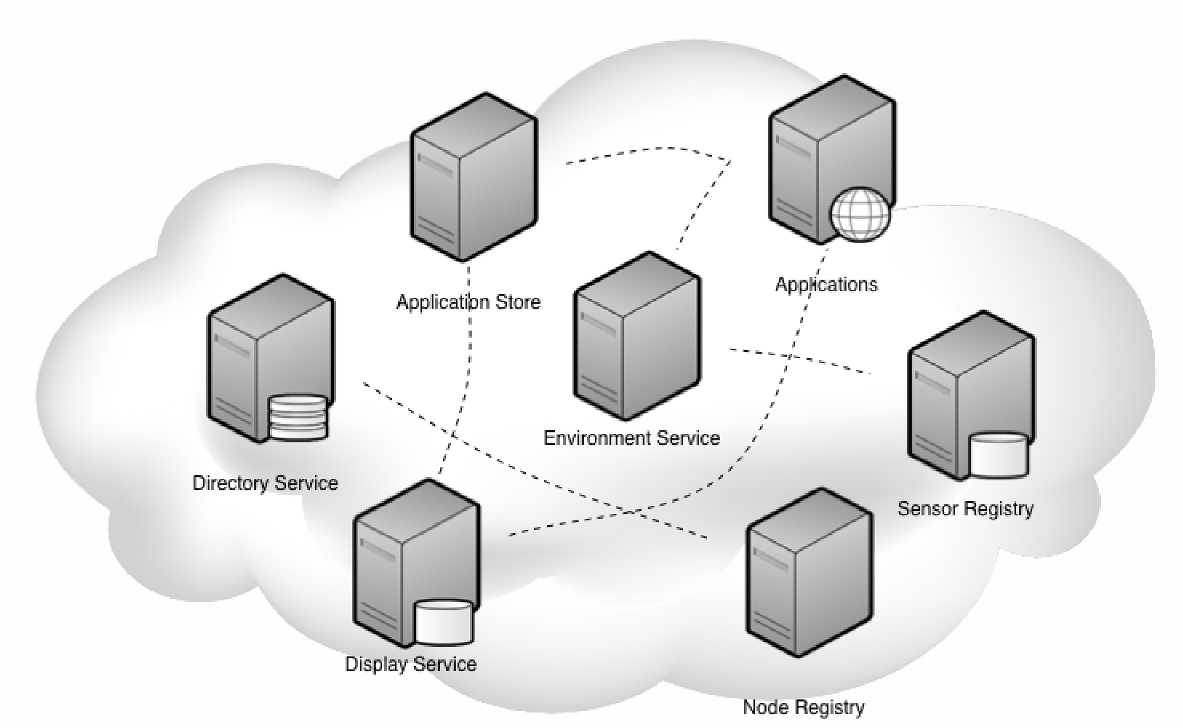


Figure 4.2: *Pd-Net* high level component modules.

4.3 "Application"/Usage scenario

The second major step to create a simulation scenario is the creation of a flow of execution that will correspond to some application *modus operandi*, in order to provide some realism to the simulation. Those execution patterns should correspond to some real or potentially real application. With this we expect to better understand the needs and the characteristics of these pervasive applications. The *Walk to School programme* [8] appears as a good example of a realistic application scenario. It is probably the envisioned scenario that have more explicit and implicit user interactions as well a more demanding execution in terms of resources of the underlying system so it emerges as a good starting point to deploy those interactions on a simulation framework.

Walk to School programme - Influencing Behaviour

Jack is six years old and participates in his local "walk to school programme" - an initiative aimed at increasing fitness among school children and addressing childhood obesity. To encourage participation among children a simple game has been deployed on the area's public display network. As Jack walks to school he passes a number of public displays. At each display he sees a cartoon character that gives him an update on his own progress and that of his friends. By visiting the displays Jack also collects "golden leaves" on his mobile phone - when he has enough of these leaves his school redeems them form a sticker book.

With this small introductory scenario we may now detail the possible interactions between the system and users creating a flow of execution expected to be performed by the users in certain daily periods. However, to do that, we have to take in care some assumptions and requirements. We do not pretend to simulate all the stages of the deployment and usage of the system and the application, although sometimes major problems of the systems came from the most unexpected interactions.

We have also to clearly separate the interactions that are dependent of a specific usage scenario from those who are always present to sustain the systems integrity and functionality, so we could easily create different or more scenarios of usage, creating a more real system operation, without having to create from the scratch every single interaction. Once more, we claim that multiple assumptions have to be taken in care because the deployment of all interactions and execution patterns into a simulation framework appears to be almost unfeasible.

4.4 Scenario specific interactions

There are interactions that only have meaning in some specific scenario usages, for example, every mobile application will potentially have an interaction flow absolutely different from another one because of the demands of which one of them. In our *Walk to school programme* scenario we have an mobile application were kids can bootstrap to the system and collect their *"golden leaves"*, this specific set of actions create a usage pattern that is necessarily different from another one.

Figure 4.3 depicts the interactions generated by the action of collect a golden leave. As the diagram shows, when a kid wants to collect a leave has to explicitly contact the system, starting his mobile *Walk to School* application, It automatically contacts the *Directory Service* to gather the list of displays near him, performing a *LookUp* (Figure 4.3 step 1), then he has to choose the display that he want to interact with. Now this "check in" process is made next to the *Environment Service* (Figure 4.3 step 2). After that he can perform the interactions that lets him to collect the leave (Figure 4.3 step 3). Those interactions could have a wide range of possibilities, assuming that the application have more options than directly collect the leave, however we can define a different number of interactions that each person would realize until complete the gathering of the leaves. Collecting them may also, lead to the need of rescheduling or update some state, thus the application may have to contact the system in order to request some behavioral change of an individual or a set of displays (Figure 4.3 step 3.1). The system, more specifically the *Display Service* will be in charge of contacting the displays

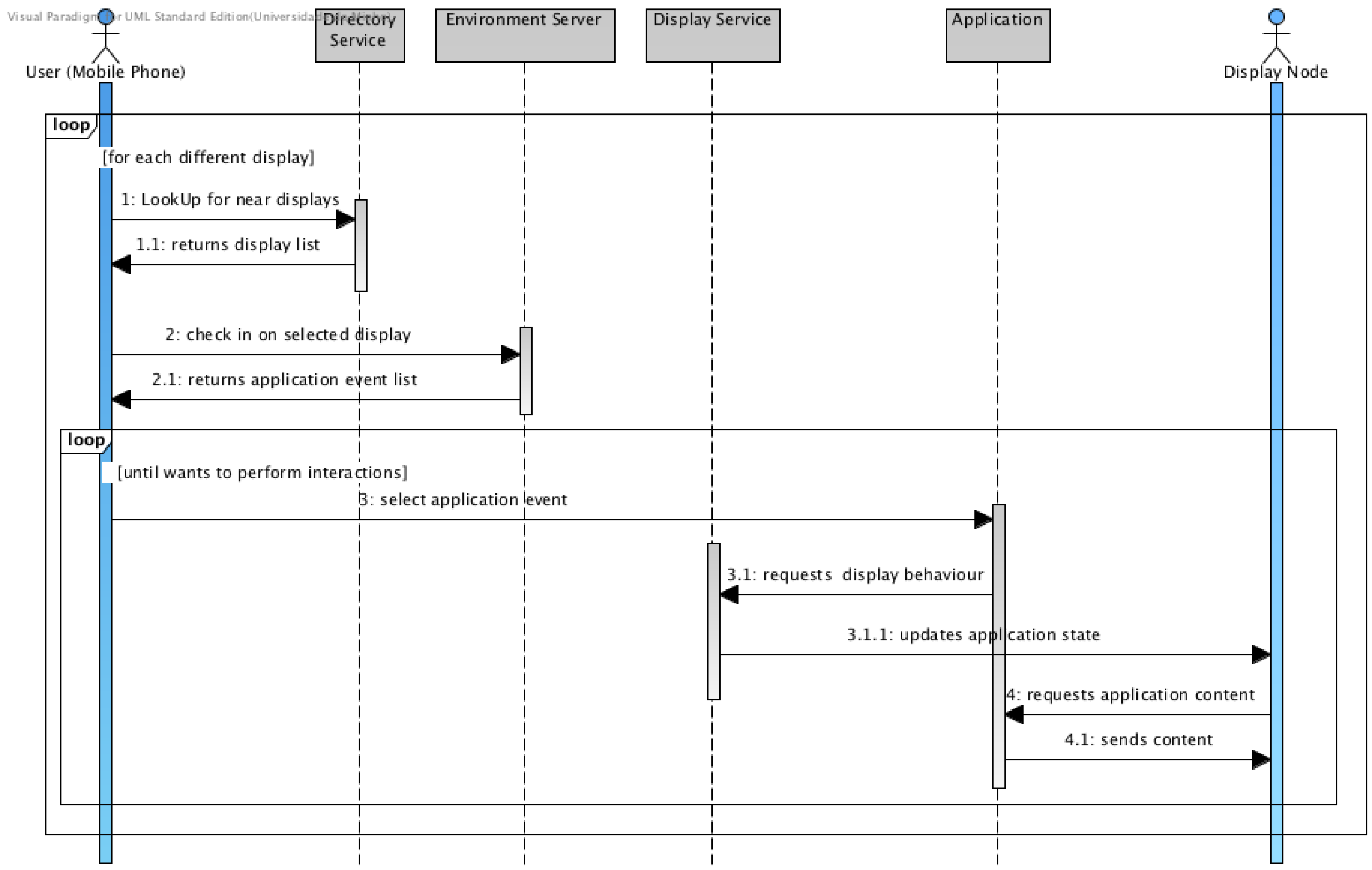

Visual Paradigm for UML Standard Edition(Universidade do Minho)
User (Mobile Phone)
Directory Service
Environment Server
Display Service
Application
Display Node
loop
[for each different display]
1: LookUp for near displays
1.1: returns display list
2: check in on selected display
2.1: returns application event list
loop
[until wants to perform interactions]
3: select application event
3.1: requests display behaviour
3.1.1: updates application state
4: requests application content
4.1: sends content

Figure 4.3: Privacy Aware - Context Aware interactions.

passing instructions of what the displays have to do in the immediate future (Figure 4.3 step 3.1.1, 4 and 4.1).

All these interactions are part of a utilization context that we call *Privacy Aware - Context Aware* and it has to occur every time a user wants to collect a leave on a different display.

The previous interaction sequence may generate a global report to all displays that are using the application. For example, a special leave that can be collected from now on, or that a new record has been beaten. That can be seen on Figure 4.4, the *Dispay Service* receives the global message request (Figure 4.4 step 1), computes the right receivers and sends a broadcast

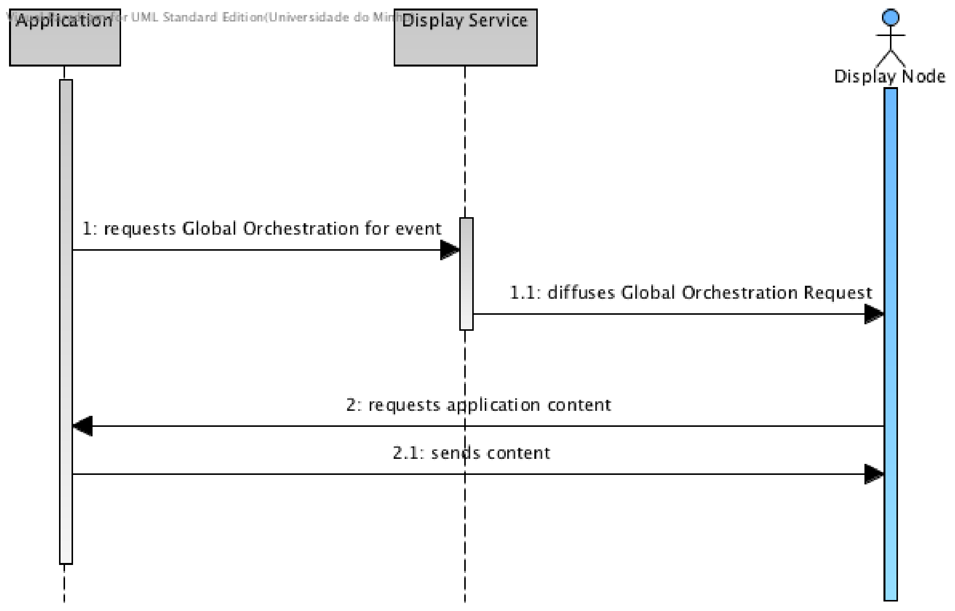

Application
UML Standard Edition(Universidade do Min
Display Service
Display Node
1: requests Global Orchestration for event
1.1: diffuses Global Orchestration Request
2: requests application content
2.1: sends content

Figure 4.4: Global Event interactions.

message to the *Display Nodes* (Figure 4.4 step 1.1). The display nodes that receives the global message request will then contact the applications to update their inner state (Figure 4.4 step 2).

Pervasive display systems have, the possibility of somehow sense nearby persons automatically adapt their behavior accordingly. In our simulation, physical sensors, could sense the presence of the individuals that are walking on the display neighborhood, contacting the system, sending or updating the state around them. Figure 4.5 depicts the interactions that compose the *Implicit Context Aware* scenario of interactions. Thus, when physical sensors sense some relevant context change or sense some specific user, it contacts the system to send that information (Figure 4.5 step 1). The *Environment Service* has the function of collect that information and make it available. *Applications* on the other hand, may now, frequently query the system to update their internal information and send them to the displays (Figure 4.5 steps 2, 2.1 and 2.1.1).

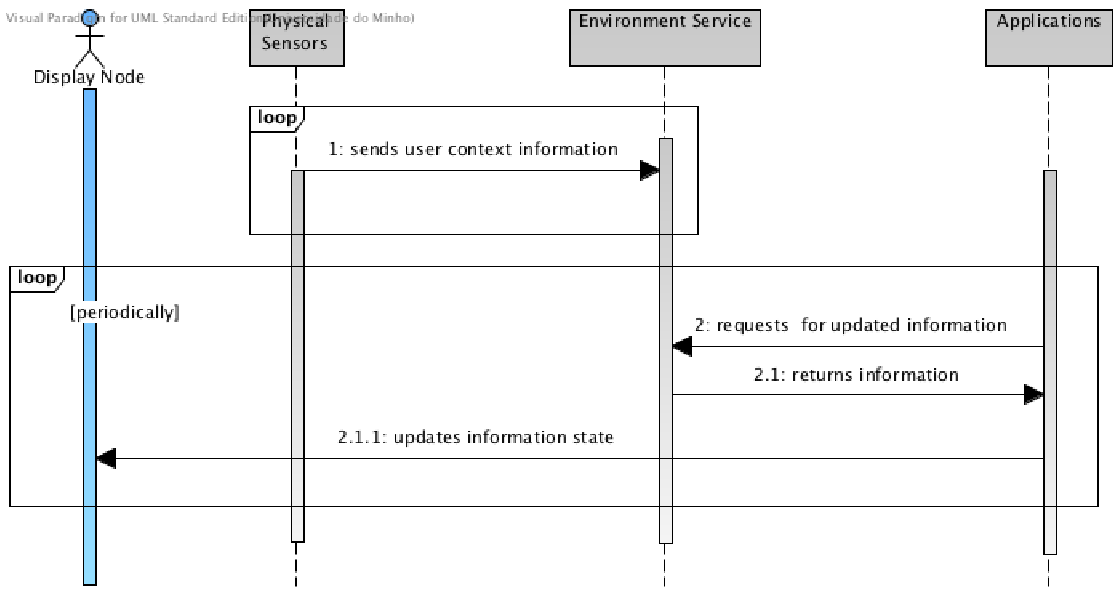

Visual Paradigm for UML Standard Edition(Universidade do Minho)
Display Node
Physical Sensors
Environment Service
Applications
loop
1: sends user context information
loop
[periodically]
2: requests for updated information
2.1: returns information
2.1.1: updates information state

Figure 4.5: Implicit Context Aware interactions.

Another interaction possible to be seen in almost any display systems is the one where the users interact with the system directly with the displays with some touchable surface. In this case Figure 4.6 presents what may occur after that direct display interaction by some user (Figure 4.6 step 1). After interact with the application, users suppose to have some kind feedback (Figure 4.6 step 1.1.1.1). However that feedback, in order of having correct contents, may have to be enriched with some contextual information (Figure 4.6 step 1.1 and 1.1.1). Also, after those direct interactions the applications may have the need to broadcast some message, contacting the *Display Service* to manage that request (Figure 4.6 step 1.1.1.2). The following steps to accomplish that request have already been seen in Figure 4.4

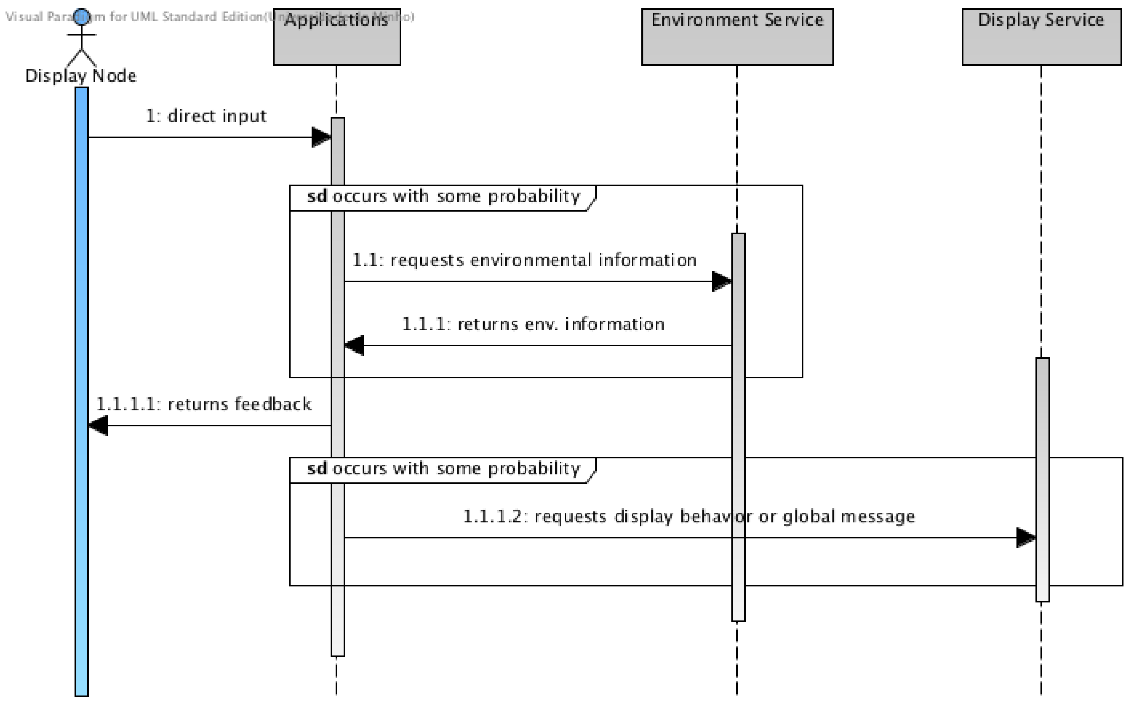

Visual Paradigm for UML Standard Edition
Display Node
Applications
Environment Service
Display Service
1: direct input
sd occurs with some probability
1.1: requests environmental information
1.1.1: returns env. information
1.1.1.1: returns feedback
sd occurs with some probability
1.1.1.2: requests display behavior or global message

Figure 4.6: Direct Display interactions.

4.5 Non specific scenario interactions

This set of non specific scenario interactions are performed to maintain systems integrity independently of the usage scenario. The separation and categorization of those interactions are required to have a tight system testbed. With that separation we can also create more scenario specific interactions on top of these non specific scenario interactions and simulate them at the same time, not having to perform significant changes on previous implementations. In the end we expect to have more realistic results about the performance of our architecture.

Although being, at a first sight, non critical interactions, because apparently they do not influence directly the user experience, they are important for several reasons. Particularly, some of communications cannot be delayed or cached, if so, they could lead to some system incongruence, that would damage users quality of information provided and the mere fact of existing these set of interactions, just for system own maintenance, between his components could lead to a systems overload, compromising the adequate response to the users, affecting their quality of experience. In our simulations

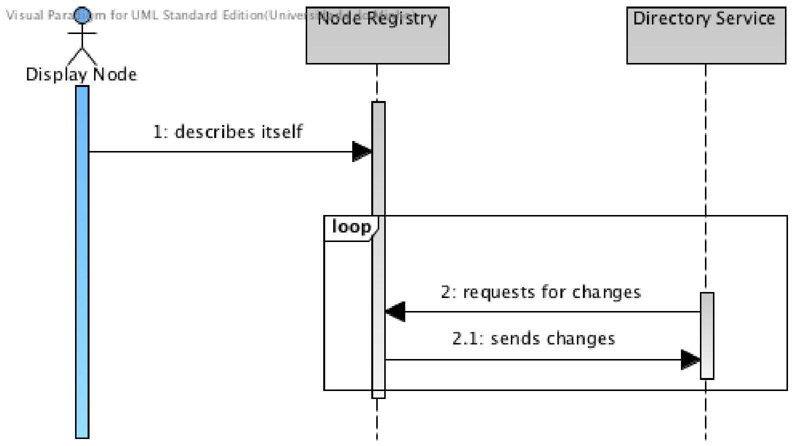


Figure 4.7: *Display Node* interactions.

we have only implemented the ones that would help us to create data structures containing information that would be useful to achieve integrity and coherence about the displays, facilitating the production of the simulation code.

The Figure 4.7 depicts the two implemented interactions, that are the *Display Node* bootstrapping to the system, in the *Node Registry*, and the consequent polling from the *Directory Service* requesting for some relevant update.

Although, despite not implemented, we can point out some interactions that have to be inevitably performed to maintain a correct system state. Those interactions and can be seen as requirements and assumptions to achieve a functional integrity of the system. Among others, the interactions that we assume in this computational environment, in a very raw vision, are:

- 3rd party *Applications* are registered on the *Application Stores*
- *Applications* are subscripted by the *Display Nodes*
- *Physical sensors* are registered at the *Sensor Registries*
- Users have already their profiles kept on the system
- The system is able to schedule any *Application*, explicitly or implicitly
- The system is constantly logging his state and operations
- The system is able to detect and recover from incorrect functional states

With this, simulations will only represent the states of interaction by the users with the system, assuming that all the system are loaded and prepared to handle the requests.

4.6 Simulation Characteristics and Parameters

One of the problems with the study of pervasive network architectures is the uncertainty about the dispersion of the functional and physical elements that are part of the system. It is almost impossible to have a clear idea about the normal operation of a pervasive system in terms of the presence of the components on the network, specially such a complex one as the proposed in this project. However, in our case, we have to address some parts of the system, simulating their presence on local networks (LAN) or in more broad networks such as Metropolitan Area Networks (MAN) or Wide Area Networks (WAN). With this decisions we expect to obtain more realistic results, embedding average times of the latencies on those networks .

Also, as the system is intended to work above the Internet infrastructure, there are several measurements and statistics about the global network latencies [1] [2] that can help us to adjust the communication times under various conditions, from local to intercontinental communication.

Thus some parameters have to be mapped to connections between components of our system, to give the idea of which one of them may be in a more or less distant situation. Parameters like connection delays, packet loss ratios, connect data rates among others. Other parameter that have been taken into consideration is the number of requests that each component is capable to handle and how many connections can be queued for later treatment, this has a major importance because it directly influences the response times or even message drops under extreme operations.

The average processing time of each request on each component is also one of the parameters that could largely influence the system fluidity and one of the most difficult to predict. In fact it is almost impossible to have a correct value on this parameter, depending on each component algorithm implementation the associated time complexities [3] could vary from a *T(n)=O(1)* to a, for example, *T(n)=O(n!)*. Having this, the values used should be what we expect to be a reasonable processing time, notwithstanding, different values could be assigned on each execution in order to have more accurate results.

[1] http://ipnetwork.bgtmo.ip.att.net/pws/global_network_avgs.html
[2] http://www.verizonbusiness.com/about/network/latency/
[3] http://en.wikipedia.org/wiki/Time_complexity

4.7 Scenario specific simulation parameters

On our simulation we have made a clear separation concerning the parameters that are assigned to some application scenario to those which are part of the core infrastructure. With this, we can, on the same simulation, have different scenarios of usage of the system, getting even more realistic outputs concerning the usage of the system with a multitude of interaction patterns.

In this stage, our implementation only has the *Walk to school programme* interactions and a set of parameters have to be taken in consideration. Among other more technical details most of the parameters that we envisaged are related with the Human natural behaviors. Therefore to mimic the natural usage of the system we created the following parameters:

- number of *Display Nodes* visited on the way to school and the respective time between them
- number of interactions expected to occur on each display to collect *"golden leaves"*
- number of direct interactions on the *Display Nodes* and time interval between them

It is noteworthy that these interactions should be made, almost entirely, by people with very young ages, probably between the 5 to 14 years. Thus the time and frequency used on the parameters should be necessarily different from other application scenarios, that focus on older people. These differences, even slightly, can have a major impact on the system overall performance and acceptance as well as can give us wrong result data sets.
Complementing the previous parameters we have also accomplished some regarding the operationality of the system. Deriving from the persons interactions or the sensorial information gathered, the system could have the need to adapt to provide more accurate results and feedbacks to the users. The *Applications* or the *Physical Sensors* could trigger some behavior in other parts of the system (as seen previously) and the frequency of those phenomena are regulated by the following:

- frequency of requests to the *Environment Service* by the *Applications*
- frequency of requests to the *Display Service* requesting a change of behavior or a global message
- average number of displays that should receive a global message

All of simulation parameters that we have seen have to be flexible enough to create results with some validity and significancy on some real implementation. Depending on the expectations of usage of the system and interaction types, the input values should assume some value between a given interval or representative distribution. The differences on the interaction types and times could create multiple stress situations that we expect to comprehend and give correct guidelines to solve them. We could also say that with these three steps accomplished we have a complete simulation scenario and we could start to collect and analyze the output results. The various input values to the parameters should provide us the sufficient amount of data to detect the components that should be the focus of our efforts, maybe applying a scalability techniques to a overcome possible bottleneck.

Chapter 5

Simulation Deployment/Implementation

Simulation represents a very powerful tool to those who have to design, test and deploy networks, specially those who tend to be large with a multitude of components and interactions. Pervasive displays networks have a predisposition to have grow indefinitely so the previous study provided by the simulation tools can save lots of money and time anticipating problems that would arise. Having a strong, flexible and extensible simulation model has a major importance to obtain correct values to help us in our decisions.

In this chapter we intend to cover all the decisions related with the actual implementation of a simulation model for pervasive displays networks, since the choice of the framework to the chosen input values to produce results.

5.1 Network Simulation Frameworks

On the field of the discrete event simulation many frameworks make part of our range of choices. However, since the beginning *OMNeT++* [29] was a reference. Its creation had the purpose of being a network, multiprocessor and distributed systems simulator, notwithstanding, it could be aimed to other simulation types, trying to fulfill the gap between the purely research-oriented open source and the commercial payed distributions. It has been made public in 1997 and, since then, multiple models have been created and enhanced to facilitate the creation of the build blocks of the simulations.

The development of this simulation framework from the early days was driven by some principles: a) *"enable large-scale simulation, simulation models need to be hierarchical, and built from reusable components as much as possible"*; b) *"simulation software should facilitate visualizing and debugging of simulation models in order to reduce debugging time"*; c) *"simulation software itself should be modular, customizable and should allow embedding simulations into larger applications such as network planning software"*; d) *"data interfaces should be open, should be possible to generate and process input and output files with commonly available software tools"*; e)*should provide an Integrated Development Environment that largely facilitates model development and analyzing results"*.

Taking these principles and considerations, *OMNeT++* works in a modular form with message passing between modules. There are two types of modules, *simple modules* and *compound modules* (Figure 5.1), that can be combined in a hierarchical form without any limits. Written in *C++*, those modules, may perform almost any desired action when receiving a message or during the simulation lifetime. Those modules can communicate via *messages*, that could contain any desired fields and information structured either representing some existing protocol or some personal specific simulation demands. Messages, typically, are sent between module *gates* (graphically in blue on Figure 5.1), linked together with *connections*. Yet, messages can be sent directly to the desired modules. Every module can have associated *parameters* that contain the information needed to correctly configure its

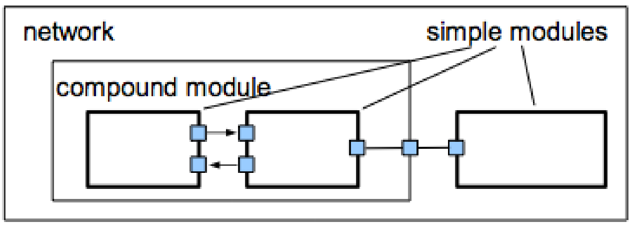
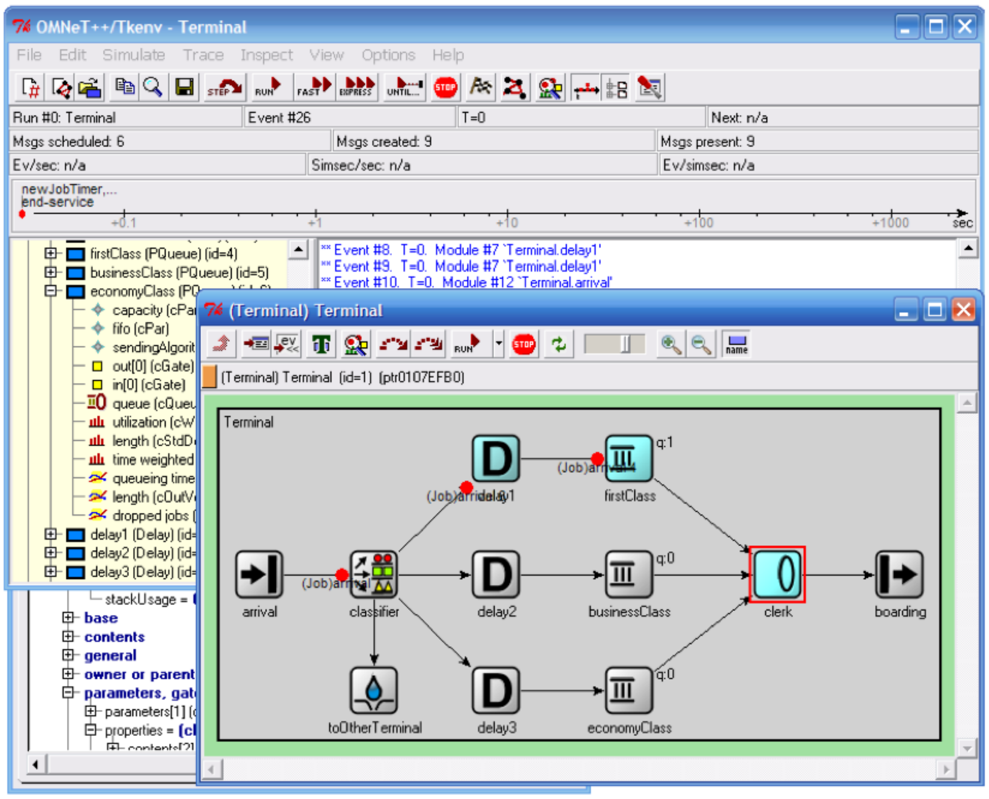


Figure 5.1: Internal module structure of *OMNeT++*.

behavior. They could assume values that can go from the simplest boolean value to some random number defined in some distribution or even strings.

To have a running simulation we have to dispose modules in some form to create a topology. This can be done using the *Network Description* (NED) language. A complete NED definition should encompass the simple and compound module definitions as well as the network definitions. Simple module have on it a description of the gates and the all associated parameters and compound modules have to be complemented with the submodules interconnections. The last layer of a simulation has always to be a *network*, that is a compound module self-contained from which the simulation will be executed. As expected in such a modular and reusable simulation framework there is



Run #0: Terminal | Event #26 | T=0 | Next: n/a
Msgs scheduled: 6 | Msgs created: 9 | Msgs present: 9
Ev/sec: n/a | Simsec/sec: n/a | Ev/simsec: n/a

newJobTimer,...
end-service
+0.1 +1 +10 +100 +1000 sec

```
firstClass (PQueue) (id=4)
businessClass (PQueue) (id=5)
economyClass (PQ
  capacity (cPar
  fifo (cPar)
  sendingAlgorit
  out[0] (cGate)
  in[0] (cGate)
  queue (cQueu
  utilization (cW
  length (cStdD
  time weighted
  queueing time
  length (cOutV
  dropped jobs (
delay1 (Delay) (id=
delay2 (Delay) (id=
delay3 (Delay) (id=
  stackUsage =
base
contents
general
owner or parent
parameters, gat
  parameters[1] (
  properties = (cl
```

```
** Event #8.  T=0.  Module #7 `Terminal.delay1'
** Event #9.  T=0.  Module #7 `Terminal.delay1'
** Event #10.  T=0.  Module #12 `Terminal.arrival'
```

(Terminal) Terminal (id=1) (ptr0107EFB0)

Figure 5.2: *Tkenv* User interface.

a clear separation between the generic simulation scenario and the scenario specific parameterized simulation. To achieve that, *OMNeT++* also has the

.INI files where all of the parameters can be initialized with different values according the reality that we pretend to simulate. As a resume all of these file types and specification languages can the easily mapped into:

- behavior → C++ files
- topology → NED files
- initial values → INI files

Not less important this framework has a built in a powerful tool of analysis of the simulation outcomes. Usually the processing of the collected data is one of the most time consuming tasks for those who make simulations. This framework has a simple form to produce and tune output graphics saving precious time, time that can be used to improve the model.

Running simulations can also have a visual debug using the *Tkenv*, it can be used in three different ways: automatic animation (actual visual message exchange and node state changes animation), module output windows (textual debugging and tracing) and object inspectors (to follow object state or content). To accomplish all of this, *OMNeT++* gently offers an Integrated Development Environment (IDE) that enables us to easily produce the source code of our simulations.

Besides *OMNeT++*, another simulation tool that may compete with it, is the *NS* (in versions 2 and 3). In fact it is the broader used network simulator by the academics. It is has been created starting from early developments on the discrete event-based simulation techniques.

Unlike *OMNeT++*, *NS-2* has the goal of being exclusively a network simulator, so it does not have a clear separation between the infrastructure and the simulation models. *NS-2* [2] lacks supporting for hierarchical models, it uses Tcl scripts to define the simulation models and the network topology, and uses *C++* language to implement simulation kernels and components. The big problem with this unclear separation is that makes the production of reusable models and components very difficult as the creation of graphical editors almost impossible.

The *NS-3* [12] tries to overcome some handicaps existing on the previous version. It cannot be considered an evolution of *NS-2* because the simulation modules are not compatible between versions. It is also written in *C++* however it has been improved with a *Python* interface to develop scripts, instead of Tcl scrips. Some of the *C++* design patterns were implemented to improve it, callbacks, smart pointers and templates are example of that. Other aspects were improved like an alignment with real systems and protocols and the support to easily integrate other softwares without the need of rewrite the simulation models.

Even, considering performance [30] *OMNeT++* has a word to say, it can be less memory consuming than *NS* versions letting his simulation have a bigger scale, fundamental on pervasive networks.

Other simulators exists and besides they have not been tested or used by us, including both *NS* versions, a major research regarding simulation frameworks were made.

Simulators like *J-SIM* [23], *SSFNet* [7] or *GloMoSim* [32] are very focused to study very particular problems and they are tuned to provide the best results for them. They were also not taking into consideration because there are almost abandoned since 2005 due to their low acceptance.

By all of these aspects, we have chosen *OMNeT++* as our ally to attack the implementation of our simulation models.

5.2 OMNet++ Implementation

The implementation of our model is highly modular, the component modules can be instantiated or replaced by others lately developed. In this way each functional component has its own module that can have its own technical characteristics, depending on the hardware or software interactions intend to be studied.

At the top level we have implemented the topology, *FullArchitecture* (Figure 5.3). With this kind of topology what we intend to achieve is a better way to subdivide the displays and mobile users clusters, representing different physical regions where the elements of the screen segment are deployed.

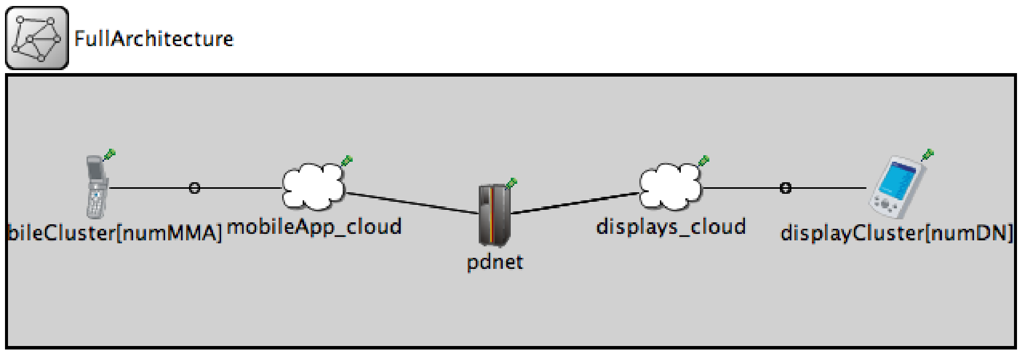
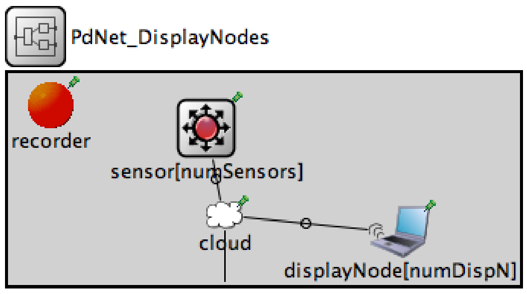


Figure 5.3: Full implemented architecture.

Then we just have to configure the connection delays associated to each cluster remaining the underlying pervasive architecture the same. This implementation also helps us to easily define the number of entities on each region giving us more control on the simulation results. Those two cluster



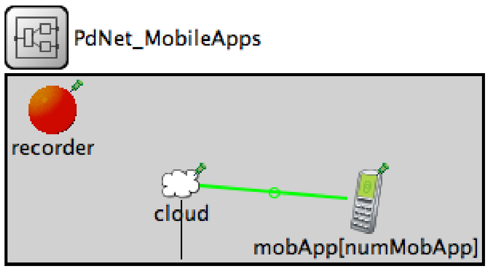


Figure 5.4: Display node module

Figure 5.5: Mobile users module

modules can be seen in detail on Figures 5.4 and 5.5. On the left we can see the sensor and display node submodules that depending on the parameter values (*numSensors* and *numDispN*) will have more or less instances on the running simulations. The right image shows us the module that only contains the mobile users, represented by the *mobApp[*]* submodule. Both of them have an instance of the *Recorder* module to collect all the elected values for each cluster type, forming datasets to be later analyzed.

All of the communication between modules are made through input or output *gates*. As the number of instances of each entity is unknown we have created *clouds* that, concentrating entities connections makes easier the task of creating and replacing modules by new ones. This happens because the number of external interface gates of the module is always the same. We illustrate this issue on the following listing of code.

Listing 5.1: Example NED source code

```
module PdNet_MobileApps{
    parameters:
        volatile int numMobApp;
    gates:
        //external
        inout dispServiceG;
        inout dirServiceG;
        inout envServiceG;
        inout applicationsG;

    //module components and properties
    submodules:
        mobApp[numMobApp]: MobileApplication;

        cloud: MobileAppsCloud {
            gates:
                g[numMobApp+4];
        }

        recorder: Rec;

    connections allowunconnected:
        //internal connections
        mobApp[i].g <--> cloud.g[i+4] for i=0..numMobApp-1;

        //outside network gate
        cloud.g[0] <--> dirServiceG;
        cloud.g[1] <--> dispServiceG;
        cloud.g[2] <--> envServiceG;
        cloud.g[3] <--> applicationsG;
}
```

With this example, we see that the module that creates the mobile users cluster (*PdNet_MobileApps*) has four gates interfacing the pervasive architecture, cloud vector positions 0 to 3, while the all instantiated submodules connect to the cloud on the subsequent positions.

All of the clouds presented on the topology have the same principles, however the behavior of each one could have significant variations, depending on the messages that they have to relay.

The pervasive architecture that should live on the network segment is fully represented on Figure 5.6. There we can see the modules and the connections between them. In our first simulation attempts, the functional component modules do not have implemented any load balancing or load distribution techniques, meaning that we are not yet concerned about scalability solutions

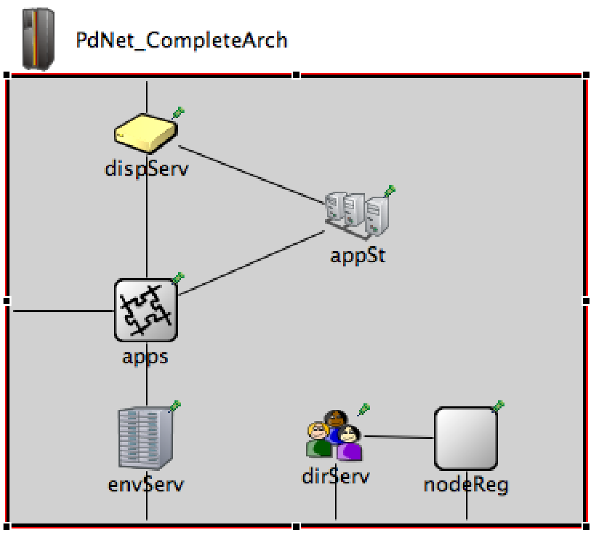


Figure 5.6: Full implemented architecture.

for each part of the network. Instead we intend to study how the cooperation between those components degrade the overall performance of the network. The main focus is on the analysis of the high level interactions that occur within the system for each application scenario. Hoping with that to improve the communication flow of the applications and the underlying supporting infrastructure. Notwithstanding, the modules are in a well defined and enhanced implementation, like a real testbed demands, with the objective of supporting system evolution as studies progress.

All of the seen modules and submodules and their distribution form the network topology has was used on our simulations.

5.3 Component Modules behavior and Message definition

Our simulation has three types of behaviors associated with the components. Firstly, as we have seen we have the clouds that have only to relay messages between modules. Then we have the core network components that in a first sight act as standard web servers. Finally we have the component modules where the interactions could be triggered.

The mobile applications, the physical sensors and the displays feed the system with messages that will be digested by the pervasive architecture to provide some user response. To achieve that, we had to define the messages to be transmitted and the set of steps to be made on each component.

The relay implementation of the clouds works as a simple router, it receives a message and accordingly the receiving gate it makes a two step analysis: first it verifies the message destination and then verifies the message type. Depending on each message type it will give the correct treatment computing any necessary information and redirecting to the right destination.

Listing 5.2: ApplicationCloud Source code

```
void AppsCloud::handleMessage(cMessage *msg){
    if (msg->getArrivalGate()->getIndex()<5){ // inbound messages
        int gSize = gateSize("g");
        //randoms the directory service that receives the request
        int rr = 5 + intrand(gSize-5);
        send(msg, "g$o", rr); //index 2 first dirService module index
    }
    else if (msg->getArrivalGate()->getIndex()>=5){ // outbound messages
        if(strcmp(msg->getName(),"MobileApp")==0){
            send(msg, "g$o", 4);
        }
        if(strcmp(msg->getName(),"DisplApp")==0){
            AppMsg *dA = check_and_cast<AppMsg *>(msg);
            if(dA->hasPar("contextInfo")){
                if  ((int)dA->par("contextInfo") == 0)
                    //relays msg to the environment service
                    send(msg, "g$o", 3);
                if((int)dA->par("contextInfo") == 1 || (int)dA->par("contextInfo") == 2)
                    //relays msg to the display node
                    send(msg, "g$o", 0);
            }
        }
    }
}
```

As the previous source code example shows, the interconnections between components are implemented and the system works together to provide the proper response to each request, independently of its source or destination. It can be seen on the chosen gate to forward the messages, *send(msg, "g$o", 4);*, the number 4 says that the message should be sent to the mobile application module.

For each component on the network segment the implementation has been mainly guided by the best way to replicate a web server. With this, we have then implemented all the associated logic of message handling with the notion of processing times, number of simultaneous connections and number of processing cores as a real web server should work. As we are implementing an application layer server with loosely coupled connections, with each message not having a state to maintain, it seems a good starting approach to those modules. Of course, the reality is not this simple, however if we intend to append any logic to improve or mimic any future middleware decisions, those should always be the foundations for those components.

The following code represents our simple implementation of the queueing system implemented on the servers. When a message arrives the serves checks if it has space left on the queue to process the request, if not, the message is sent back with the *rejected* parameter. Having the opportunity to be processed the server computes its position on the queue and reschedules the message with a delay that should address the response time for that message. It computes the sending delay by multiplying its position on the queue by the average processing time for each request on the queue adding it later to the current simulation time *simTime() + (processingTime*(requests/nCores))*. The formula also contemplates the number of cores that, despite not being so linear, each new core should reduce the size of the queue by providing a much more faster response for each request.

Listing 5.3: Server Source code

```
void EnvironmentService::handleMessage(cMessage *msg){
    if(msg->isSelfMessage()){
        requests--;
        send(msg,"g$o");
    }
    else {
        if(requests < maxLoad*nCores){
            requests++;
            //receives checkIn from a mobile application
            if(strcmp(.....)==0){
            ....
                if(requests%nCores==0)
                    scheduleAt(simTime() +
                      (par("processingTime").doubleValue()*(requests/nCores)), cI);
                else scheduleAt(simTime() +
                      (par("processingTime").doubleValue()*((requests/nCores)+1)), cI);
            }
        }
        else {
            msg->addPar("rejected");
            send(msg, "g$o");
        }  }
}
```

The rejected messages are returned to its sender that assumes that the processing time it is *-1*, as shown in Figure 5.7. This value emerges because in our simulation we do not have any implementation of a recovery strategy. Those recordings assume a great importance because each failed response

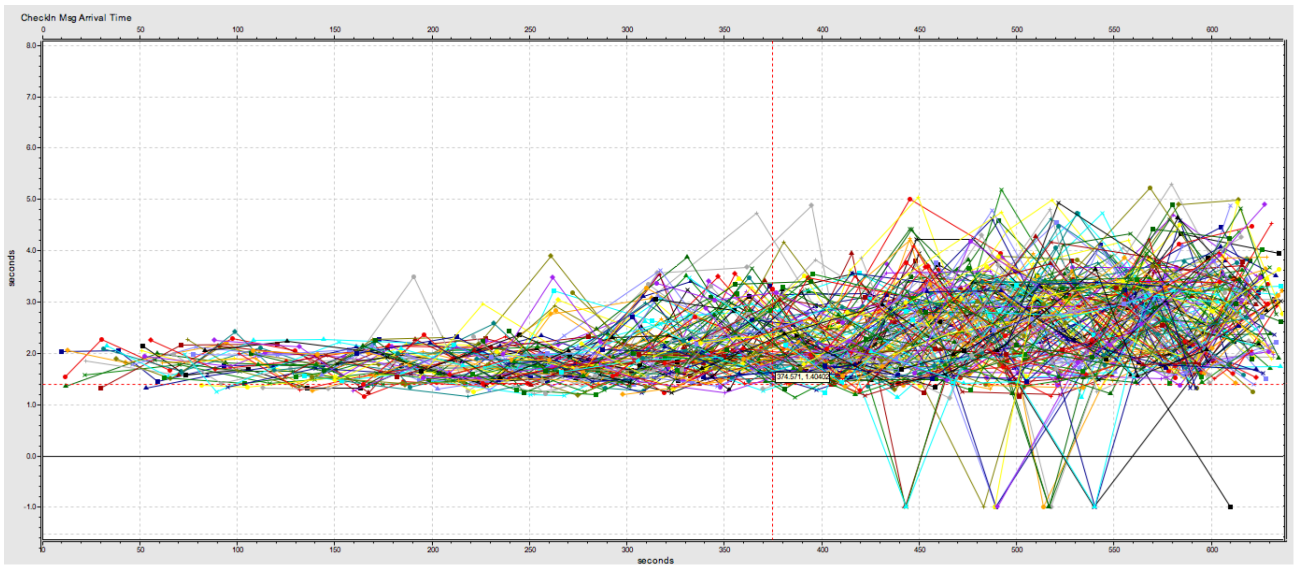


Figure 5.7: Fictitious *Check in* interaction arrival times.

message potentially damages the user experience, once it will inevitably increase the system response time and with this representation we can clearly see when the system starts to fail.

Where the application scenario has its real representation is on the edges of the system, more precisely, on the modules that generates user related inputs to the system: the sensors, the display nodes and the mobile applications. Naturally the major efforts were made in these modules. For each one we have tried to code the behavior that best resembles the reality, having parameters to configure and easily change those behaviors. The simulated application scenario has a logic that was transformed accordingly the programming methods of the chosen framework. The following lines of code demonstrates exactly how it works. Initially, we had to create a *initialize();* function that will be called just once during the lifetime of the simulation. This function creates and schedules a message to itself that will be lately handled by the *handleMessage(cMessage *msg);* function. That *handleMessage* function will be ready to receive and to treat the incoming messages, in this case only from itself, during the simulation. With this code block we have a simple implementation of a physical sensor, that has a parameter were we can define, with some value, the sensing frequency, sending contextual information.

Listing 5.4: Physical sensor source code example

```
void PhysicalSensor::initialize(){
    AppMsg* sU = new AppMsg("Sensing");
    sU->setAppEvent("some contextual information!!");
    scheduleAt(simTime() + par("senseFreq"), sU );
}

void PhysicalSensor::handleMessage(cMessage *msg){
    if (msg->isSelfMessage()) {
        send(msg, "g$o");

        AppMsg* sU = new AppMsg("Sensing");
        scheduleAt(simTime() + par("senseFreq"), sU );
    }
}
```

The components that generate input messages to the system work as the previous code example, with the notion of self messaging to schedule events. Of course, modules that have a more complex or dynamic behavior will have a greater level of difficulty to accomplish all the steps required by the application scenario. In contrast to mobile applications and display node modules, the physical sensors only have a one-way flow of information. This means that they only feed the system with the context information, not receiving any feedback. The other two module types, in almost every action they perform, they expect some response that should be properly treated by the handle message function.

The messages that are being transmitted throughout the simulation have a simple definition on the framework. The fields that compound the messages are very simple, with fields to store the gate numbers of the clouds to know the exact way back path of each and the required fields to store any information, for example some application event data.

Listing 5.5: Message Definition example

```
packet AppMsg {
    // path
    int mobileGate;
    int mobileAppCloudGate;

    int displayNumber;
    int displayClusterNumber;

    string appEvent;
}
```

After the message defined, the framework assumes the creation of two files (*.cc* and *.h*) that should contain auxiliary functions as the one responsible to change data on each field, *getters* and *setters*, functions to add new parameters or even functions to get access to message creation times.

5.4 Parameters Input values

Having all the implementation set up with the code prepared to be initialized with the correct parameters, is time to chose the right ones to have a tuned simulation. As we have seen, there are two major sets of module components so consequently two sets of parameters. Those that make part of the network segment: the clouds and the core components; and those in the screen segment, physical sensors, mobile applications and display nodes.

We have prepared two simulations to show that our simulation testbed really reflects the usage of a pervasive displays network with different models. To achieve that, we have made a simulation where we have much more mobile application users interacting with the system, and less display nodes users with direct display interactions and other with more displays than mobile users interacting with the system. As these two forms of interaction have different interaction patterns we expect to see significant differences on the system behavior. For both running simulations, we pretend to simulate a simple system installation were the underlying architecture is "near" the users, on the same country region. This means that there are no users in some distant locations therefore with low connection latencies.

Thus we define the number of clusters for the mobile users (*numMMA*) and display nodes (*numDN*) as one, with the average connection latency times recorded to Europe. We have defined a bigger latency to the mobile applications clusters because of the probable access network, via 3G or Wi-Fi. These two parameters have a normal distribution truncated to nonnegative values with the shown average and standard deviation (*truncnormal(mean, stddev, rng=0)*). These can be seen on the listing above.

Listing 5.6: First parameters

```
##number of clusters
FullArchitecture.numMMA = 1
FullArchitecture.numDN = 1

###connection to clusters delay
FullArchitecture.mobileDelay = truncnormal(0.017s,0.003s)
FullArchitecture.displaysDelay = truncnormal(0.014s,0.005s)
```

After this we had to set the parameters that directly represent the expected human behavior and the created application scenario. Those parameters, unfortunately are not based in any real measurements, however the chosen values seems quite legit to us. For now the shown parameters are for the first simulation, where there are more mobile application users than display node users.

We have defined some parameters to represent that behavior: a)The number of users that should "live" on each cluster (*numMobApp*). b)The starting time for their interactions on the simulation (*startingTime*), to ensure that users not start their activity at the same time. c)The time between each new display (*time2Display*), as seen on the definition of the application scenario (Chapter 4) a mobile user (a kid going to school) passes through multiple displays until cease his activity and this parameter tries to replicate it. Then, there exists the parameter to define the interaction reaction time of a mobile user after it receive some system response on his mobile equipment (*sendInterval*).

Before defining the behavior of the display nodes cluster we also have to define of how many displays should exist on that cluster (*numDispNum*) and the number of associated physical sensor (*numSensors*). Display nodes, in our scenario, are capable to generate, not only receive, information. We embodied those displays as large city touch screens. In this way, users can directly interact with it, in some starting time (*firstInteraction*) and with some frequency after the system feedback (*interactionInterval*). One of the parameters is also the frequency of the sensed information by the physical sensors (*senseFreq*).

The previous parameters do not have a fixed static value during the simulation, we have defined values in a normal or uniform distribution hopping to better mimic the reality.

The last two parameters are related with the *Application* module and they represent the frequency that an application should request for some contextual information (*envServMsgFreq*), to provide more accurate results, or requests the system to spread a global message trough other displays (*globalMsgFreq*). One more time we do not have studies about the correct input values to these parameters.

Listing 5.7: Behavior Parameters

```
## Mobile users Cluster
FullArchitecture.mobileCluster[*].numMobApp = 2000
FullArchitecture.mobileCluster[*].mobApp[*].startingTime = uniform(0.5s, 30s)
FullArchitecture.mobileCluster[*].mobApp[*].time2Display = truncnormal(200s, 40s)
FullArchitecture.mobileCluster[*].mobApp[*].sendInterval = truncnormal(10s, 2s)

## Display nodes Cluster
FullArchitecture.displayCluster[*].numDispN = 100
FullArchitecture.displayCluster[*].numSensors = 50
FullArchitecture.displayCluster[*].sensor[*].senseFreq = truncnormal(40s,5s)

FullArchitecture.displayCluster[*].displayNode[*].firstInteraction = uniform(5s,25s)
FullArchitecture.displayCluster[*].displayNode[*].interactionInterval=truncnormal(15s,3s)

FullArchitecture.pdnet.apps.pdnetApps[*].envServMsgFreq = uniform(20,70)
FullArchitecture.pdnet.apps.pdnetApps[*].globalMsgFreq = uniform(70,200)
```

Finally we had to concern about the network segment components parameters. For each system component we could define the number of *cores* that represents the number of simultaneous requests that can be processed, the *queueSize* that represents the server *backlog* and the *processingTime* for each request. The first two parameters have static values that are unchangeable during the simulation but the processing time could be variable.

Those components, for now, should have a similar configuration, the only changing parameter is the processing time that will be assigned with values that we assume to be reasonable. We do not have any measurements about the average processing times of these components. The below listing shows the parameters and values used for the *Environment Service* as example. It also shows the used values for the processing time of each message for each component.

Listing 5.8: Components Input values

```
## Environment service
FullArchitecture.pdnet.envServ.numEnvS = 1
FullArchitecture.pdnet.envServ.pdnetEnvServ[*].cores = 1
FullArchitecture.pdnet.envServ.pdnetEnvServ[*].queueSize = 512
## Previous parameters are the same to all components

##Application processing time
FullArchitecture.pdnet.apps.pdnetApps[*].processingTime = truncnormal(0.06s, 0.015s)

##Directory service processing time
FullArchitecture.pdnet.dirServ.pdnetDirServ[*].processingTime = truncnormal(0.04s, 0.01s)

##Display service processing time
FullArchitecture.pdnet.dispServ.dispServ[*].processingTime = truncnormal(0.05s, 0.01s)

##Environment service processing time
FullArchitecture.pdnet.envServ.pdnetEnvServ[*].processingTime = truncnormal(0.07s, 0.015s)
```

To finish the explanation of the simulation parameters, we just have to say how the simulation is defined to finish. A parameter was created (*maxMsgToEndSim*) on the mobile cluster that basically says that the simulation ends when the first mobile application reaches that specified value. In this way, it is possible that, at the end of the simulation, the mobile applications have not sent the same number of messages.

Listing 5.9: Simulation "end" parameter

```
FullArchitecture.mobileCluster[*].mobApp[*].maxMsgToEndSim = 1000
```

The input values, particularly those associated with the application usage of the system, are maximized to test the system response capacity without any data loss. What we intend is to discover when some component reaches the point of having to reject messages. As we do not implement any fault tolerance algorithms, that could recover the information state in a time that do not compromise the user interaction, this seems to be a good method to assess where the system struggles. Another important aspect lies on the fact that the running simulations do not have a day time representation. Meaning that the we do not simulate any lull period like those expected to occur by night. Instead we try to depict system usage on heavy usage times. On the second simulation these principles remain the same. However, the changes on the number of elements that generate input to the system should give us, as we have already said, very different results. That is because the methods and the frequencies of those inputs are significantly different.

The complete set of input values for the second simulation can be seen on Appendix A.

Chapter 6

Results and Analysis

On any simulation what really worths is the final outcome, the data that derives from all the implemented integrating parts. Those simulation results should be the main ingredients to help us to create and refine systems, letting us to save time and money on deployments.

In this chapter, it will be presented the collected data generated through the input parameters, analyzing the output graphics and presenting some characteristics that arise from the simulation models. Mainly, this section shows that our implementation could be a platform to study different design patterns between components, contributing to inform the design of solutions for pervasive displays networks.

6.1 Measures and Results

The presented results are the outcome of the running simulation with the parameters described and explained on previous chapter. We have seen that the parameters were adjusted to lead the system to a constant stress situation under two different situations. The first where mobile application users number is far greater than display node users, in contrast with the second situation where the number of display node users is higher than the mobile application users.

Following the previous chapter considerations, about the creation of the foundations to achieve a fully functional simulation environment for our pervasive architecture, it is imperative to decide what should be the items to measure. Possibilities are almost endless, however, as the focus of almost pervasive architectures and networks is Human centric, efforts will be, mostly, concerning the response times of the interactions performed by the users. Those interactions could have a explicit nature, through personal mobile phones or interactive displays, as also a implicit nature being triggered by the sensors around the displays.

We will record the time that takes a request message since its creation until the arrival to the correct receiver.

Another parameter that is being recorded is the number of lost messages on the system. Those messages appear as result of some server having achieved its maximum load. Those loads are also one of the recorded values during the lifetime of the simulation. With this we hope to understand where the system struggles to provide us the expected feedback.

What we attend to achieve with the simulation, in this first stage, is to grasp how the interactions and subsequent interdependencies between the components that make part of the pervasive system influence its performance. We intend to show that the design patterns and the unbalances on the number of different types of users lead to significant changes on the system behavior.

We hope that our analysis may provide important guidelines for those who will have the responsibility of managing the system, either in terms of hardware dispersion, system middleware design and even for those who develop the applications.

To do that, we have to initialize the defined parameters with values that should depict those changes. The running simulations will then provide the results to conduct the following decisions and improvements on the system.

6.2 Analysis

On the following graphics results are aggregated by segment and by cluster.

On a scenario where we have more display nodes with direct interaction inputs, and consequently with more physical sensors around them, what we can clearly confirm is that over time those sensors will overflow the *Environment Service* with new contextual information. This scenario counts, as

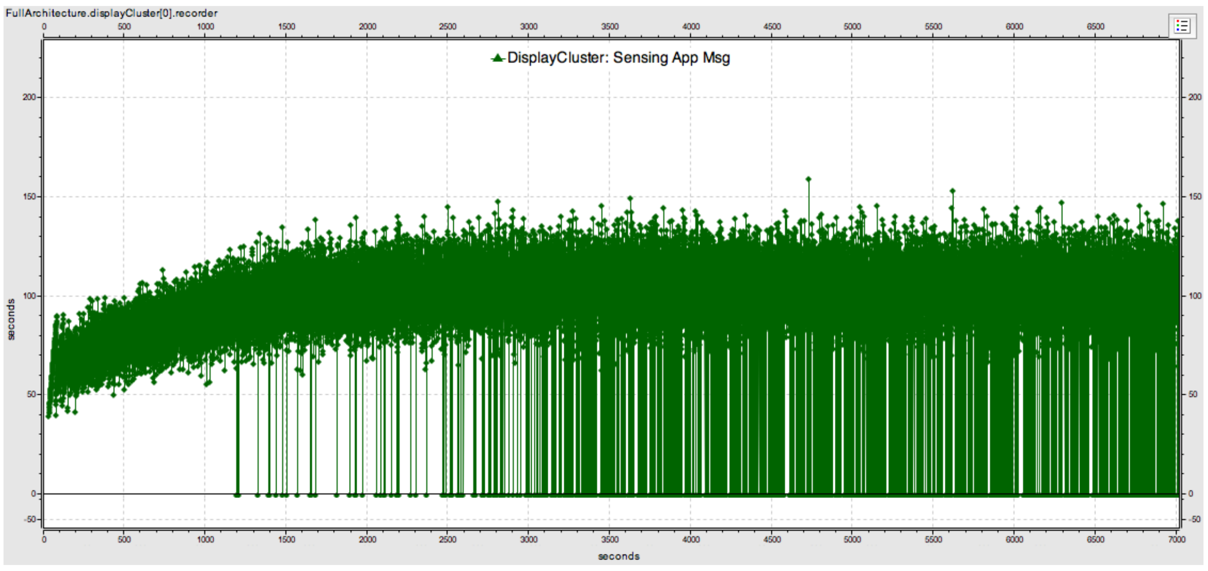


Figure 6.1: Physical Sensor message times.

we have defined on the parameters, with 100 mobile application users, 400 display nodes with direct interactions and 200 physical sensors. The sensing messages, as seen on Figure 6.1, start to take increasingly more time to be processed until the server can not handle all of them, starting to reject some of them. One of consequences of that overflow is that, as we can observe on Figure 6.2, *Check In* made by the mobile application users next to the same *Environment Service* starts to fail. The failure on this interaction compromises the fluidity of the mobile application, and ultimately terminate it because if a user cannot bootstrap the system, he also cannot interact with it.

With this configuration the server loads are, as expected, high on the the

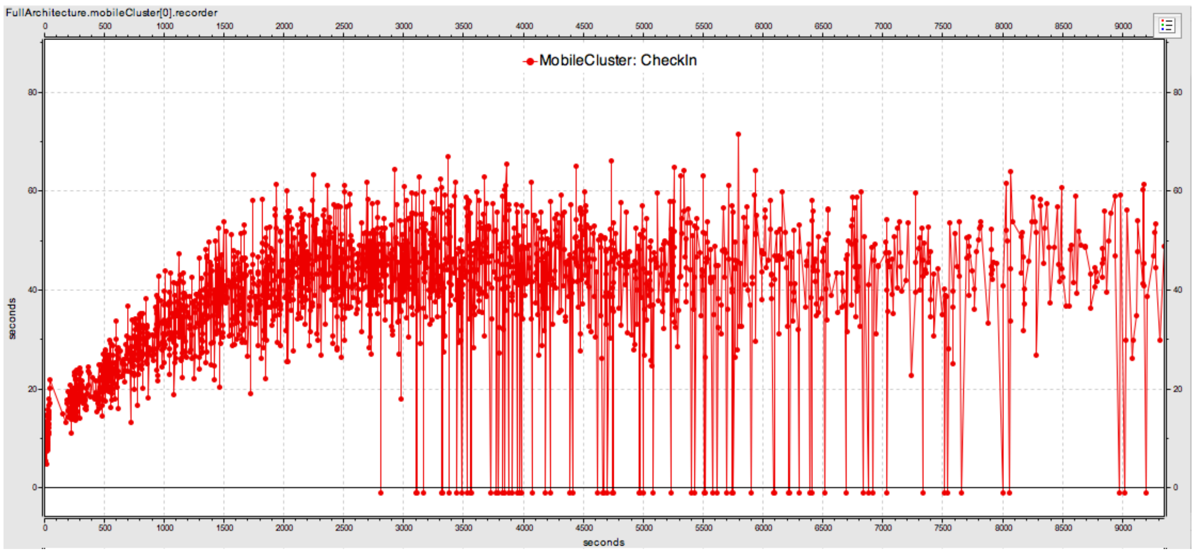


Figure 6.2: Check In message times.

Application server and on the *Environment Service* server. The other two main servers, the *Display Service* and the *Directory Service* have residual loads. That is due to the fact that on our configuration those two servers are only used, respectively, to compute and spread global messages to the display nodes and to provide answer to the *LookUp* requests made by the mobile applications. That can be seen on Figure 6.3.

We can see that *Application* response times to the direct display interactions and mobile application interactions combined lead the server to almost a maximum load. However only a few messages are rejected during the simulation time on the mobile cluster, Figure 6.4 and Figure 6.5 in blue.

Figure 6.5 also shows us the times of the *GlobalMessage* requests in red.

The other scenario, where the number of display nodes and physical sensors are diminished, can embrace several thousands of mobile users with their mobile applications requests. This happens by several factors, one of them is the end of overload by physical sensors on the environments service servers with update requests. Another one is that despite the significant number of mobile application users, the requests are sparse enough to not compromise the systems usage. The application server handles the requests without

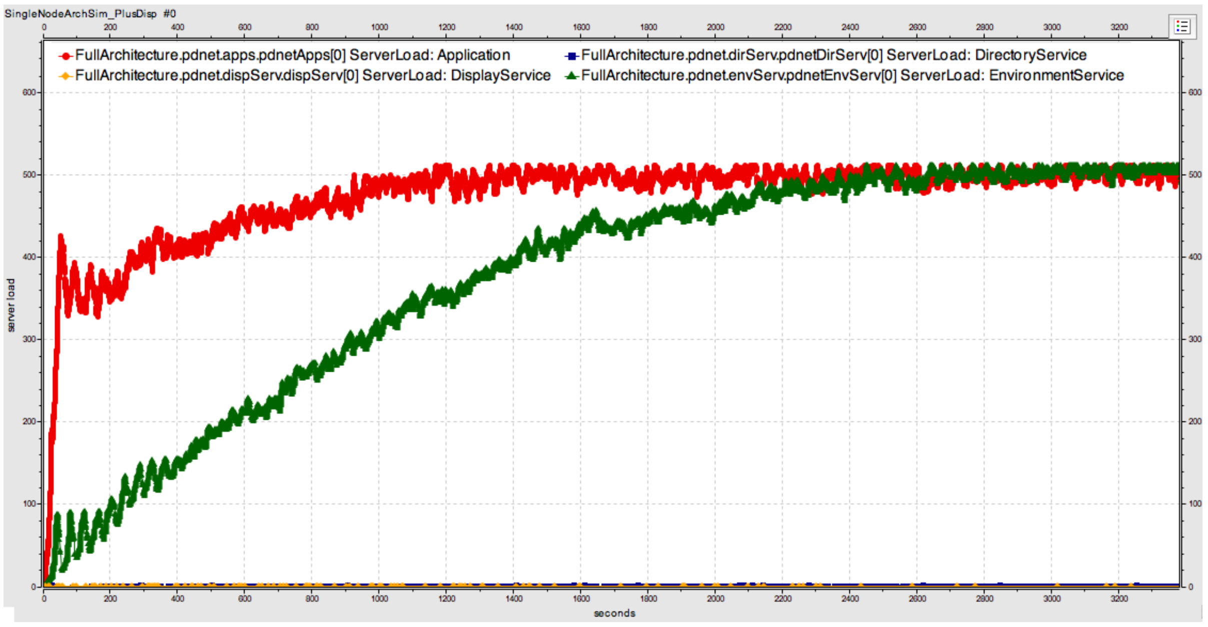


Figure 6.3: Server loads.

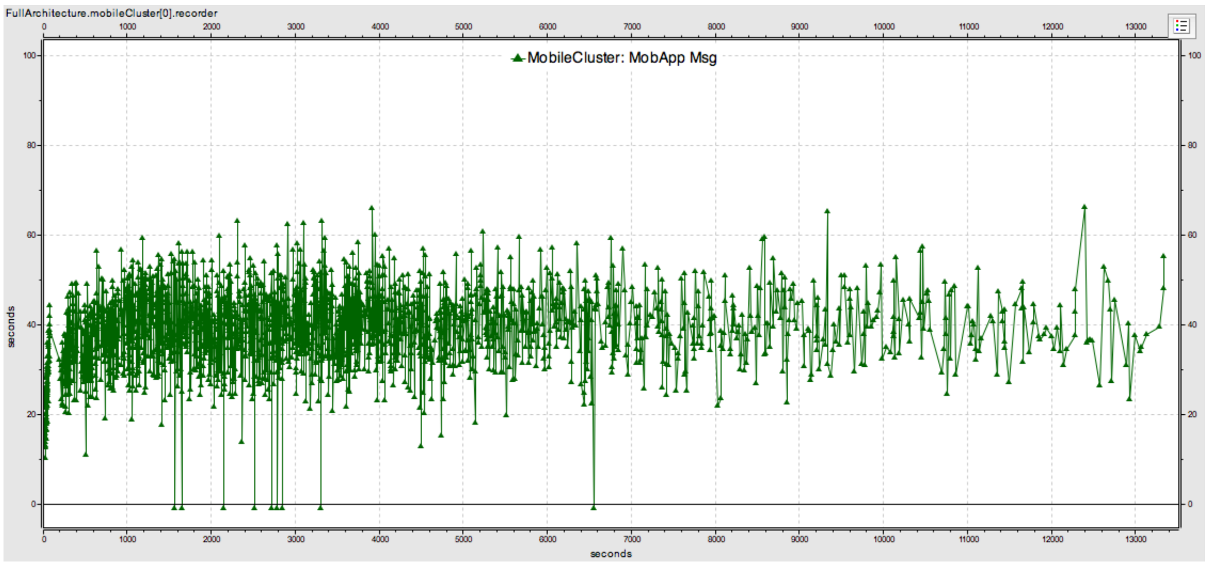


Figure 6.4: Mobile applications application times.

high loads. The other components that compose our pervasive system also handles with the loads that those interactions put on them.

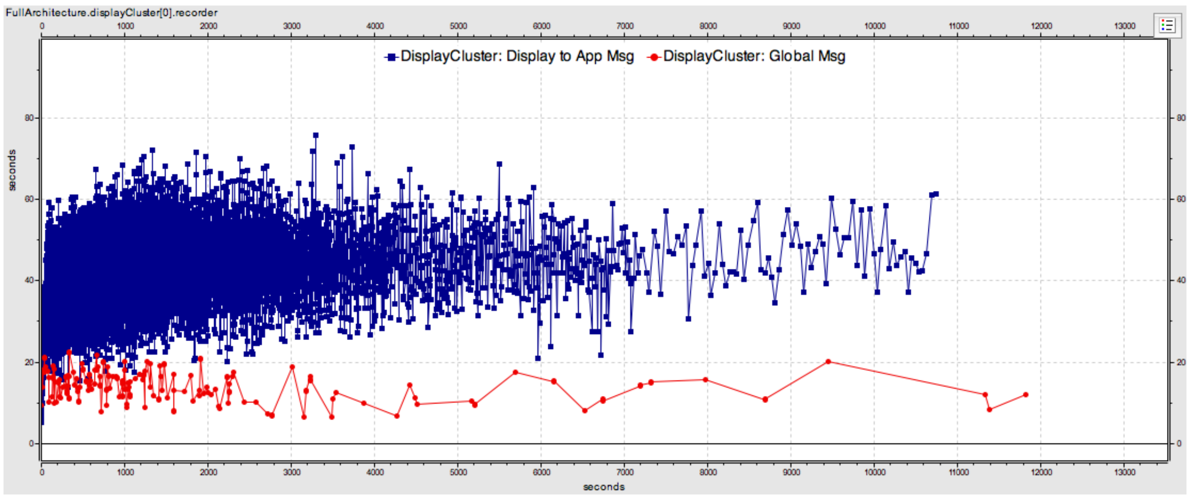


Figure 6.5: Display nodes application message and global message times.

The resulting graphics shows that the system can handle very well 2000 mobile applications, 100 display nodes interacting and 50 physical sensors with, barely, no effort.

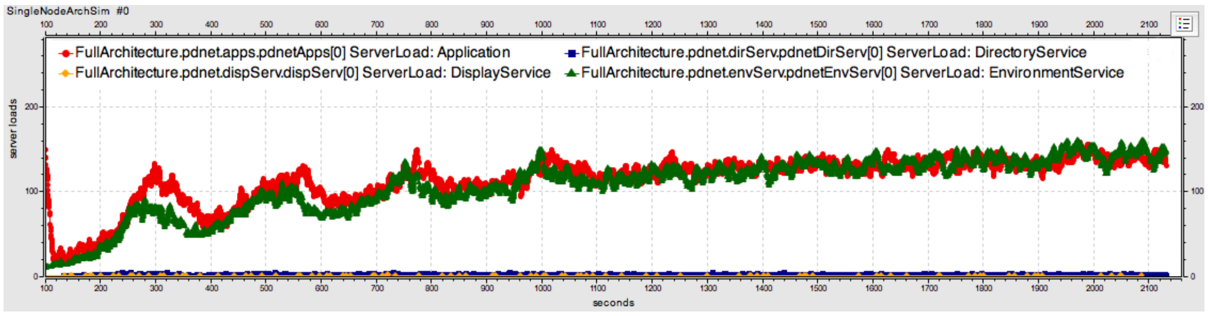


Figure 6.6: Server Loads.

Once more in Figure 6.6, showing the server loads, the components that have more load are the *Application* server and the *Environment Service* server. The other two, *Display Service* and *Directory Service* servers, have insignificant loads on this scenario and configuration.

The arrival times of the *LookUp*, *Check In* and *MobileApplication* messages are shown on Figure 6.7. As expected, the message times have a clear match between their arrival time and the server loads.

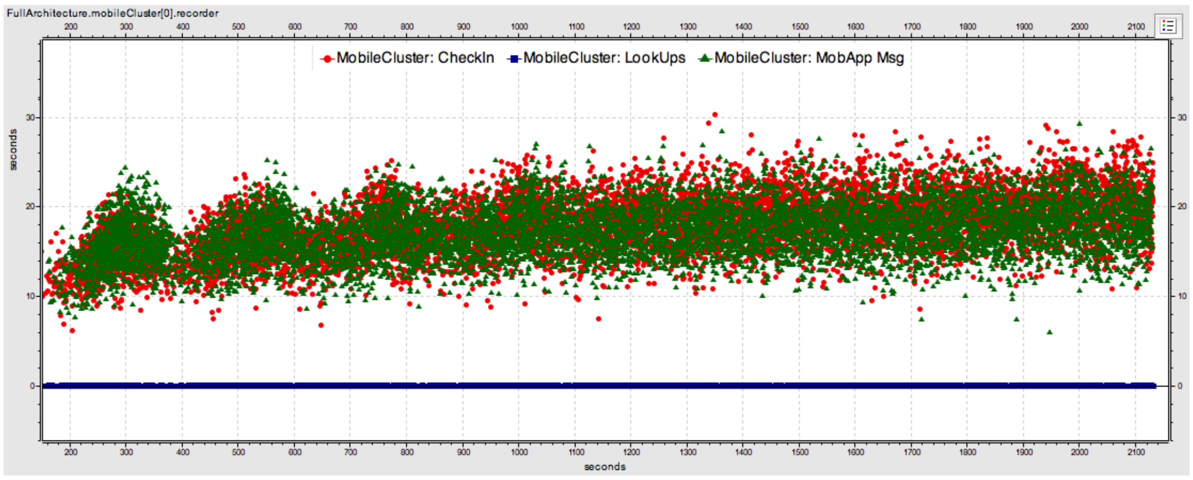


Figure 6.7: Mobile Cluster aggregated times.

The *LookUp* message is treated almost instantaneously due to the almost inexistent server load while the other two types of message need more time to arrive their destinies.

The following graphic, seen on Figure 6.8, shows the times for the display node cluster interactions. It presents for the *Sensing*, *DisplayApplication* and *GlobalMessage* the times that takes to arrive to the cluster since their creation.

The presented graphics and analysis are merely demonstrative to show that our simulation testbed really represents the pervasive displays networks interactions. They do not depict any real scenario, in terms of input parameters values, but they made a clear representation that different interaction flows can lead the system to different stress situations. Our tool, may be and should be instantiated and parameterized according the needs and the values that is pretended to study. With it we hope to help and stimulate a correct growth of pervasive display systems.

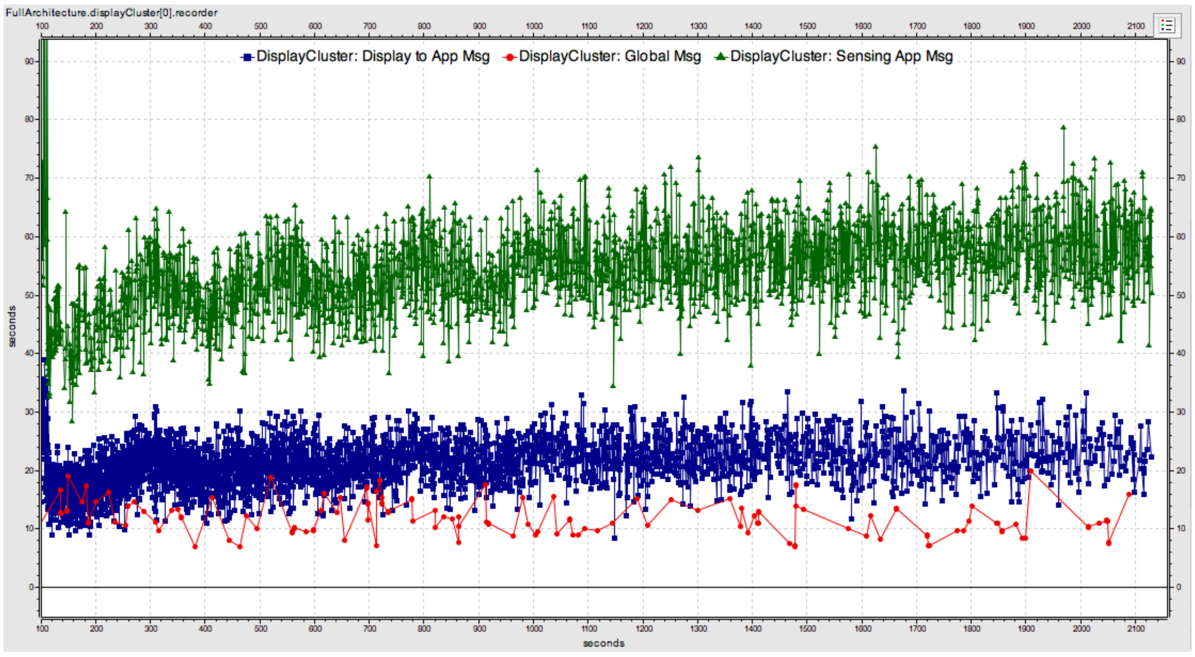


Figure 6.8: Display Node cluster aggregated times.

Chapter 7

Conclusions

With the growth of this emergent research area, pervasive displays networks, it is mandatory to settle down some concepts and start to introduce methodology in the community. It is important to congregate efforts with common objectives. Nowadays, from what we have seen, the research teams do not give much attention to the existing implementations and start creating from zero their own visions. None of the existing systems relies on some previous implemented solutions, and they do not drive an iterative process learning and improving themselves with the previous work spending too much time and money in single deployments, not focusing on the expansion and improving of the previous concepts and implementations.

Besides these facts, it is relevant to point out that those deployments and test systems usually do not care about the growth of the network, and this is a key aspect to create a real public pervasive display network. This work intended to explore the scientific challenges of the underlying network, providing a tool to study the scalability problems and how should the core components be spread throughout this global network, defining the Pd-Net architecture and provide a basic layout for this project. It should also provide a quick-start study for those who want to focus their attention to other aspects rather than be worried with the topologies and technologies associated.

Although this study do not present any suggestion about the design patterns that should be implemented throughout the system, it tries to demonstrate that different usage scenarios and decisions on the implementation have significant changes on the system performance. Mainly, the outcome of our work is the creation of a tool to simulate pervasive displays networks with their very specific requirements. We have also made an attempt to clarify some of the aspects regarding the components and the difficulties that may arise from these pervasive networks. Framing and characterizing some of the processes that inevitably compose the system.

In particular, we want to give a major importance to the creation and specification of the simulation scenario, without it would never be possible to assess our implementation and extract any results from it. Thinking on the scenario inherent interactions also made a contribution to define and categorize some processes and requirements of the pervasive displays networks.

Regarding our simulations, the obtained results are just to prove that our implementation really reflects different configurations of the network, showing that those, even slightly changes could create dramatic changes on its behavior and performance.

However, this is a non stop work that should evolve and grow according to the needs and the innovations behind this research area. Many disruptive ideas should arise from the creative heads that are in this multidisciplinary area. So, it is fundamental to follow those ideas supporting them with the underlying technology, trying to embody some of the most disruptive visions of public displays, walking through the most challenging requirements.

7.1 Future Work

The future of pervasive displays networks is being constructed step by step, a lot of work is still to be made and many problems are waiting to be solved. We hope that these networks became a reality in a near future and we would be thrilled if our work provide a contribute for a brighter future for them.

In the future, to improve our study and our simulation tool, we intend to see several points enhanced:

- To obtain a well coordinated effort between middleware development teams
- To have a clearer idea about the interactions between the system core components
- To implement more accurately the interactions performed by new pervasive applications on different scenarios
- To create more modules to test scalability techniques on each segment of the network
- To embed on the same running simulation different applications with their interactions
- To improve the input values that are directly related to Human behaviors against public displays
- To have accurate values of the processing times for each component
- To improve the accuracy of the queues used on the servers
- To implement different design patterns on the modules

Appendix A

Appendix

Listing A.1: Parameters Input values

```
## Environment service
FullArchitecture.pdnet.envServ.numEnvS = 1
FullArchitecture.pdnet.envServ.pdnetEnvServ[*].cores = 1
FullArchitecture.pdnet.envServ.pdnetEnvServ[*].queueSize = 512
## Previous parameters are the same to all components

##Application processing time
FullArchitecture.pdnet.apps.pdnetApps[*].processingTime = truncnormal(0.06s, 0.015s)

##Directory service processing time
FullArchitecture.pdnet.dirServ.pdnetDirServ[*].processingTime = truncnormal(0.04s, 0.01s)

##Display service processing time
FullArchitecture.pdnet.dispServ.dispServ[*].processingTime = truncnormal(0.05s, 0.01s)

##Environment service processing time
FullArchitecture.pdnet.envServ.pdnetEnvServ[*].processingTime = truncnormal(0.07s, 0.015s)

##Application Server
FullArchitecture.pdnet.apps.pdnetApps[*].envServMsgFreq = uniform(20,70)
FullArchitecture.pdnet.apps.pdnetApps[*].globalMsgFreq = uniform(70,200)

## Directory Service Server
FullArchitecture.pdnet.dirServ.pdnetDirServ[*].firstReq = uniform(1s, 60s)
FullArchitecture.pdnet.dirServ.pdnetDirServ[*].sendInterval = truncnormal(60s, 5s)

### cluster number
FullArchitecture.numMMA = 1
FullArchitecture.numDN = 1

###connection to clusters delay
FullArchitecture.mobileDelay = truncnormal(0.017s,0.003s)
FullArchitecture.displaysDelay = truncnormal(0.014s,0.005s)

########################
## Mobile Users Cluster

FullArchitecture.mobileCluster[*].numMobApp = 100
FullArchitecture.mobileCluster[*].mobApp[*].sendInterval = truncnormal(10s, 2s)
#FullArchitecture.mobileCluster[*].mobApp[*].startingTime = 0.2s
FullArchitecture.mobileCluster[*].mobApp[*].startingTime = uniform(0.5s, 30s)
FullArchitecture.mobileCluster[*].mobApp[*].time2Display = truncnormal(200s, 40s)
```

```
##########################
## Display Node Cluster

FullArchitecture.pdnet.apps.pdnetApps[*].maxExistingDisplays = 400
#all existing displays on all clusters
# FullArchitecture.displayCluster[*].numDispN * FullArchitecture.numDN

FullArchitecture.displayCluster[*].numDispN = 400
FullArchitecture.displayCluster[*].numSensors = 200
FullArchitecture.displayCluster[*].sensor[*].senseFreq = truncnormal(40s,5s)
FullArchitecture.displayCluster[*].displayNode[*].firstInteraction = uniform(5s,25s)
FullArchitecture.displayCluster[*].displayNode[*].interactionInterval = truncnormal(15s, 3s)


FullArchitecture.mobileCluster[*].mobApp[*].maxMsgToEndSim = 100
```

Bibliography

[1] Florian Alt and Stefan Schneegass. A conceptual architecture for pervasive advertising in public display networks. In *Proceedings of the 3rd Workshop on Infrastructure and Design Challenges of Coupled Display Visual Interfaces*, PPD'12, 2012.

[2] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala. Improving simulation for network research. Technical Report 99-702b, University of Southern California, March 1999. revised September 1999, to appear in IEEE Computer.

[3] A. Chandler, J. Finney, C. Lewis, and A. Dix. Toward emergent technology for blended public displays. In *Proceedings of the 11th international conference on Ubiquitous computing*, pages 101–104. ACM, 2009.

[4] E.F. Churchill, L. Nelson, L. Denoue, and A. Girgensohn. The plasma poster network: Posting multimedia content in public places. In *Proceedings of INTERACT*, volume 3, pages 599–606, 2003.

[5] S. Clinch, J. Harkes, A. Friday, N. Davies, and M. Satyanarayanan. How close is close enough? understanding the role of cloudlets in supporting display appropriation by mobile users. In *Pervasive Computing and Communications (PerCom), 2012 IEEE International Conference on*, pages 122 –127, march 2012.

[6] Sarah Clinch, Nigel Davies, Thomas Kubitza, and Albrecht Schmidt. Designing application stores for public display networks. In *Proceedings of the 2012 International Symposium on Pervasive Displays*, PerDis '12, pages 10:1–10:6, New York, NY, USA, 2012. ACM.

[7] J.H. Cowie, D.M. Nicol, and A.T. Ogielski. Modeling the global internet. *Computing in Science Engineering*, 1(1):42 –50, jan/feb 1999.

[8] Nigel Davies, Marc Langheinrich, Rui Jose, and Albrecht Schmidt. Open display networks: A communications medium for the 21st century. *Computer*, 45:58–64, 2012.

[9] M. Dayarathna, A. Withana, and K. Sugiura. Infoshare: Design and Implementation of Scalable Multimedia Signage Architecture for Wireless Ubiquitous Environments. *Wireless Personal Communications*, pages 1–25.

[10] J. De Vriendt, P. Laine, C. Lerouge, and Xiaofeng Xu. Mobile network evolution: a revolution on the move. *Communications Magazine, IEEE*, 40(4):104 –111, apr 2002.

[11] Tommi Heikkinen, Tomas Linden, Timo Ojala, Hannu Kukka, Marko Jurmu, and Simo Hosio. Lessons Learned from the Deployment and Maintenance of UBI-Hotspots. In *2010 4th International Conference on Multimedia and Ubiquitous Engineering*, pages 1–6. IEEE, August 2010.

[12] Thomas R. Henderson, Sumit Roy, Sally Floyd, and George F. Riley. ns-3 project goals. In *Proceeding from the 2006 workshop on ns-2: the IP network simulator*, WNS2 '06, New York, NY, USA, 2006. ACM.

[13] S. Hosio, H. Kukka, M. Jurmu, T. Ojala, and J. Riekki. Enhancing interactive public displays with social networking services. In *Proceedings of the 9th International Conference on Mobile and Ubiquitous Multimedia*, page 23. ACM, 2010.

[14] R. José, N. Otero, S. Izadi, and R. Harper. Instant places: Using bluetooth for situated interaction in public displays. *Pervasive Computing, IEEE*, 7(4):52–57, 2008.

[15] C. Kray, G. Kortuem, and A. Krüger. Adaptive navigation support with public displays. In *Proceedings of the 10th international conference on Intelligent user interfaces*, pages 326–328. ACM, 2005.

[16] J. Müller, J. Exeler, M. Buzeck, and A. Krüger. Reflectivesigns: Digital signs that adapt to audience attention. *Pervasive Computing*, pages 17–24, 2009.

[17] J Müller and M Jentsch. Exploring factors that influence the combined use of mobile devices and public displays for pedestrian navigation. *... of the 5th Nordic conference on ...*, 2008.

[18] J. Müller, O. Paczkowski, and A. Krüger. Situated public news and reminder displays. *Ambient intelligence*, pages 248–265, 2007.

[19] J. Müller, D. Wilmsmann, J. Exeler, M. Buzeck, A. Schmidt, T. Jay, and A. Krüger. Display blindness: The effect of expectations on attention towards digital signage. *Pervasive Computing*, pages 1–8, 2009.

[20] T Ojala and H Kukka. UBI-hotspot 1.0: Large-scale long-term deployment of interactive public displays in a city center. *... and Services (ICIW ...*, 2010.

[21] T. Ojala, V. Valkama, H. Kukka, T. Heikkinen, T. Lindén, M. Jurmu, F. Kruger, and S. Hosio. UBI-hotspots: sustainable ecosystem infrastructure for real world urban computing research and business. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, pages 196–202. ACM, 2010.

[22] M. Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[23] Ahmed Sobeih, Wei-Peng Chen, Jennifer C. Hou, Lu-Chuan Kung, Ning Li, Hyuk Lim, Hung-Ying Tyan, and Honghai Zhang. J-sim: A simulation environment for wireless sensor networks. In *Proceedings of the 38th annual Symposium on Simulation*, ANSS '05, pages 175–187, Washington, DC, USA, 2005. IEEE Computer Society.

[24] O. Storz, A. Friday, and N. Davies. Supporting content scheduling on situated public displays. *Computers & Graphics*, 30(5):681–691, 2006.

[25] O. Storz, A. Friday, N. Davies, J. Finney, C. Sas, and J.G. Sheridan. Public ubiquitous computing systems: Lessons from the e-campus display deployments. *Pervasive Computing, IEEE*, 5(3):40–47, 2006.

[26] Kazunori Sugiura, Miyuru Dayarathna, and Anusha Withana. Design and implementation of distributed and scalable multimedia signage system. In *2010 Second International Conference on Ubiquitous and Future Networks (ICUFN)*, pages 273–278. IEEE, June 2010.

[27] S. Tarkoma, J. Kangasharju, T. Lindholm, and K. Raatikainen. Fuego: Experiences with mobile data communication and synchronization. In *Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on*, pages 1–5. IEEE, 2006.

[28] Ville Valkama and Timo Ojala. Stakeholder value propositions on open community testbed of interactive public displays. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, MEDES '11, pages 107–113, New York, NY, USA, 2011. ACM.

[29] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 60:1–60:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[30] E. Weingartner, H. vom Lehn, and K. Wehrle. A performance comparison of recent network simulators. In *Communications, 2009. ICC '09. IEEE International Conference on*, pages 1 –5, june 2009.

[31] M. Weiser. The computer for the 21st century. *Scientific American*, 265(3):94–104, 1991.

[32] X. Zeng, R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *Parallel and Distributed Simulation, 1998. PADS 98. Proceedings. Twelfth Workshop on*, pages 154 –161, may 1998.