Smartphone exploitation in Ambient Intelligence Environments

Ftylitakis Nikolaos

Thesis submitted in partial fulfillment of the requirements for the

Masters' of Science degree in Computer Science

University of Crete
School of Sciences and Engineering
Computer Science Department

Thesis Advisor: Prof. *Constantine Stephanidis*

This work has been supported by the Institute of Computer Science of the Foundation for Research and Technology-Hellas (FORTH).

UNIVERSITY OF CRETE

COMPUTER SCIENCE DEPARTMENT

Smartphone exploitation in Ambient Intelligence

Thesis submitted by

FTYLITAKIS NIKOLAOS

in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

THESIS APPROVAL

Author: _____
Ftylitakis Nikolaos

Committee Approvals:

Supervisor _____
Constantine Stephanidis
Professor

Member _____
Apostolos Traganitis
Professor

Member _____
Dimitris Grammenos
Principal Researcher

Department Approval: _____
Angelos Bilas
Professor, Director of Graduate Studies

Heraklion, March 2013

Abstract

By now, the number of the smartphones in use is estimated to billions, being an integral acquisition of a large part of the population and the everyday life. Meanwhile, mobile phones have evolved rapidly both from software and hardware aspect. In this way, each user carries a potential means that can provide a great range of capabilities if utilised properly. Meanwhile, the concept of Ambient Intelligence involves the exploitation of every resource, technical or not, to improve the user experience of interplaying with an "intelligent" environment.

This thesis will focus on the convergence of the world of smartphones with that of ambient intelligence. Great benefits can be induced through the utilization of smartphone capabilities as a means of controlling a smart environment or as the medium for the environment to "sense" user status and intentions.

The creation of a toolkit is proposed to handle the reciprocal communication between the smartphones and the Ambient Intelligence Environments. It is a multiplatform application targeting most of the modern mobile operating systems. It is used to expose the smartphone data and capabilities into the Ambient Intelligence Environment, based on a Service Oriented Architecture.

The feature list of the toolkit was designed to fulfill requirements set by the different research fields currently active in Ambient Intelligence. The context aware systems are benefited by the complete list of exposed sensors which includes inertial, geo-location and environmental sensors. Moreover, device data are extracted such as the profile and lock status, the charging state and currently selected operation language. Gesture-based systems are handled by 7 supported gesture recognizers having as input the physical movements of the mobile device into 3D space. In parallel, the NFC support will come to elevate the touch-less interaction with the environment. The list of features is completed by the support of the direct access to the file system of the smartphone.

Two projects were implemented using the proposed toolkit. SmartCube is a custom Smart Object to control Smart Home appliances. Moreover, the "A day at work" scenario explored the smartphone integration into office environments.

An evaluation study was conducted with 11 participants to assess the gesture recognition rates and the user experience in gesture-based systems through smartphone. The results were encouraging since most of the participants considered such an interaction method as useful and easy to use.

Περίληψη

Ο αριθμός των έξυπνων κινητών τηλεφώνων πλέον μετράται σε δισεκατομμύρια. Είναι απόκτημα ενός μεγάλου μέρους το πληθυσμού και κομμάτι της καθημερινής ζωής. Ταυτόχρονα τα κινητά τηλέφωνα έχουν εξελιχθεί ραγδαία τα τελευταία χρόνια σε θέματα υλικού και λογισμικού. Το γεγονός αυτό καταδεικνύει ότι ο κάθε χρήστης διαθέτει μαζί του ένα δυνητικό μέσο που μπορεί να προσφέρει ένα μεγάλο εύρος λειτουργιών αν αξιοποιηθεί σωστά. Ταυτόχρονα, η έννοια της Διάχυτης Νοημοσύνης συμπεριλαμβάνει την εκμετάλλευση κάθε τεχνολογικού, ή μη, μέσου στην βελτιστοποίηση της εμπειρίας του χρήστη κατά την αλληλεπίδραση του με ένα έξυπνο περιβάλλον.

Η εργασία αυτή θα εστιάσει στην σύγκληση του κόσμου των έξυπνων κινητών τηλεφώνων με τον κόσμο της Διάχυτης Νοημοσύνης. Η αξιοποίηση των δυνατοτήτων ενός έξυπνου κινητού τηλεφώνου μπορεί είτε να δημιουργήσει ένα μέσο ελέγχου του περιβάλλοντος χώρου είτε ως το μέσο για να μπορέσει το περιβάλλον να «αισθανθεί» την κατάσταση του χρήστη και τις προθέσεις του.

Για την δημιουργία μιας γέφυρας επικοινωνίας μεταξύ των κινητών τηλεφώνων και των Περιβαλλόντων Διάχυτης Νοημοσύνης, προτείνεται η κατασκευή ενός συνόλου εργαλείων. Τα εργαλεία αυτά θα υποστηρίζονται ταυτόχρονα για όλα τα μοντέρνα κινητά λειτουργικά συστήματα. Ο στόχος τους θα είναι η διάθεση των δεδομένων και των δυνατοτήτων των κινητών τηλεφώνων σε Περιβάλλοντα Διάχυτης Νοημοσύνης βασιζόμενα πάνω σε μια υπηρεσιοκεντρική αρχιτεκτονική

Η λίστα των υποστηριζόμενων λειτουργιών από τα εργαλεία αυτά σχεδιάστηκε για να ικανοποιεί τις προϋποθέσεις που έχουν τεθεί από τα διαφορετικά ενεργά ερευνητικά πεδία γύρο από την Διάχυτη Νοημοσύνη. Τα συστήματα επίγνωσης πλαισίου επωφελούνται από την πλήρη υποστήριξη αισθητήρων όπως οι αδρανειακοί αισθητήρες, οι αισθητήρες γεωτοποθέτησης και περιβάλλοντος. Επιπλέον, παρέχονται δεδομένα που αφορούν την συσκευή όπως το επιλεγμένο προφίλ, την κατάσταση κλειδώματος και φόρτισης και την ενεργή γλώσσα επιλογής. Τα συστήματα που βασίζονται σε αναγνώριση χειρονομιών καλύπτονται από τις 7 υποστηριζόμενες χειρονομίες οι οποίες παράγονται από την φυσική κίνηση του κινητού τηλεφώνου στον τρισδιάστατο χώρο. Ταυτόχρονα, η διάθεση της NFC τεχνολογίας έρχεται να επαυξήσει την αλληλεπίδραση με το περιβάλλον μέσω προσομοίωσης της αφής. Η λίστα ολοκληρώνεται με την υποστήριξη της πλήρους προσβασιμότητας στο σύστημα αρχείων του κινητου΄τηλεφώνου.

Δύο εφαρμογές αναπτύχθηκαν πάνω στα παραπάνω εργαλεία. Ο έξυπνος κύβος είναι μια ιδιοκατασκευή που έχει ως σκοπό τον χειρισμό οικιακών συσκευών μέσα σε ένα έξυπνο σπίτι. Επίσης, το σενάριο «Μια μέρα στην δουλειά» ήταν το έναυσμα για την έρευνα γύρω από την ενσωμάτωση των κινητών τηλεφώνων σε εργασιακά περιβάλλοντα.

Τέλος, έλαβε χώρα μια αξιολόγηση με 11 συμμετέχοντες για να διερευνηθούν τα ποσοστά επιτυχίας αναγνώρισης των χειρονομιών που εκτελούνται μέσω της κίνησης των κινητών τηλεφώνων. Ταυτόχρονα αξιολογήθηκε και η εμπειρία χρήσης εφαρμογών που μπορούν να χειριστούν μέσω χειρονομιών. Τα αποτελέσματα ήταν ενθαρρυντικά καθώς το μεγαλύτερο ποσοστό των χρηστών χαρακτήρισαν αυτή την μέθοδο αλληλεπίδρασης ώς χρήσιμη και εύκολη στην χρήση.

Acknowledgements

I would like to thank my supervisor Prof. Constantine Stephanidis for the support and encouragement throughout the conduction of this thesis.

Furthermore, I would like to thank my colleagues at the Human-Computer Interaction laboratory of ICS-FORTH, especially Dr. Dimitris Grammenos, and Georgios Margietis for their guidance throughout this thesis.

The work reported in this thesis has been conducted in the context of a scholarship provided by the Institute of Computer Science (ICS) of the Foundation for Research and Technology – Hellas (FORTH), including financial support and the use of the necessary equipment and facilities.

Additionally, I would like to thank my friends Nikolas, Manos, Panos, Chyssa, Vasilis and Akis for cheering up everyday life. Separately I would like to thank Giannis for his valuable assistance. As a special mention, I owe my gratitude to Vasso for her psychological and practical support during the conduction of this thesis, without her the outcome would not be the same. Finally, I thank my family for all those years of trust and support.

Table of Contents

1 Introduction 5

2 Background and Related Work 8

2.1 Ambient Intelligence 8

2.1.1 Interaction techniques 10

2.1.2 Context awareness 17

2.2 Related work 18

2.2.1 Context Extraction 18

2.2.2 Gesture Based Systems 21

2.2.3 NFC based Systems 23

2.2.4 Toolkits 24

2.3 General Considerations 26

2.4 Privacy concerns 28

3 Motivation and Contributions 29

4 Design Considerations 34

4.1 Multiplatform support 34

4.2 Utilized Hardware Technologies 37

4.2.1 Inertial sensors 37

4.2.2 NFC 40

4.3 AmIE Integration 41

4.3.1 Selected Design Approach 43

5 Harvesting & Exposing Data 48

5.1 Information exchange guidelines 48

5.1.1 Connection Type 48

5.1.2 Exchanged data format 50

5.1.3 Message contents 52

5.2 Smartphone Harvester mobile application 55

5.2.1 Core 56

5.2.2 Socket Handler 58

5.2.3 Sensors 59

5.2.4 Gestures 60

5.2.5 Audio Manager 68

5.2.6 Command Executer 68

5.2.7 Hardware & Device Information Manager 69
5.2.8 Network Interface Manager 69
5.2.9 GUI 70
5.2.10 Completing the gaps in Qt's compatibility 71
5.3 SmartphoneHarvester Middleware Proxy 73
5.3.1 Proxy 74
5.3.2 Broker 75
6 Towards Multimodal Applications 77
6.1 A day at Work 77
6.1.1 NFC Message Server 78
6.1.2 Smartphone Door Control 80
6.1.3 Welcome Screen 80
6.1.4 Mobile at Work Assistant 81
6.1.5 Presentation Table 85
6.2 SmartCube – home appliances control through a Cube 91
6.2.1 Raspberry Pi 92
6.2.2 SmartCube design 92
6.2.3 Home interaction 94
7 Evaluation 98
7.1 User Evaluation 98
7.1.1 Set-up and Participants 98
7.1.2 Evaluation Goal & Process 98
7.1.3 Results 99
7.2 Multiplatform Support 104
8 Conclusion 107
8.1 Summary 107
8.2 Contributions 108
8.3 Future Work 112
9 References 114
10 Appendixes 119
10.1 List of Abbreviations 119
10.2 Famine Service Descriptions 120
10.3 Evaluation Questionnaire 123

List of Figures

Figure 1 The Human-Centric Design in AmI 9
Figure 2: Reactable(Up Left), Tangible Bots (Up Right), Museum Exhibit (Down) 14
Figure 3: Siftables & AudioCubes 15
Figure 4: Implicit / Explicit mobile interaction 17
Figure 5: E-Gesgture & Magic Ring 23
Figure 6: Smartphone Statistics 34
Figure 7: Sensors 3D Axis (from qt-project.org) 38
Figure 8: Device's sides (from qt-project.org) 39
Figure 9: Device's orientation values (from qt-project.org) 39
Figure 10: 3D rotation Axis (from qt-project.org) 40
Figure 11: Famine middleware architecture 42
Figure 12: Initial Smartphone - Proxy architecture 45
Figure 13: Final Smartphone - Proxy - Broker architecture 47
Figure 14: Example JSON object 51
Figure 15: Proposed Protocol enumerations 53
Figure 16: Proposed message format 55
Figure 17: SmartphoneHarvester mobile application architecture 57
Figure 18: Initial GUI design 70
Figure 19: Final GUI 71
Figure 20: Broker application architecture 73
Figure 21: NFC message server Interface through IDL language 79
Figure 22: Welcome message on screen 81
Figure 23: Mobile At Work Assistant views (1/2) 81
Figure 24: Mobile At Work Assistant views (2/2) 82
Figure 25: Presentation Table interface of User Area 87
Figure 26: Different Toolbars in Presentation Table application 90
Figure 27: SmartCube initial design 93
Figure 28: SmartCube final implementation 94
Figure 29: Multiplatform Support (iPod, HTC Desire HD, Nokia 701) 105
Figure 30: Samsung Galaxy SII mini 105
Figure 31: Meego emulator 106
Figure 32: Raspberry Pi Raspbian “wheezy” 106
Figure 33: Multiplatform Feature Support 106
Figure 34: SmartphoneHarvester Service Interface (part 1/3) 120
Figure 35: SmartphoneHarvester Service Interface (part 2/3) 121
Figure 36: SmartphoneHarvester Service Interface (part 3/3) 121
Figure 37: SmartphoneBroker service Interface 122

List of Tables

Table 1: NFC tag types 41
Table 2: Shared Folder info 68
Table 3: List of interaction correspondence between SmartCuve and smartphone 97
Table 4: Gesture recognition rates 100
Table 5: Presentation table user tasks 102

List of Diagrams

Diagram 1: User Success Rate in Presentation Table 103
Diagram 2: Questionnaire results 104

1 Introduction

Smartphones have emerged during the past 5 years. The birth of iPhone and the creation of Android platform established a new way of thinking when it comes to handheld devices. Through a frantic race of performance improvements and innovations, the smartphone has evolved into an easy operable computing unit. The hardware has become a powerful palm-sized computer with enhanced capabilities that can overrun a basic personal computer. Among the enhancements, a plethora of sensing techniques has got embedded to the hardware used to apply adaptations to the smartphone, leveraging user experience. The software platforms in parallel have become mature, easy to use and programmatically accessible, exposing the complete underlying functionality. In this context, considering that the number of the smartphones in use is esteemed to be billions, the technological innovation is existent in the everyday life.

Meanwhile, Ambient Intelligence is progressing by the utilization of every resource, technological or not, to amend the user experience within “Intelligent” environments. The Smart Environments, as part of the Ubiquitous Computing concept, incorporate omnipresent and non-obtrusive technological infrastructures to improve the life of its habitants. Being human centric, it proactively acts based on user intentions and reacts to user’s actions, to assist the achievement of his scope. Thus, it creates a direct connection between the human and the surrounding environment.

The convergence of the smartphone world with that of Ambient Intelligence constitutes the main idea of this thesis. Although the interplay of the aforementioned worlds has already taken place, there is no consistency and forward think design across the different approaches that are proposed till today. The number of the active smartphones is continuously growing, indicating that an increasing number of people are progressively acquiring a tool. If this tool is correctly exploited, it could be transformed into a tool for controlling the user’s surrounding environment or into a sensing medium for adaptations of the environment through user status recognition.

Through this work, the creation of a toolkit is proposed to elevate the integration of the mobile devices into the Ambient Intelligence Environments. It will be designed regarding both the end users and the application developers. The end user experience ought to be intuitive with low cognitive effort, non-intrusive to the use of the smartphone and easy to install and configure. On the other hand, the developer has to be equipped with an easily

exploitable toolkit with comprehensible API, exposing a verbose list of device capabilities. Most importantly, it is obligatory to have complete integration with Ambien Intelligence environments and to be able to provide data that can be exploited in multiple research fields such as context awareness and gesture-based systems. The proposed toolkit, namely SmartphoneHarvester, is a multiplatform smartphone application that will expose the smartphone capabilities into Ambient Intelligence Environments.

The multiplatform support was a key aspect in order to support the widest variety of devices possible, directly affecting the number of deployable devices. The toolkit has to be able to provide abstraction across the different mobile operating systems. Also, the implementation was crucial to be developed once and be deployable everywhere eliminating the cost of supporting multiple implementations. Moreover, since the smartphone market could probably change in the next 5 years, the toolkit must be able to handle future platforms for future compatibility.

Regarding the capability exposure, the toolkit initially has to have an established connection with an infrastructure that will be able to redistribute the captured data and the commands. The complete integration to Ambient Intelligence systems can provide the above since the utilization of environment's middleware will enable the redistribution of the data from the mobile device to any other application. Also, since the connection is bidirectional, the remote applications will be able to place requests directly to the mobile device.

The exposed data has to cover a complete range of data sources. Being used in an Ambient Intelligence environment, context aware systems will need data for activity recognition whereas gesture-based systems will require gesture recognition as a means of controlling smart home appliances. In this way, the final list of supported data sources includes, but is not limited to, accelerometer, magnetometer and gyroscope sensors, orientation and proximity sensors, GPS, NFC. Finally, the exposure of newly data sources that were not available in the past could open up new research practices.

The structure of the remaining thesis is divided into the following chapters:

Chapter 2 will provide an up-to-date overview of all the related work around the field of mobile data exploitation starting by the definition of Mobile Human-Computer Interaction.

Chapter 3 will provide a more in depth analysis of the motivation statement and will enlist the desired contributions and goals that this thesis intended to address.

Chapter 4 will discuss the design considerations that were faced before the development of the toolkit. This includes the system architecture and the selection of the appropriate tools to build upon for multiplatform support

Chapter 5 is dedicated to the analysis of the implementation of SmartphoneHarvester toolkit. The communication protocol that was specified introduces the chapter, followed by a thorough investigation of the mobile application's design and implementation. The chapter ends by presenting the server application that will connect the mobile application to the server

Chapter 6 will present two end user applications developed as clients to the SmartphoneHarvester toolkit. Those will be used as a reference point that is the proposed development approach.

Chapter 7 will include the evaluation results regarding the gestural recognition rates and the system usability, along with the user experience.

Chapter 8 concludes by presenting the key contributions made regarding the goals that were defined through the motivation and contribution statement in section 3. Ultimately, this thesis ends with a description of the future work to be done.

2 Background and Related Work

2.1 Ambient Intelligence

Computer science has succeeded a tremendous evolution during the past few decades. Thirty years ago the computer presence in our everyday lives was inexistent. Processing machines were infrequent, weak in processing power and slightly inter-connectable. Over the years, personal computers evolved, portable devices emerged and a great number of microprocessors were embedded to nearly everything. Network connected devices became common, creating a vast grid of information exchange.

Early enough researchers foresaw the progress, envisioning a future where all the technological inventions will proactively act for us. Information processing will take place in a seamless way by interconnected devices, spreading around our environment, mostly beyond human perception. This idea got formalized by the creation of Ubiquitous Computing theory introduced by Mark Weiser in 1991. Weiser's main conception gets summarized as "*Machines that fit the human environment instead of forcing humans to enter theirs*" [14].

Ubiquitous computing proposes the creation of environments where hidden computational systems continuously track, adapt and act. Tracking will be possible using various sensors, providing, among others, the location and the activities of any user. Using the captured information, the system will be able to adapt to user needs and, finally, act proactively to help the user achieve what intended. Initial implementations could potentially be considered intrusive but, following the progress of technology and people's increasing familiarity with the computer presence, such practices are applied to our surrounding world. "*…only when things disappear … are we freed to use them without thinking and so to focus beyond them on new goals*" [14].

By the end of 2000, the deployment of software had increased significantly. Despite the creation of feature-rich products, the usability and the actual user experience were not taken into account, resulting in long training periods per product or service. Driven by this aforementioned fact, Zelkha and Epstein [15], on 1998, introduced Ambient Intelligence (AmI) field of research. AmI aimed to amend the way the users interact with Ubiquitous Computing Environments and improve the User Interfaces (UIs). This idea promoted the concept of the Human-Computer Interaction applied into Ubiquitous Computing Environments.

In this way, the user became the most important aspect of any system. Everything is built tailored to meet individual needs with respect to user opinion [16]. For that purpose, iterative User Evaluations are conducted at various development states of an AmI system to collect feedback. Thus, the designer and the users are co-creating the system [17].(Figure 1 The Human-Centric Design in AmI)

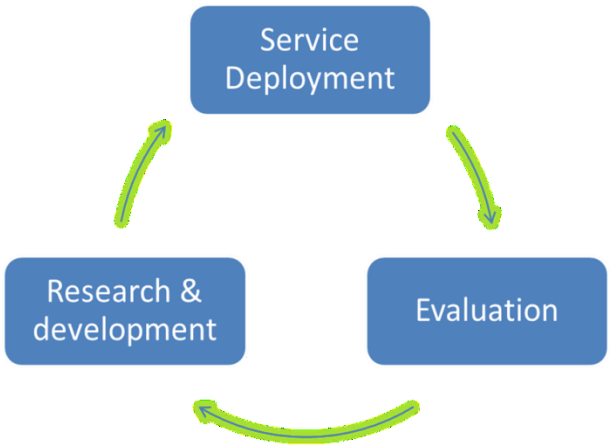

Figure 1 The Human-Centric Design in AmI

A Ubiquitous Computing Environment (UCE), in the context of Ambient Intelligence, will be renamed into Ambient Intelligent Environment (AmIE) [16]. That is, an anticipatory system [18] waiting for the user's action, ideally predicting it and acting in a non-obtrusive way.

AmIE's core components can be listed as follows;

- *Hardware:* Embedded, network-enabled devices integrated into the surrounding space acting as input and output devices. They could include cameras, microcontrollers, sensors, mobile devices, screens, personal computers etc.
- *Physical interaction:* User's physical movements in the environment are translated into commands or activities. The act of operating an input controller could be also included.
- *User Interfaces:* The direct visual output generated from a computational unit mainly projected on a screen. Commonly includes active controls which can be accessed by the use of an input method.
- Reasoning unit: the software running at the background orchestrating every input / output and reasoning on how the environment should act upon receiving any input.

In the following sections, the two fundamental concepts of Ambient Intelligence, that this dissertation was built upon, will be explained. To begin with, the various input methods will be analyzed in the paragraph of the Interaction Techniques, followed by the introduction to Context Awareness and its area of application.

2.1.1 Interaction techniques

In the context of smart environments a fundamental concern lies on the techniques through which the human user applies physical actions as input for the underlying intelligence system. In general, the word *interaction* is defined as "*a mutual or reciprocal action or influence*"[1]. By definition, it suggests the application of an action upon a subject, with or without receiving a subsequent reaction. The interaction in Ambient Intelligence, or Human-Computer Interaction (HCI), describes the physical actions executed by the users in a technological-rich environment as a means of producing input events for the Ambient Intelligent systems, combined with the environment outputs which stimulate the human senses [19].

Bearing in mind the HCI definitions given by Dix et al. [38] and Wilson et al. [37], the interactions can be divided in two separate categories; explicit and implicit. *Explicit* is when the user initiates an action anticipating a reaction. *Implicit* interaction utilizes passive monitoring of the user, extracting information which is later used for interaction adaptation. Going further, the *explicit* interaction gets subcategorized into *direct* and *indirect* The direct interaction involves the users' first-hand interaction with the system from the beginning to the fulfillment of the task. The *indirect* contains the usage of sensors or extra processing as a means to execute the control: the sensors capture the users' actions and comprehend them as implicit input.

Hence, the emerge of AmIE enabled the creation of a wide variety of input and output devices and methods. Among the most profound examples are many interconnected devices which cooperate to detect human intentions, sensors that capture the user movements translating them to commands, custom devices with buttons and touch-screen areas that provide tangible interaction. The list can be easily expanded and will continue to grow over the years. As a result, it is common to encounter various input modes such as speech, pen, touch, hand gestures, gaze and head / body movements [20,21,22] along with

[1] WordReference online dictionary @ http://www.wordreference.com

basic output devices like displays, lights and sounds. In order to shape a basic structure of the already existing solutions on which we will built upon, it would be useful to include a brief enlistment of the interaction techniques applied to different approaches met in the literature.

- *Audio*:

Audio was one of the first input methods extensively used. Initially, it was used for speech recognition so as to enable computer control over voice, helping impaired persons interact with their environment [23]. In this case, apart from the understanding of what it said, it was as well used to detect the identity of the person acting. Later it incorporated the localization of the user by the initial application of microphone arrays [25], or, more recently, a set of Kinect devices [26]. Finally, emotion extraction can take place by capturing shouts or laughter. Despite all that, speech recognition does encounter two main issues; the medium recognition rates and the limited vocabulary available [40].

- *Pen:*

Before the appearance of the touch screens, pen was the dominant tool used to directly interact with screens, by either pressure or electric conductance applied from the pen to a special surface. Nowadays, they are utilized through cameras combined with emitting Infrared light source simulating the mouse input on large physical areas [27]. This approach has the physical drawback that the light must be directly visible by the camera. In that way, an effective angle range of usage is defined and can be broadened by the use of more cameras.

- *Touch:*

Touch is the most widely adopted interaction technique, dominant in personal computers, mobile phones, embedded solutions, museum exhibits and information kiosks. Mainly exists as an extra-layer placed upon screens enabling finger interaction directly on the viewed artifacts and controls [19].

- *Gestures:*

Any human movement that can consistently be reproduced is eligible to be considered as a gesture. This includes, but is not limited to, body movements, multi-touch actions or single-pointed drawn paths on 2D spaces. The gestures can be associated with

immediate controls over anything that can be controlled. They are considered intuitive since the meaning of the gesture can be self-explained by the actual path of the movement [28, 29]. For example, the movement of a hand starting from its idle position moving upwards could be linked with the opening of the window's shades, as it is the exact same movement of the hand as if it was physically applying the same action on the shades. Furthermore, the variety of a person's movements and the ability to combine gestures creating others, suggests that the available vocabulary in theory can be limitless.

As in any other area of research, gesture usage inflicts drawbacks. Most of the existing appliances require extra hardware depending on the types of gestures that need to be recognized. The extra hardware could include cameras but could also consist of intrusive wearable devices. Moving to the detection process, the support of custom gestures implies the use of demanding classifiers in terms of processing power and possibly long training periods. Finally, when designing a gesture-controlled system, the architects need to take into consideration the amount of physical effort that is needed to execute the gestures repeatedly which could lead to the users' fatigue.

- Body & Head Movements:

Body movement is an interaction technique where the system uses as input the user's body pose or position in the environment. The user's location in space is a very natural interaction technique, especially in non-constrained systems, where users are allowed to freely move, as the systems respond to the movement of the user. Grammenos et. al. [71] developed a multi-user system which displays an ancient painting and presents additional information with regard to the users' position (i.e. areas of interest and restored parts). Another similar category of body movement as means of interaction is interactive floors, which display information at the floor using bottom projection. Such an example is [72], who apply interactive floors to offer kinesthetic interaction for children.

Head tracking refers to the technique where the systems track the user's head and have information about its orientation, mainly applicable in augmented and virtual reality systems. An indicative example is [73], who apply head tracking to monitor the driver's awareness. Furthermore, head tracking is often combined with head-mounted displays, as mentioned in [74]. Head tracking is usually applied in combination with other techniques, as on its own it fails to provide information to the system whether the user aims to interact with the system or is simple looking around.

- Smart Objects:

In the former techniques, the user is completely detached from the use of any external device. Frequently though, the necessity arises to provide tangible objects that support a predefined number of functionalities while being easy to comprehend. It is a natural object that is directly operated by a human, through physical movements applied on it, translating them into input for the system. Mark Weiser [14] proposed the first categorization of Smart Objects according to their size; tabs, the wearable centimeter sized devices; pads, the hand-held decimeter-sized devices and boards, meter sized interactive display devices. Regardless of their size smart objects need to be connected to a network, with Micro Electro-Mechanical Systems (MEMS), potentially light-emitting, constructed from materials that conform to the main characteristics of the current environment [31]. The basic goal is to blend seamlessly in the environment of usage and becoming part of it.

2.1.1.1 Multimodal Interaction

The above defined interaction techniques provide a complete set of input methods that enable new schemes of communication between human and the digital world. The combined use of two or more interaction techniques formulates the Multimodal interaction. As described by Oviatt et al. [22] "*Multimodal systems process two or more combined user input modes in a coordinated manner with multimedia system output*".

2.1.1.2 Tangible Interaction

The term tangible has come up in the Smart Object definition, deriving from the fact that the Smart Objects constitute the broader concept of Tangible User Interfaces (TUI). In this concept, the user manipulates the physical objects which are sequentially recognized by the system. These physical objects transfer the user's control to the system, creating a link between the physical and the digital world [51]. Although the Smart Objects fulfill the above definition, TUIs are not restricted only to their use Objects without embedding any technological components can be used in combination with a number of detection methods to provide corresponding controls.

For example, the reacTIVision project [52] has greatly contributed to the engagement of technology-free objects. It did not only provide a framework but also the guidelines for the creation of smart tables with interactive surfaces, with the surface being both the screen and the detection area of the objects placed on the table. Each object's

bottom side hosts a tag as the mean for identification. Taking advantage of this framework, dozens of projects with impressive demonstrations of innovative TUIs have arisen (Figure 2: Reactable(Up Left), Tangible Bots (Up Right), Museum Exhibit (Down)). Additionally, the Reactable [53] is one of the most well-known projects that generate music from the objects currently placed on a surface, taking into account their orientation and their proximity to each other. Moreover, Tangible Bots [54] introduces the autonomous tangibles with live feedback and interconnection between them, while Combinatorix [55] uses simple tiles to help collaborative learning of probabilities. Moving away from the reacTIVision applications, in Grammenos et al. [75] it is proposed the creation of an interactive coffee table whose functionality is altered by the use of different table cloths. Finally the system proposed in [56] is an interactive museum exhibit projecting information on tracked plates.

Figure 2: Reactable(Up Left), Tangible Bots (Up Right), Museum Exhibit (Down)

In the literature of TUIs, it is quite common to come across custom cubes containing microcontrollers. Early enough Kranz at al. [57] implemented a cube as a concept device for tangible interaction, with possible application in children's creative activity and learning. It used 3D accelerometer as the main input device and was able to detect simple gestures. Its outputs were 6 small screens, one on each side, showing low-resolution black and white images combined with simple auditory feedback by a small speaker. Interestingly enough, it could run with very low energy consumption. Later, the AudioCubes [58] advanced the cube idea into a tangible interface for generating sounds. They are light emitting cubes with optical sensors on the 4 vertical sides that sense the surrounding cubes and generate sounds according to the proximity with the other. Finally, the proximity idea enabled Siftables [59]

to create little tiles equipped with small screens which are able to interact with each other. Information is exchanged through infrared data transmission, while their orientation can be tracked by an accelerometer placed on each tile. Thus, they created an adaptable system in which different sequences of tiles produced different outputs (Figure 3: Siftables & AudioCubes).

Figure 3: Siftables & AudioCubes

2.1.1.3 Mobile Human-Computer Interaction

Moving to the mobile device world, the interaction between a mobile device and a human user shapes the term of Mobile Human-Computer Interaction. As described by Love et al. [33], the Mobile Human-Computer Interaction is the research of human interaction with mobile computing systems. In this field, the user's expectations are taken into account, adapting the mobile systems and mobile applications to the user needs. This denotes any action performed by the user on the mobile device combined with the analogous feedback.

Simple actions can be included, such as making a phone call, writing a text message, browsing over the internet and gaming. The real challenge here is that everything gets executed in a limited set of device capabilities. The hardware limitations, combined with the limitations imposed by the portable nature of the mobile devices, have created this separate research field[1]. Its main areas of interest focus on the input and output methods applied on mobile devices along with the design of specially adapted user interfaces to improve user experience. This research field got established by the first Mobile HCI conference in 1998 and evolved over the years; its current state can be perceived from the latest analysis of today's trends and opportunities [34, 35, 36] around this field.

[1] Mobile HCI conference @ http://www.mobilehci2013.org/

2.1.1.4 Physical Mobile Interaction with AmIE

By taking a look at today's mobile phone specifications, we can see a number of inertial sensors (accelerometer, gyroscope), connectivity modules, wide screens and extendable application bases. This fact lures the research community to exploit those capabilities. In this thesis, we are going to focus on the appliance of mobile phones in Ambient Intelligence environments.

Going back to the definition of the Smart Objects, which was given in paragraph *Interaction techniques* (section 2.1.1.1), and reviewing their main characteristics, it is clear that mobile phones fulfill the requirements to be considered as Smart Objects. The inertial sensors satisfy the existence of MEMS, the 3G and WiFi modules provide continuous connectivity and profound seamless integration of the device into the everyday life. Recalling the *explicit* and *implicit* interaction categorization, the Smartphone usage in Ambient Intelligence Environments can be projected on them (Figure 4: Implicit / Explicit mobile interaction). In this thesis, both of these types will be subjects of usage.

Firstly, the explicit interaction adapted on the mobile device usage implies the use of a mobile phone as a means of transferring an action applied on it to the environment with the intention of controlling a specific device or service. This definition formulates the concept of *Physical Mobile Interaction with Ambient Intelligence Environments*. The above process can involve one or more input methods supported by the mobile phone. The microphone can be used along with speech recognition to produce commands to control house contents [39]. Key events can be used as shortcuts that trigger operations such as media control of a Hi-Fi [68]. Touchscreen availability, along with third-party applications can support a variety of functionalities. Graphical Interfaces can be designed to provide the front-end to control nearly everything contained in a smart environment [40]. Cell phone's camera images can be used, among others, to extract the environment's attributes like light color and intensity [5] or to scan and identify barcodes [41]. The use of sensors translates user's movements into data through accelerometers, gyroscopes, GPS etc. These data can form gestures to provide control through natural movements [10, 40]. A more in depth analysis of such systems will follow in section 2.2.

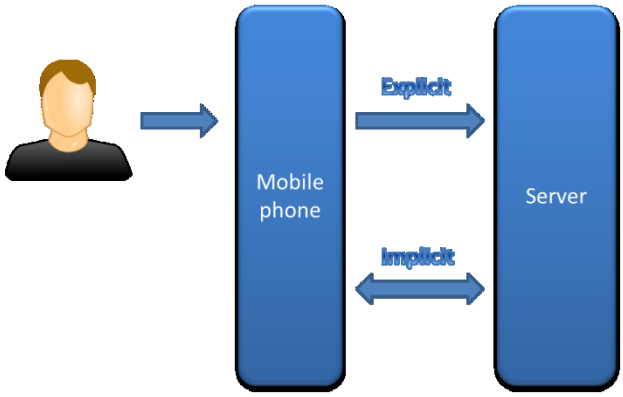

Figure 4: Implicit / Explicit mobile interaction

The implicit interactions through mobile phones are elevated by their passive operation. Nowadays, people are surrounded by smartphones, as they are present nearly at any action taken in their lives. The smartphone receives the same natural forces as its owner and senses the surrounding environment: It is easy to imagine the existence of software applications installed and running at the background, omnipresent, constantly extracting information of the surrounding environment. Such examples can be found in [1,2,4,5,6,7], which will be thoroughly explained in section 2.1. This is the basis of the appliance of context-awareness and context extraction through mobile phones. Context-Awareness will be explained in the next section (2.1.1.2) while existing work based on its theory will be presented in section 2.2.1

2.1.2 Context awareness

The term "context-awareness" in ubiquitous computing was firstly introduced in 1994, by Schilit [42]. Relying on this definition, a context-aware system is able to sense the environment of the user, detect the changes and adapt to them, taking into account the available services and the people that surround the user. Dey and Abowd [8], further evolved the definition by dividing the primary context types into location, identity, activity and time followed by the secondary context containing personal information regarding the user. The four abovementioned types, according to Prekop and Burnett [9], form the external context, which is the measurable part of the environment. The cognitive data, on the other hand, shape the internal context containing thoughts, intentions and feelings. Context awareness lies under the implicit interactions in an AmIE and will be further explained in section 2.2.1 from the smartphone point of view.

2.2 Related work

Smartphones are an exciting new world of hand-held computers capable of introducing new aspects to Human-Computer Interaction. Palm sized screens, powerful processors, inertial sensors are all accessible through mature Application Programming Interfaces (APIs). As previously stated, a smartphone constitutes a perfect Smart Object ready to be put in action.

In the following sections, there will be a presentation of the already existing projects in the research areas that are going to be covered in this thesis. We will start by exploring Context-Aware Systems using smartphones for context extraction in section 2.2.1. Moving on in section 2.2.2 the usage of the embedded inertial sensors to recognize gestures so as to control user's environment will be covered. Section 2.2.3 contains the usage of NFC (Near Field Communication) to introduce touch interaction and Section 2.2.4 will state a reference to toolkits as means of abstracting input.

2.2.1 Context Extraction

Reviewing the concept of context awareness through past definitions, while taking into account the human-centric perspective [1,5] of the context, it can be divided into three categories [1]. The *physical context* describes the actual world around the user; the *internal context* contains non-tangible ideas like user feelings and the *social context* that gathers the user interaction with other people. The above information can adequately be extracted through smartphones by harvesting sensors and application data. A smartphone is the most effective means of achieving user data extraction. Receiving the exact physical forces applied to the user makes the description of the surrounding environment (physical context) feasible. In parallel, the continuous incensement of data packet usage through smartphones establishes a mechanism of exploiting the data shared over the internet (internal & social context). The context-aware applications can be divided into three discrete categories [1].

- Appliances where users are free of carrying any device, yet functionality remain existent. They are considered as Intelligent Applications.
- Applications whose provided content adapts based on user's context.
- Applications that are mainly focused on the interconnection between the users and the inter-sharing of their contexts such as location information.

This dissertation will mainly focus on the first application type, with an addition to the third type worth mentioning. As we will see, the majority of the projects have a main structure of functionality. That is, continuous logging of data on the phone and periodic, or at will, uploading of data to a server for analysis.

We will start by presenting a project created to help research purposes. ContextLogger2 [2] was built for logging as much information as possible regarding the user actions taken on a smartphone, accompanied by device information. Among the logged information we can find a wide variety of data. Some examples are hardware data retrieved by the background daemons such as GPS data, key-press events, Bluetooth proximity scans, or software data like profile changes, call status, application focus etc. Those data are logged in a SQLite3 [3] database hosted at the mobile device and are sent to a server via HTTP POST message on request, with the SSL usage being optional.

Moving to an application which actually creates an output, SurroundSense [5] achieves location recognition through fingerprinting of the place's identifiable contexts and making the transition from physical localization (GPS) to logical localization (unification of identifiable contexts). Supposing that closely located shops have diversity in ambient environmental data they create a fingerprint for each location during the training period. That information is later used for the localization of the user matching fingerprint pairs. A fingerprint contains the sound, the color, the light and the human movement pattern combined with the classic GPS, GSM and Wifi data. When all the above context data are used in conjunction, the system can recognize the location of the user with average success rate of 85% according to the authors. It has to be noted that, the recognition is not real-time, although it is stated as future work, since each captured data have to be separately uploaded to the server, processed by classifiers to produce the result.

In both aforementioned projects, their data were used for completely technical manners. A very interesting and active area of research though is how to use mobile phones to implicitly extract data from their users. Such data include the current activity of the user, the surrounding environment or even the identification of feelings [4, 6, 43]. In this context, the infiltration of Social Network Sites (SNS) is noticeable [4, 6], both as a data source and as an output for the processed results.

Furthermore, CenceMe [6] presents a complete sensing platform making the most out of a smartphone. All the hardware components provided by a smartphone device are

used to classify the use's activity (Standing, Walking, Running), the mobility status (Stationary, Walking, driving), the indoor or outdoor presence and to detect whether the user is having a conversation. Furthermore, the system detects the user's places of interest based on collected GPS data and assumes the exposure to UV radiation combining the weather forecast and the indoor/outdoor status with the mobility status that indicates the presence of the user in a vehicle. In this way, the user context is formed after uploading of the data to external servers. CenceMe was the first to introduce the use of SNS in context awareness systems. After the successful context extraction, it is able to automatically inject it in user's Social Network accounts.

Taking a step further, Gonzalo Bl´ azquez Gil et al [4] proposed the inContexto project, the successor of CenceMe. Its main target is to achieve real-time complete user state extraction, retrieving both activity and emotional data though micro electro mechanical systems (MEMS) and device application data respectively. Its difference lies on the fact that the SNSs are used as an input. Such sites contain thoughts and emotions deliberately provided by the user through the smartphone. A good practice of this project is the unification of simple activities, like "walking", with the GPS signal that points at the mall into higher level activities like "walking at the mall". The rates of data collection were increased so the use of cloud computing became a necessity. Interestingly enough, this approach had increased recognition rates and compared to previous literature [5,6,7] there is a notable maximum increment of 19%. Another useful note taken by this research is their concern about the device's placement on the human body. Small changes in the device's position can produce invalid results, implying that future systems need to be designed to handle such cases.

By reviewing the above literature around the context awareness a trend can be identified: research is progressively moving from the physical world to the world of senses. It has become a challenge to identify the user's feelings and thoughts. If, at some time, those can be extracted in real time we will be half way to the truly proactively engaged systems. An interesting work that resembles to it, is the identification of personality characteristics in [43]. The first phase of the project was about data collection. An application running on the mobile phones of the 83 contestants were collecting data originated by calls, messaging, Bluetooth scan results and statistics on application usage habits. After collecting data for 8 months, the users were asked to state their personality characteristics. Those data were

compared to the results of the data gathering and classification and resulted in an impressive 75% of convergence.

A quick view to the smartphone market reveals that the smartphones are counted into billions of devices around the world[1]. This vast number could be made of use by creating crowd-sourcing applications. Co-evaluating the very nature of a smartphone, its portability and continuous presence in user's activities, with the device leveraged capabilities, the smartphones constitute large wireless sensor networks. This forms a large information-gathering network able to provide data that would be otherwise extremely expensive and time consuming to collect. Such an example can be found in [44] where the use of smartphone sensor networks to collect physical and environmental conditions is studied. After the creation of an infrastructure for collecting data from Android-based mobile phones to cloud servers, *Wreck Watch* [44] was created to detect traffic accidents through accelerometer inputs. Upon the detection of an accident, a preselected list of contacts stored in the phone is notified. Meanwhile, the information is probed to other users, in case someone is close to the incident to help, through a web service interface. In this study, a discussion is left open regarding the battery autonomy along with the creation of a middleware to homogenize the application usage across the different mobile operating systems.

2.2.2 Gesture Based Systems

Gestures are an active field of research on the methods for robust gesture recognition, innovative ways of taking advantage of the recognized gestures, new areas of appliance as well as the utilization of new coming hardware to assist the above operations. In the process of describing such systems three questions need to be answered. How are the data captured and through what sensors or hardware types? Where does the data process take place? For what purpose the captured data were used?

A gesture-based system is driven by capturing human physical movements. As described in previous sections, body movement can be adequately detected with the use of cameras and complex data processing techniques. Apart from the implicit movement detection suggested by cameras use, explicit movement detection can take place with the use of sensors on the user's body. One of the variables in such systems is the decision of picking the appropriate hardware device that will contain the necessary MEMS. A Nintendo

[1] Vision Mobile, (2013) "Developer Economics 2013" @ www.DeveloperEconomics.com

Wii controller in Neßelrath et al.[10] was used in combination with a Nunchuk controller summing into two 3-Dimentional Accelerometers able to represent the movement of both of the user's hands. Meanwhile, a more custom and promising approach was proposed by Yuksel and Adali [45] and Park et al. [48]. As suggested, a wearable device will be specially created and wrapped around the wrist of the user. Both projects use a 3D accelerometer module in conjunction with a 3D-gyroscope and a 3D-magnetometer respectively. An intriguing concept, though, was the intention of creating a "*Magic Ring*" [46], also containing accelerometer and magnetometer sensors. In our field of interest and as mentioned before, smartphones have a complete set of MEMS so examples of their usage in capturing motion can be found in a number of projects [40, 47, 49, 50].

Moving on with the data processing, in most of the previously mentioned projects the gesture classification is executed on the mobile phone. Whether the MEMS are embedded on the device or explicitly connected to it (wirelessly or not), the data processing takes place on the mobile phone side. Magic Ring, though, analyses the data on the microcontroller, while in Neßelrath et at. [10] the data from the Wii controller are sent directly to a computer using Bluetooth for further processing.

The gathering of the data and the gesture recognition are followed by the actual end application usage. This can be divided into two distinct categories; the applications where the gesture results are passed to a computer and later on to the appropriate service, and the applications that directly command the selected device. Starting from the later, in [45] an Android phone is connected to a wearable device that provides the sensor input and to a Bluetooth speaker for output. The gestures control the phone, like answering a call without physical touch. Taking the actions to remote objects, E-Gesture [48] and the Magic Ring [46] (Figure 5: E-Gesgture & Magic Ring) use the gestures to control objects like toy-boats, mp3 players, cd radio, and television. For direct Smart Home control, projects like Dita [49] and I'Home [40] were created to assist every day in-home operations. When an application is made to control both the house and on-device applications through gestures, Double-flip [50] is a proposed gesture to be used, without risking overlapping, for switching the control from the smart home to the mobile device and vice versa.

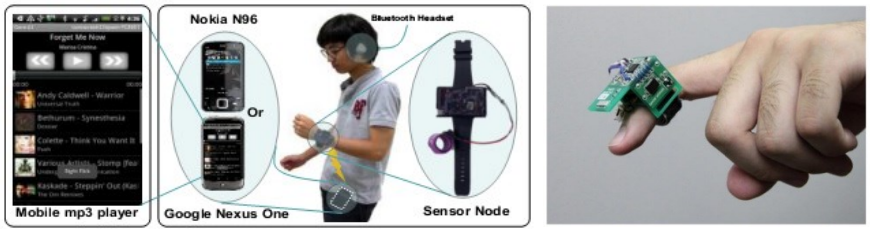

Figure 5: E-Gesgture & Magic Ring

2.2.3 NFC based Systems

Touch is one of our main senses, through which we can feel the actual physical presence of an object. It is the unconscious medium to act upon it. At the same time it constitutes a tangible method of selecting an object that is existent in our environment and within our reach. These natural operations have become a great research subject when it comes to interaction in Ambient Intelligence Environments. The objective is to support analogous operations between the user and the AmIE. That would result in the most natural way of interacting with the surrounding technology with the least cognitive effort.

In this context, Near Field Communication (NFC) technology has been adapted by AmI research and its usage is starting to grow. NFC is a touch-driven protocol for exchanging small amount of information either among supporting devices or among NFC tags and supporting devices. Common uses of this technology can be found in Mobile payment using the mobile phone as an enhanced credit card. Google Wallet[1] is the best representative wrapping credit card credentials under one system making them available through touching the mobile phone on specific readers. Other usages include mobile ticketing, product information systems, location-based information and indoor navigation [60].

The Ambient Intelligence systems that make use of NFC follow the same principle. NFC tags are placed at the areas or on the objects that will be controlled. The user is holding an NFC-enabled mobile phone and taps the desired tag. A mobile phone daemon application identifies the event redirecting it to a computer. The final usage of the captured data is application specific and will be described in the following paragraphs.

[1] Google Wallet @ http://www.google.com/wallet/

An interesting approach was brought by Broll et al. [62] and Hardy et al. [61] where grids of NFC tags we created to form a surface of interaction for projected user interfaces. Imagine a wall on which the user interface of an application is projected. This setup, by itself, does not support touch interaction so the user interface is not directly accessible. The proposed solution is to hang a surface on the wall which will consist of grid of NFC tags closely placed together and project the information on it. The projected user interface design will follow the above grid division. Each user interface element has to be placed exactly where an NFC tag will lie when the UI is projected on the wall. In that way, the user interacts physically with a tile of the NFC grid and this is translated into clicking that area in the UI. If the area corresponds to a user interface element, it generates interaction.

In an effort to extend this approach from one wall to a whole building, the methodology has to change: the tags should be placed on controllable objects with one by one association. Upon touching the tag, the user intuitively selects the associated object to be controlled. For example the user is in the office and is about to light the lamp. Using the mobile phone the user touches the NFC tag associated with the lamp and toggles its state. This selection method is called context switching and it is common in the literature [63, 64, 65, 66]. Furthermore in SeTITaC [63] and easyMeeting [66], NFC tags are used as shortcuts for operations. By the time the user reaches the office and touches the NFC tag placed at the entrance, the system gets updated that the user has arrived along with all the other co-workers. Each user has a tag attached at the office which is used by the colleagues to leave messages in case of absence. Last, NFC tags are placed outside the meeting office each one o trigger a different operation when detected, including booking the meeting room or ordering coffee.

The aforementioned examples illustrate the two main practices regarding the NFC usage in AmIE; either as a way to trigger an action or as a way to associate the user with a place or an object. Both of these practices will be utilized in this thesis.

2.2.4 Toolkits

All the projects that have been reviewed until now share a common misconception; in every implementation of a system that involves the use of a mobile phone there is a mobile application created specifically for that project. Even if same devices and operating systems are used there is no common ground to stand on: this constitutes a great overhead for the already created systems and for the systems to come. For this reason, toolkits are

created to unify the experience. Two examples of such toolkits will be presented in the following paragraphs.

Chronologically starting from the oldest, iStuff Mobile [68] was created as a prototyping toolkit for mobile interaction in ubiquitous computing environments. As a continuation of iStuff toolkit [69] (for Tangible User Interface prototyping), iStuff Mobile preserved the same core application. Event Heap is executed on a computer distributing the received data through a Publish-Subscribe pattern. The distribution paths are set at runtime using a tool named Patch Panel. On the Event Heap embedded sensors and smartphones can get connected thus being able to exchange information. A background application running on the mobile phone is responsible for redirecting commands received from the Event Heap to the foreground application and vice versa. In that way, it is able to utilize external hardware to simulate mobile phones or redirect input from external hardware to the mobile phone expanding its inputs. The final target is to control home appliances connected on the Event Heap. It is worth mentioning that the implementation was made for Symbian OS and the connections between the mobile phone and computer were held through Bluetooth.

The second and most up to date toolkit is Amarino [69]. It is an open-source project for Android phones supporting the interfacing of the phone with Arduino[1] microcontrollers through Bluetooth. The application runs on the phone as a service and can extract all the available sensor data, the phone's state (whether it is ringing or not) and its battery state. Each data source can be separately initiated or stopped either from the Amarino's user interface on the mobile or from the Amarino plugin API that is provided, enabling other applications to utilize Amarino functionality. Any activated data source sends the data to the paired Arduino board. This process allows the externalization of the phone's state and its utilization by the external hardware, whatever that does. Existent projects that use Amarino include AmbientLight, a light emitting heart whose color can be altered through the mobile phone and a wearable device called CallMyShirt, a t-shirt with controllable light pads glowing when a phone call is received or when is running out of battery. Finally, the research project in [45], which is already presented in the Gesture-Based systems section, used Amarino to control a wearable device and initiate calls on the mobile phone.

[1] Arduino main page @ http://www.arduino.cc/

Even though the above are quite impressing, they remain constrained by the operating system and by the specific targeting on Arduino microcontrollers as the receiving end. Due to those limitations, its use in Ambient Intelligence environments is limited to specific hardware types, making it inadequate for general use.

2.3 General Considerations

Most of the reviewed projects share common input. The sources are either provided by smartphones or by external hardware like, for example, prototyping boards. The accelerometer is used in all context aware systems for activity recognition. Meanwhile, gesture based systems utilize it to extract body movement. The accelerometer though is insufficient when it stands on its own, thus more data sources are needed. Such common additional data sources include GPS data, sound capturing, cameras, key-presses, gyroscope sensors and compasses. Software information is also extracted sometimes, depending on the project's objective. The CenceMe [6] and inContexto [4] projects utilize data from Social Networks while ContextLogger2 [2] stores the profile changes, the call status and active application statistics. The variety is wide and the evolvement of smartphones enables even more data sources to be available such as the light sensor, NFC magnetometer and orientation sensor. As Wreck Watch states, the unification of such data with a smartphone can allow previously impossible hardware combinations, opening up unexplored areas of research. As a result, the plethora of exposed data through a smartphone could result to innovative new ideas and prototyping in the future.

Although the data sources can be the same among projects, their handling techniques vary. In context extraction systems, the smartphones are constantly logging information on the device. ContextLogger2 and SurroundSense store the data until a request is made by an operator. CenseMe, inContexto and Wreck Watch use web services to periodically upload the data to a remote server for further processing. In contrast with those systems, a number of projects that were overviewed require instant data transmission. The appliance control in Neßelrath et al. [10] and I'm Home [40], the smartphone control in E-Gesture project [48] and iStuff Mobile toolkit [68], all require the data to be available instantly. Each different approach requires a different application design.

The projects that require live data transmission, such as those that target gesture recognition, use Bluetooth connectivity. Bluetooth is existent in all the mobile devices and

the embedded solutions can be easily extended to support it. For example, in Yuksel and Adali [45] and Park et al. [48] it is used to transmit data from wearable devices to a computer. The E-Gesture project [48], iStuff Mobile toolkit [68] and Amarino toolkit [69] all utilize Bluetooth to connect prototyping boards to smartphones. The Neßelrath et al. [10] follows by connecting a Nintendo Wii Controller to a smart home environment to control the appliances. The use of Bluetooth though, poses a number of issues. Based on its specifications, the maximum radius of reception field is 10 meters in open space. Thus, its usage causes a location restriction. Moreover, its speed is significantly smaller when compared to a WiFi solution. Last but not least, its usage is much more complex than that of the WiFi.

In the past, Bluetooth was used as it was available by most of the devices, though nowadays the existence of WiFi networks in everyday life has become common. WiFi connections are already used by Dita [49] and I'm Home [40] projects in Smart Home environments to control appliances through gestures. Also, recently created applications for large scale user context extraction, such as the inContexto [4] and the Wreck Watch [44], are using web services with HTTP requests over WiFi. The WiFi solves Bluetooth issues of effective range, data transfer speed and complexity. Thus, it is considered as the best approach for connectivity solutions.

All the projects that were reviewed have a fundamental drawback. Their implementation on the smartphone side is specifically created for a specific mobile operating system. Most of the projects that were created before 2010 were implemented for Symbian platform. Symbian operating system is nowadays obsolete and barely used by the public, turning the projects that were specifically developed on it to obsolete as well. Such projects are the ContextLogger2 [2], SurroundSense [5], CenseMe [6], [43] and iStuff mobile toolkit [68]. Modern approaches include the development on Android OS and iOS. The Android platform is nowadays the first choice for development and deployment in research because of its prevalence in the public and openness to the developers. Systems that are developed on Android are Wreck Watch [44], inContexto [4], E-Gesture [48] and Amarino toolkit [69]. Last, the I'm Home [40] project utilized the iOS platform.

The above fact presents the relation of research with the mobile phone market. As soon as the mobile operating system shares change in the future, a number of research projects will become obsolete as well. This statement is already identified in the future work

of Wreck Watch, [43] and iStuff mobile [68], who aimed to port their project to other mobile operating systems. It is interesting to note that Wreck Watch proposed the idea of introducing a middleware that will be running on the mobile phones. Through the middleware the developers will be able to create applications using a higher level language, achieving the desired multiplatform support.

2.4 Privacy concerns

Because of the human-centric nature of the Human-Computer Interaction research field, it becomes clear that important privacy issues arise [1, 11]. The more data we have, the more accurate will be the proactive decision making. But considering that most of the utilized data originate from user actions, activities, sounds, and even feelings, it is obvious that Ambient Intelligent systems potentially exploit personal information [12]. A humorous comment that is worth quoting here is that "*The old sayings that 'the walls have ears' and 'if these walls could talk' have become the disturbing reality.*" [13].

For example, screenshots from the phone's camera were used for ambient color extraction. If those screenshots were captured facing straight ahead, faces from the surrounding crowd could be captured without their prior approval [5]. Small audio recordings can be captured for noise level detection to adapt sound level in Smart Environments. Those sound portions could contain personal conversations or, taking a step further, key tone can be used to intercept passwords [11]. Sensor logging is used in activity recognition [4, 5, 6, 7, 8]. An illegitimate third-party could exploit the logs watching every move of the user.

To provide assurance and reliability on such delicate challenges, policies have to be strictly predefined by the researchers. Policies, in accordance with the privacy preferences of the participating users, should be respected. All the collected data need to be confidential and used only for the purposes of each system.

3 Motivation and Contributions

The burst of smartphones during the past few years has taken place through a raving race of high performance and innovation. The appearance of new and powerful hardware enhancements has made a smartphone analogous to a simple personal computer, using software designed as such to expose the complete device capabilities. By now, it is estimated that billions of smartphones have been integrated into everyday life of people[1]. In this way, each user carries a medium that, if utilised properly, can potentially provide a great range of capabilities.

Meanwhile, the Ambient Intelligence field is evolving through the utilization of technical, or not, resources to assist the achievement of user's aims. The exploitation of the smartphone's capabilities in an Ambient Intelligence Environment could introduce great potentials.

A smartphone can function either as a way to control the surroundings or as the means to make the environment capable to "sense" user's status and intentions. Then, bearing in mind the fact that this device is in the hands of millions of people, it can be easily deducted that its appropriate use will ameliorate significantly trivial everyday operations without any extra cost. This alleviation will be achieved by enhanced functionalities, offered by a device that is already intimate. As a result, even if the proposed new environment is a terra incognita, as the user interacts with an already known interface, the adaptation time is eliminated. Therefore, presuming that in the future more and more places will become abundant in automations and clever solutions, the user will already have the appropriate means to interact with them, irrespective of the purpose of each site. This approach, simplifies not only the utilization of the system for the end user, but also its generation for the developer.

From the programmers' point of view, it can be taken for granted that there already is a combination of functionalities available to the end user. Thus, if this aggregation is properly designed, then the programmers will be able to concentrate even more on other issues, saving a significant amount of time from testing the interaction between human and mobile phones. Of course, a vital prerequisite is the manageable exploitation of the data that are produced from the use of the mobile devices. This is because such a solution should be created as assistance for the programmers and the researchers and not to pose

[1] Vision Mobile, (2013) "Developer Economics 2013" @ www.DeveloperEconomics.com

limitations on the ways that the system interacts with the user. Moreover, satisfying the requirement of rapid prototyping promotes the prompt generation and implementation of new systems, along with the ability to try a variety of different proposals that can be applied instantly. In that way, the research will proceed quickly with immediate results. Last but not least, the necessary facilities for an ambient intelligence environment decrease, accelerating the development process of a system.

In a review of some already existent projects, we have seen many examples that make use of smartphone capabilities or collect sensor data from microcontrollers. During its implementation phase, each project had to implement its own system of collecting the necessary data and make them available to the applications needed. This process was necessary even when same smartphones were used based on the same operating system, or when same hardware was to be utilized collecting the same type of data. This fact imposed the loss of many hours of research by "reinventing the wheel". Additionally, it is highly probable that not all of the implemented solutions for collecting data were feature-rich, something that could have affected the quality of the deliverable system not having taken advantage of all the available functionalities. The answer to this issue is the adoption of a generic toolkit that will extract all the necessary functionality from a mobile device and will provide the required interconnection with the Ambient Intelligence Environment. SmartphoneHarvester is the proposed toolkit by this dissertation, created to fulfill the above statement.

From the users' point of view, the system should;

- Incorporate functionalities that can be easily used
- Be intuitive with low cognitive effort
- Be non-intrusive
- Support multiple platforms to include the biggest market share slice
- Seamless interaction across platforms
- Be easy to install
- Be easy to configure
- Be able to support future platforms

From the developer's point of view, the system should;

- Support complete integration with ambient intelligence environments

- Be easily exploitable by other applications.
- Have comprehensible Application Programming Interface (API)
- Provide abstraction across the different mobile operating systems
- Be compatible with multiple programming languages to cover a respectable range of developers.
- Support functionalities that could satisfy the requirements of different research fields
    - Context Aware systems
    - Gesture-based systems
    - NFC systems
- Introduce the availability of technologies that are existent though remain unused
- Provide remote control of its components
- Be easily extendable for future additions.

The proposed toolkit will be directly connected to a middleware. The iStuff mobile toolkit [68] had proposed the use of a middleware to make the data captured available to other applications. Though in its case, all the sensor data originate from external sensors. The sensors are connected to the middleware and the middleware redirects them to the mobile. Though, the utilized middleware was very primitive, not accessible by any remote application, remaining in the boundaries of one computer. Meanwhile, Amarino [69] extracts data from the device and sends them to Arduino boards[1] via Bluetooth. Amarino is a great toolkit which supports the utilization of many data sources on an Android mobile phone. Even though it is great for rapid prototyping on embedded solutions based on Arduino, the diversity and capabilities of the end applications are limited by the deployment target. If a system could be built which could provide the same data sources like Amarino, and expose them through an enhanced middleware approach like iStuff mobile does, it would be ideal for prototyping in Smart Environments. SmartphoneHarvester toolkit is built upon the above idea. It provides a variety of data and functionality which are accessible by other applications through a middleware.

Moreover, it is crucial the toolkit to have multiplatform support. In the past, each review project had to decide which operating system would the deliverable support. The ContextLogger2 [2], SurroundSense [5], CenseMe [6], [43] and iStuff mobile toolkit [68] had

[1] Arduino main page @ http://www.arduino.cc/

chosen Symbian. On the other hand Wreck Watch [44], inContexto [4], E-Gesture [48] and Amarino toolkit [69] selected the Android platform and finally the I'm Home [40] was developed for iOS. Asa result, there is an undeniable fragmentation on the platform selection which needs to be handled. It is one of the key features that make this toolkit unique. It is implemented once and is successfully deployed on Android, iOS (iPhone), Symbian and Linux.

Additionally, the created system will be able to collect a wide variety of data. The principal goal is to cover as many data sources as possible that will be utilized in Context Aware systems, Gesture-based systems and NFC enabled applications. Each data source will be enabled dynamically, thus it will be using only the requested modules, a fact that promoted the robustness of the system. Unlike that, context logging systems require to access data from the smartphones that include, but are not limited to, GPS data, key-events, profile changes, device's call status, camera images, sound recordings, data from inertial and environmental sensors. Such systems are the ContextLogger2 [2], SurroundSense [5], CenseMe [6], Wreck Watch [44], inContexto [4]. The CenseMe [6] and inContexto [4] projects also exploit data from Social Network Services (SNS), though this approach was not chosen for the moment in our project. The reason that led to this decision is that the exposure of such data to other applications through the middleware would imply violation of users' privacy. The camera usage alike was not included to the list of the toolkit's supported features. The rest of the data sources, including NFC, are intended to be supported as well. A significant touchstone, though, that opens the way for new capabilities, is the support of file system access, which has not been used anywhere in the literature before.

Meanwhile, the gesture-based systems need to be supported. A common approach of the related systems is to support a predefined vocabulary of gestures. This technique is adopted to simplify the recognizers and to be comprehensible by the users. For example, both the I'm Home [40] project and the Neßelrath et al [10] selected the use of predefined gestures to control home appliances. Likewise, the same approach was used in the development of our toolkit to benefit from the simplicity and the low computational overhead that it imposes.

As mentioned before, the aim of the toolkit is to support both context aware systems and real time gesture based systems. Regarding the data transmission, any data that will be collected by the mobile device will be instantly redirected to the middleware. That is because context logging can take place on a remote device once the data are redirected. On the other hand, in case an approach was the data would be logged on the device was chosen; the execution of a gesture would be send to the middleware as soon as the log was uploaded. This would be unacceptable, since the intention of the gestures, among others, is to control appliances in real time.

Furthermore, referring to the communication method, the projects E-Gesture [48], iStuff Mobile toolkit [68] and Amarino toolkit [69] create Bluetooth connections. The I'm Home [40] and Dita [49] utilize WiFi connections whereas inContexto [4] and the Wreck Watch [44] create web services and use HTTP request for infrequent communication. Bluetooth, although existent in most devices, was rejected in our case because of its short range, low data transfer and increased complexity. Also, the HTTP requests to a web service were considered to impose a great overhead since Smartphone Harvester toolkit will need to exchange information at increased data rates and constant connections. Thus, following the I'm Home [40] and Dita [49], WiFi is selected as the communication method.

4 Design Considerations

4.1 Multiplatform support

One of the most important reasons that this project was brought to life, was to provide a generic toolkit of functionality. The word generic can be interpreted in a number of different ways but in this case it refers to the multiplatform support.

Since we are referring to multiplatform support we should first take a look at the current trends in platforms. According the VisionMobile report[1], it has become clear that we live in the era of Android domination, a fact that is illustrated profoundly in Figure 6: Smartphone Statistics. Having taken over more than the half of the market, Android is the indisputable winner. Although it is expanding fast, new, upcoming mobile operating systems, like Ubuntu, Firefox OS and Sailfish, are about to be unveiled and they are possible to change the market share distribution map.

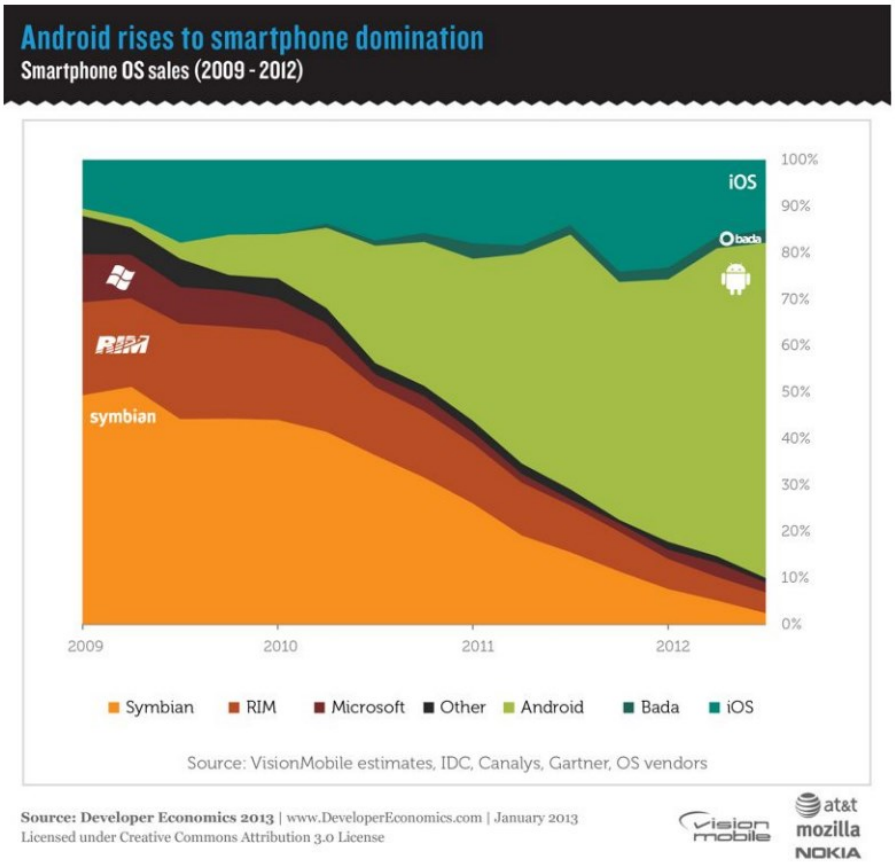

Figure 6: Smartphone Statistics

[1] Vision Mobile, (2013) "Developer Economics 2013" @ www.DeveloperEconomics.com

An obvious to achieve multiplatform support would be to port the same application to all the platforms that we are interested in. It would imply the development of a different application for each operating system. This bears the significant cost of man hours and money, when creating repeatedly the same parts of the code or maintaining similar implementations on different platforms.

The above statements indicate the use of a universal tool, eligible with all the different operating systems, has become a necessity. The list of the existing sensors is long and demanding since among others contains accelerometers, gyroscopes, light sensors, GPS and NFC. Moreover, it is necessary to be able to provide full file system access for read and write operations. The device state has to be available including the active profile, the lock status or the charging status. Being executed in a number of different mobile devices it has to be able to provide basic information about the device such as the operating system type and version. Finally it is a necessity to be able to provide networking capabilities for the interconnection with the outside world. Having the above specifications we were able to narrow down our search to one framework. The list of the rejected frameworks will be presented.

A common starting point is to examine whether the functionality of a single programming language can meet the before mentioned goals, or not. Java, which was the most generic language that run in nearly every platform during the previous decade, is now limited to only a few platforms, with no common APIs across the different operating systems. Likewise, Flash had some access to phone's hardware data but, currently, it is not supported by iOS so it cannot be used. Additionally, although HTML5 was thought to be the trend in multiplatform support nowadays, it does not yet support the entire spectrum of the required capabilities. It is still in the phase of standardization and will be in that phase till the year 2022. Though, undoubtedly, the arrival of Firefox OS, which will be based on HTML5, will push forward the language's advancement, this will be the subject of a future discussion. So, it is clear that a framework with a complete bundle of functionality has to be used instead of a language's base functionality, in order to handle the multiplatform support.

On the other hand, HTML5 is dominant when it comes to the programming language of multiplatform frameworks. That is because a key factor for the success of a mobile application is an innovative and consistent UI. As it is widely adopted for the creation of web pages, the transition of a web page development to a mobile application development is

made smoothly and cost-free. HTML5 along with CSS and JavaScript create the base for many mobile multiplatform frameworks. Such examples include foneFrame[1], widgetbox[2], iBuildApp[3], jQuery Mobile[4] and Sencha Touch[5] with all of them targeting the UI creation rather than the low level control. PhoneGap[6], on the other hand, stands as the most mature and complete framework. In this case, apart from the UI creation, it supports an accelerometer, a compass, GPS, file system access and network access. Moreover, it is open-source and free unlike the most of the aforementioned ones which have to be explicitly purchased and licensed. Unfortunately, regardless its capabilities, it still fails to provide us with all the data described in the requirements.

Moving forward, the aforesaid research on mobile development frameworks led us to the selection of the framework that could meet all the requirements. That is the Qt framework[2]. Qt is an open-source cross-platform framework based on C++. Born in 1992 by Troltech, it got evolved by Nokia from 2006 to 2011 and is now furthered developed by Digia. It was initially used for the creation of graphical user interfaces (GUI), but later on became an all-in-one tool containing nearly every functionality that a developer can request, wrapped in an easy to use and comprehend API. Written in standard C++, it supports most of the existing operating systems using the idea “write once, deploy everywhere”. The operating systems that currently support officially Qt version 5.0.1 are; Windows, Windows CE, Symbian, OS X, X11, Embedded Linux, Maemo and Meego. The Android port, known also as the Necessitas project, was created by third-party developers and is about to be officially merged to the main project. Moreover, iOS is supported through Qt4iOS project but it has been announced that it will be officially supported during the year 2013. In addition to all these, a number of other promising mobile operating systems that were recently announced are created based on Qt and it will be the basic programming interface for them. This includes the newcomer Blackberry 10, the Ubuntu mobile OS and, the barely recognized, Sailfish and Tizen. It is said, but not officially announced, that Qt will support Windows Phone 8 as well.

[1] http://www.qrdvark.com/foneFrame
[2] http://www.widgetbox.com/mobile
[3] http://ibuildapp.com
[4] http://jquerymobile.com
[5] http://www.sencha.com/products/touch
[6] http://phonegap.com
[1] www.qt-project.org

Regarding the capabilities of Qt, they can be separated in core and subsidiary functionalities. The core functionalities include;

- a signal / slot mechanism for raising and handling events in the application and threads, and
- the development of GUIs, based either on widgets or on a declarative CSS-like language named QML (Qt Meta-object Language) that binds together C++, QML and JavaScript for rapid and impressive UI prototyping.

Secondary capabilities include multimedia support, networking, database management and XML handling.

The most important part for this work is the Qt Mobility project, which covers a diverse range of features and technologies used on mobile phones. The Connectivity module is responsible for the NFC usage, while GPS access is provided by the Location API. Sensors are made available by the Sensor API and the device information is extracted through the System Information module. Multimedia module is used to capture audio input.

In conclusion, it can be deducted that both multiplatform support and a diverse range of features are guaranteed, proving that Qt forms a suitable framework to build on.

4.2 Utilized Hardware Technologies

4.2.1 Inertial sensors

Toolkit creation will be dealing a lot with sensors and, for this reason this section will give a brief overview of their functionality. Starting with the definition, the word "*inertia*" is used to describe the tendency of a physical object to resist any change in its state. The term "state" can refer to the object's velocity or momentum. An inertial sensor is the MEMS that are able to detect the above state changes, providing the actual amount of change. In correspondence to the two state types supported, the inertial sensors include the accelerometer and the gyroscope sensor.

An accelerometer is able to detect the changes in the velocity of an object. That applies both when the object is still and when it is moving at a constant speed. For the velocity to be altered, a physical force has to be applied on. In that case, the accelerometer sensor is responsible for measuring the amount of force the object is receiving and its direction in the three-dimensional space (see Figure 7: Sensors 3D Axis). Such sensors, that

are able to detect forces in all 3 dimensions, are constructed by combining 3 different accelerometers in a single chip, one for each Axis. The output of an accelerometer is a triplet representing the direction and the amount of the forces applied to the object at a given time. Each value is measured as a magnitude of gravitational force (*g*) and normally ranges around ±2*g.* That is because the maximum absolute acceleration that can be measured by smartphone sensors is commonly limited to 20 $\frac{m}{s^2}$ and compared to the gravitational force which is 9.81 $\frac{m}{s^2}$ it results to the above.

By definition, the accelerometer measures the forces applied to an object. For the triplet values to be all zero, no forces have to be applied to the object. Practically, this means that the object needs to be in the vacuum or else the gravity of the Earth will be constantly producing values. Since our work here is targeting the mobile phones, it is unlikely that any of us will see a zero triplet. Well unless you are an astronaut or an alien. As a result, the accelerometer is always measuring the gravitational force, whether it is standing still or it is moving. The accelerometer values of a stationary object are a very useful feature in our case, considering that we can detect the orientation of the device. The device orientation describes the side of the phone that is currently facing towards up, or simply said, is on top. The different orientation values are TopUp, TopDown, LeftUp, RightUp, FaceUp and FaceDown, as seen in Figure 8: Device's sides and Figure 9: Device's orientation values).

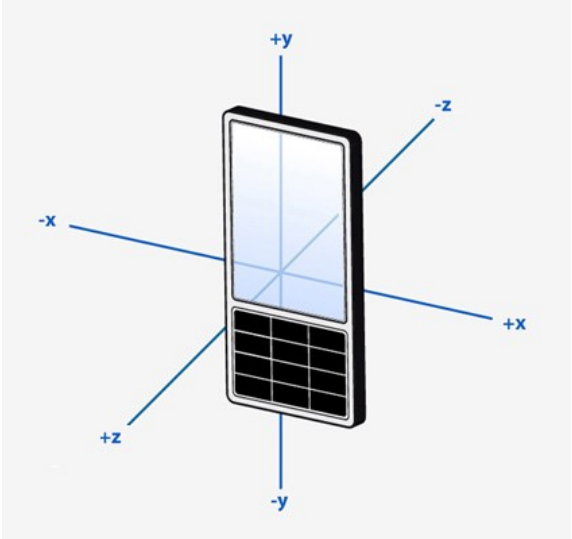

Figure 7: Sensors 3D Axis (from qt-project.org)

Figure 8: Device's sides (from qt-project.org)

Figure 9: Device's orientation values (from qt-project.org)

A gyroscope sensor measures the object's angular change rate. This means that it measures the angular acceleration on each movement of the object. Like the accelerometer, it consists of 3 gyroscopes in a chip, one for each axis of rotation in the 3D space. The axes, along with their orientation, can be seen in Figure 10: 3D rotation Axis. It also produces triplets of data where each value, states the angular acceleration in each axis measured in$\frac{d\varphi}{s^2}$. The φ value is measured in degrees. Finally, unlike accelerometer, a triplet can have zero values when the phone stands still or when its movement is parallel to all three axes at the same time.

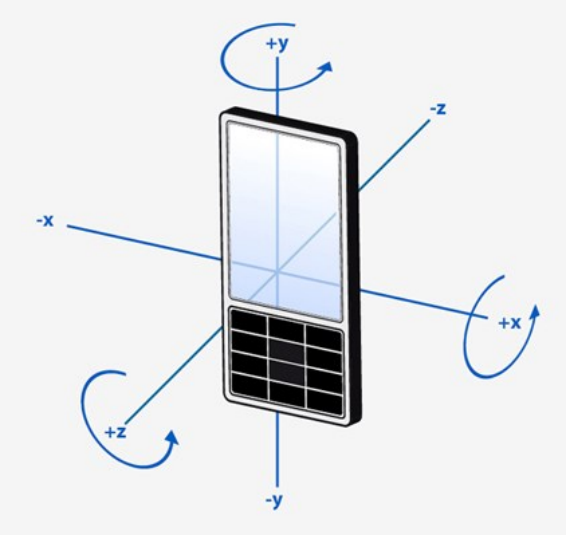

Figure 10: 3D rotation Axis (from qt-project.org)

4.2.2 NFC

Near Filed Communication (NFC) is a wireless communication method defined by a set of protocols for exchanging small amount of data in extremely close distances. Those protocols are defined by ISO/IEC 18092 standard and comply also with ISO 18000-3 that defines the RFID (Radio Frequency Identification). According to the latter, the maximum transfer rate is set to 424kbit/s and the maximum theoretical range is 20cm, whereas in practice it is 4cm.

Every operation executed through the NFC has an originator and a target. Assuming that the originator is a smartphone, or any other NFC-enabled device, the target can either be another smartphone or an NFC tag. Those two options reflect the two operation modes of the NFC protocol. The active communication mode is when the target is another device. In that case, both the originator and the target are generating their own field of reception, which indicates that both have a power supply. Hence, the connection is point-to-point and when any device is receiving information it temporarily deactivates its NFC field. The interesting part comes when using NFC compliant tags which constitute the passive communication type. When the originator gets close to the tag, the electromagnetic field, generated by the NFC reception field of the originator, produces the operational power for the tag thus fixing the lack of power supply. This inductive electrical power transfer is the reason why the maximum distance is set to 4cm since it complies with the Qi standard of Wireless Power Consortium. A great advantage, when compared to the RFID, is that the NFC tags, apart from the read operation, support write operation as well. The NFC tags support also a one-time locking of their data after which they are made permanently read-only.

Moreover, the maximum storage and the maximum speed vary depending on the tag type (see Table 1: NFC tag types).

Tag Type	Standard	Capacity	Max. speed
Type 1	ISO14443A	96 Bytes (2kByte)	106 kbit/s
Type 2	ISO14443A	48 Bytes (2kByte)	106 kbit/s
Type 3	Sony FeliCa system	2kByte	212 kbit/s
Type 4	ISO14443A and B	Up to 32kByte	between 106 kbit/s and 424 kbit/s

Table 1: NFC tag types

Any data transmitted through NFC is contained in an NDEF message. This abbreviation stands for the NFC Data Exchange Format which is a binary format for data encapsulation and is defined by the NFC Forum. Each NDEF message contains one or more records of information. Each record is constructed by the NFC Record Type Definition (RTD) in combination with one of the four specific RTDs: Text, URI, Smart Poster and Generic Control. These create the basic structure of the exchanged information so, the appropriate format of the message is selected according to the contents that are to be sent. For a more in-depth analysis of the above, it is advised to visit the NFC Forum Technical Specification's page [@http://www.nfc-forum.org/specs/spec_list/].

4.3 AmIE Integration

Our goal is to create a toolkit that exposes smartphone's data into an AmI environment. A high priority concern is how to expose those data in such an environment. This will be answered by examining the way AmI systems are designed and built. By definition, they consist of many smaller applications that run simultaneously, necessarily communicating with each other. This is the basic concept of Ubiquitous computing after all. The simplest approach for data exchange would be to create direct connections between the concerned parties. This would imply a great effort overhead because each application would have to create and handle its own connections with the external applications. Moreover, the implementation of the external applications should be affected each time a new demand arouses. It is obvious that this solution is not viable nor can it support large scale systems.

A common solution to the above problem is the use of a middleware. A middleware is responsible for creating communication channels between the applications that use it.

Based on a Service Oriented Architecture it provides the ability of creating and using services. A service is a functionality package provided by a separate application and its functionality is accessible to any system that complies with the middleware standards. The functionality exposed by the service is described in a consistent and technical manner by a dedicated language whose syntax varies, depending on the middleware implementation. This description language is identifiable regardless of the actual programming language used to implement each particular application. Thus, a consistent and implementation independent vocabulary is defined, through which the applications can exchange and comprehend information or commands.

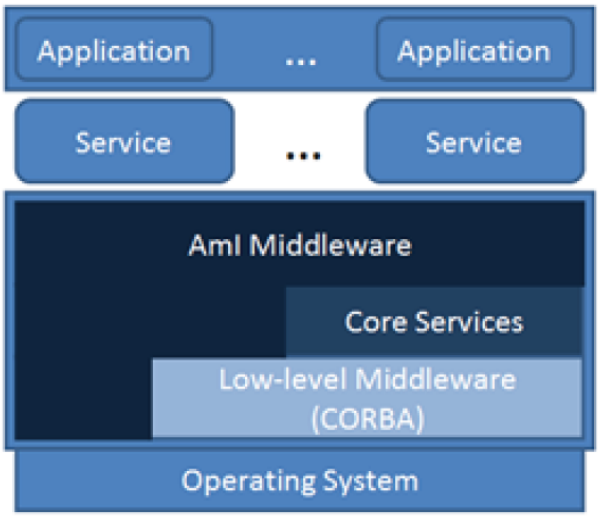

Figure 11: Famine middleware architecture

Such a middleware is implemented in our facility of Human-Computer Interaction lab in the Institute of Computer Science of the Foundation of Research and Technology - Hellas (FORTH). This middleware named Famine [70] (Figure 11: Famine middleware architecture) complies with the above definition and is implemented based on CORBA standard for multilingualism and multiplatform support. It allows the creation of services by using multiple programming languages such as C++, .NET, Java, Flash and Python. As defined by CORBA, the services are described by the Interface Definition Language (IDL). Considering the predefined requirements for general interconnection among the participating applications, the Famine middleware was ideal to build on.

4.3.1 Selected Design Approach

In this section, first, a description of the basic principles of a Service Oriented system is introduced, followed by their affect on the system design. Last will be described the architecture of the final system.

Each Service-Oriented System has two fundamental procedures that have to be executed as defined by the theory. First step is to make a service available to public and second to allow client programs to use the service. Starting with the first step, we will be referring to this procedure as the registration of the service. For an already implemented service to be made available it first has to define its interface, which will be exposed. That is, to define all the supported functions and its produced events through the aforementioned Interface Description Language (IDL). This interface will be uploaded to a Service Repository that is accessible to the clients. The actual publishing of the service is made through its registration. During the registration, the service communicates with the Naming Server of the middleware where the identification name, also known as context, will be assigned to that instance of the service. Hence, many instances of the same service can run simultaneously avoiding service overlapping or conflicts from the use of different contexts. The context can either be explicitly provided by the service, when requesting the registration, or automatically assigned by the Naming Server. When this step is completed the service is informed for the successful registration and it is made available. From that time and on, any client can use the service.

The procedures followed by a client to make use of a registered service start by obtaining the service interface. When executed, any requested service has to be resolved. This involves communication with the Naming Server of the middleware. The request from the client contains the name of the service and the context at which the service has been registered. If such a pair exists, the client gets directly connected with the service instance. Consequently, the functionality of the service can be used by the client through immediate requests to the service instance.

In continuance, applying to our system the theory described previously the aim is to expose the capabilities of the mobile phone as a service. Our goal is to expose the capabilities of the mobile phone as a service. This can only be done by making a direct connection between the mobile phone and the middleware. Unfortunately, this is impossible due to technical reasons, since Famine is supported only for Windows operating system. Bearing that in mind, it became clear from the beginning that a proxy had to be

created. The proxy will be placed between the mobile phone and the middleware which will be divided into two modules. The first module would be the mobile device proxy handling all the interconnection with the mobile phone. Internally it will be communicating with the service implementation module. The service implementation module will be responsible for translating the received message from the mobile phone through the proxy, to service responses or events. Moreover it will be able to place requests to the proxy, which, subsequently, will be redirected to the mobile phone, either waiting for a response or not. The architecture is depicted in Figure 12: Initial Smartphone - Proxy architecture.

The initial design of the system was to have only one service instance registered through which the clients could be given access to all the connected mobile devices. It would be ideal to be able to select through a list of connected devices, access the events produced by the selected devices and control them. Unfortunately, this design was rejected due to technical reasons. The problem lies in the way the clients communicate with the service. It is based on a mix of publish-subscribe pattern and point – to – point communication. The point – to – point connection is used for requests that originate from the client and if any data are returned, they are sent to the originating client. On the other hand, any event produced by the service and handled by the clients is based on the publish – subscribe model. The service is the publisher sending the events and the clients can be subscribed to receive the events. When an event is sent out by the service, no receiver can specified. All the clients that are bound to the service are automatically subscribed, so they receive the event whether they use it or not. For example, take the case of 10 connected mobile devices, all sending accelerometer data at the rate of 30 samples per second. In this scenario, a client wants to read the data of a single device. It would be impossible to do it without receiving the data from all the other devices at the same time. Also, the client would need to process all the messages and keep only those of interest. This forms a great overhead that is growing linearly depending on the number of the connected devices. For this reason, it was decided to create one instance of the service per mobile device which will be having as a context an identification name related to the mobile phone.

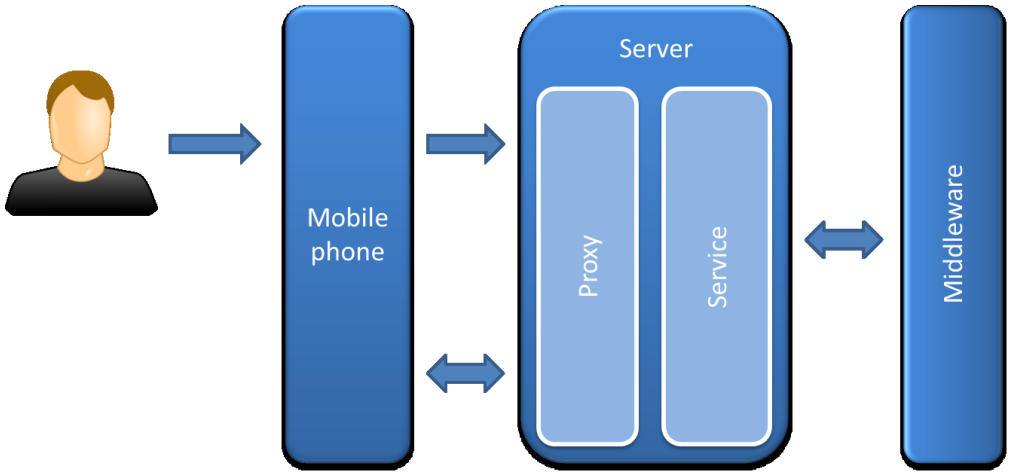

Figure 12: Initial Smartphone - Proxy architecture

Two main drawbacks can be identified in this architecture. Even though it is functional for one mobile device, the addition of a second one to the system cannot be done automatically. A new instance of the server has to be created. The server is an executable piece of software that runs manually by a user. In this way, if a new server is needed it has to be created manually which is not a viable solution. Additionally, the current setup does not support the recovery of the connected devices list. In that way, the clients would need to have hardcoded contexts of the services that are associated with the mobile devices needed to be used. These two restrictions were definitely affecting the scalability of the system so the architecture had to be extended to handle those issues.

The solution to the issue of the automatic handling of the new devices was brought by making the server element a reusable component. Actually, the pair of Proxy – Service was made a component which can be instantiated in the server as many times as needed. The Server remains the base of the application on which new devices will be able to connect. Upon connection of a remote device a pair of a Proxy and a Service instance is created. The Server handles over the active connection to the Proxy making it responsible for the data transmission and retrieval from the mobile device. When the handover is completed the service gets registered to the middleware. The context chosen for each new service is formed by the device information data that are retrieved during the socket handover. Thus, it is unique. In this way there is no limit to the number of simultaneously connected mobile devices since each one gets handled and executed in its own “sandbox”. Upon device disconnection the Proxy – Service pair is destroyed along with the service unregistering, keeping the system clean and consistent.

The next handled issue was the inability of the clients to retrieve the list of the connected devices. The solution of the automatic connection of new devices would be useless if there wouldn't be provided a way for the clients to query for the list of the connected devices. This was solved by the creation of a higher level service, at which we will be referencing as the Broker. The Broker's functionality is to be able to respond at requests of listing the connected devices and raise an event whenever a new device gets successfully connected. The Broker was created as a singleton part of the Server application described earlier, utilizing its awareness of new coming connections. When the Server creates a new connection with a new mobile device, it executes the steps described in the previous paragraph. As soon as the service instance of the mobile device gets successfully registered to the middleware the Server informs the Broker that the service got registered with a specific context. At that time, the Broker raises an event to report the above fact containing the newly registered context. Thus, any client that has already resolved Broker service gets informed about the newly connected device. If the client gets started after the emitting of the event it can retrieve explicitly the list of the connected devices again by the Broker service. Finally if the device gets disconnected, the Broker raises another event, which contains the context of the service, to report the disconnection. The updated architecture is depicted in Figure 13: Final Smartphone - Proxy - Broker architecture.

Regarding the final architecture of the system, some results and considerations can be expressed. With the above design, any client that will use our system is advised to first resolve the Broker service, even if the context of the mobile device's service is already known from a previous execution. Due to the fully dynamic design of the system, a mobile phone can get connected or disconnected arbitrarily. For this reason, any client must always be aware of the state of the service or the services in use, by utilizing the Broker. It will avoid issues when the client is requesting data from an already disconnected service where there will be getting no answer

Regarding the Broker, as it is a service as well, it has a context by which it gets registered upon the creation of a Server and unregistered at the Server's destruction. Having a one – to – one association with the Server, follows the Server's limitless feature. If there can be limitless Servers running simultaneously, then no boundary will be put on the number of Brokers, assuming that all of them will provide a unique context for each instance. The advantage of having a number of Brokers running, which means a number of different Servers as well, is that the mobile devices can be divided into groups, with each

group be represented by one Broker. For example, a Server can be started with its Broker registered by the context "*office*". Those devices that will be used in office operations will be contacting the above Server to be their entry point to our system. Similarly, the devices targeting Smart Home control operations will be contacting the Broker registered after "home" context etc.

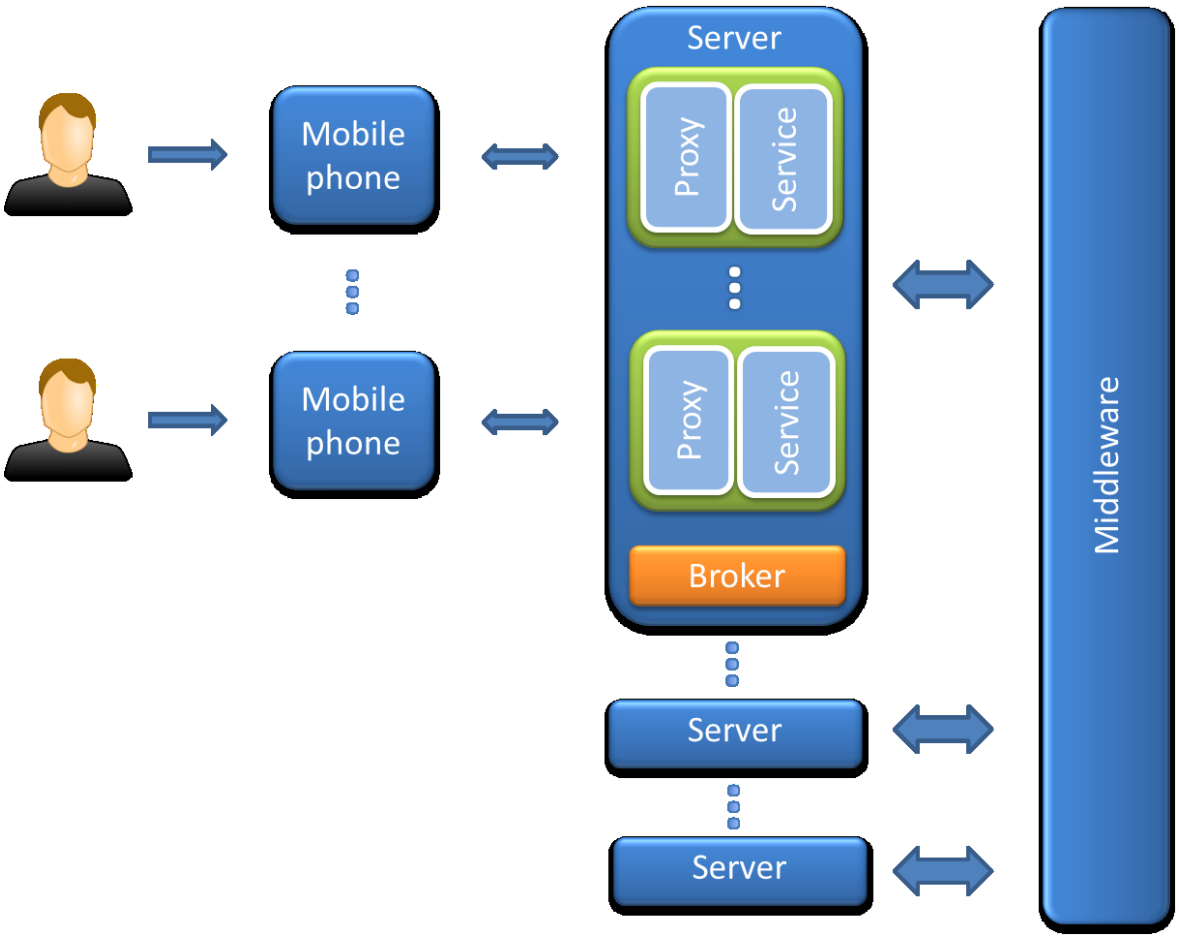

Figure 13: Final Smartphone - Proxy - Broker architecture

.

Since we have an overview of the system's architecture we can continue with its actual implementation. We will start by describing the communication protocol between the mobile device and the proxy in section 5.1. Then, in section 5.2, we will analyze the implementation of the SmartphoneHarvester mobile application with all its components and its supporting operations. Last will take place a description of the implementation details of the Server and the Broker.

5 Harvesting & Exposing Data

The previous section dealt with the analysis of the design considerations. The framework to build on, along with the main architecture of the system was also defined. Having prepared the base of our system, in this section we will start building the actual implementation by explaining the components that form it. The definition of the communication protocol between the mobile application and the service will format the information exchange in section 5.1, followed by the analysis of the Smartphone Harvester mobile application in section 5.2, its capabilities, its components and the message handling using the communication protocol. The implementation is completed by presenting the SmartphoneHarvester Middleware Proxy that contains the Broker service, the connection server and the mobile device handler in section 5.3.

5.1 Information exchange guidelines

Our system is based on the information exchange between mobile devices and computer applications. In this way, designing the system, it was crucial to take the correct decisions about the type of connection through which the data would be transferred. After the transmission, a consistent and universal data format had to be standardized for the participating parties to have a common vocabulary.

5.1.1 Connection Type

This thesis deals with mobile phones and mobile devices in general. The word "mobile" is dominant. Translating this in the context of the connection type, it makes it straightforward that the connection type has to be able to go along with the mobile nature of the device. Thus, wire connectivity by definition is ruled out. Having to choose among the different types of wireless connectivity, it was first essential to go through the supported wireless connection types by the mobile devices.

The requirements of the connection type are simple; it needs to be

- available in all mobile devices,
- easy to use with connections that are reliable and assure the integrity of the transferred data,
- small connection establishment time overhead and eligible with byte streams

The two technologies that resemble the above requirements are Bluetooth and Wi-Fi. On the one hand, Bluetooth has been extensively used in the research, though it does suffer from a number of issues. The most obvious is the range of the reception. Assuming

that class 2 Bluetooth dongles would be used, it implies that the maximum radius of reception field would be at most 10 meters in open space. Considering that we are at indoor spaces, walls and many other obstacles are included, affecting the diameter of the reception field. Even for a medium-sized house, more than one Bluetooth server would be required. This would demand time consuming operations such as disconnections from one server, connections to the other and handing over the device's data from one server to the other. This approach, for a real time system like ours, would result in time periods of unavailability. To explain it further, consider an example where accelerometer data are constantly received from a mobile device. In this case, the system may be receiving 30 samples per second. A service unavailability of 5 seconds could result in the loss of a gesture command that should be executed in that period of time. Another impediment is the limited bandwidth of Bluetooth. Bluetooth's version 2.1 with Enhanced Data Rate (EDR) achieves a max of 2.1 Mbit/s and version 3.0 with High Speed (HS) support achieves 24Mbit/s but not over Bluetooth link. A supplementary 802.11 connection is used for data transfer with Bluetooth handling only the handshake. Since Bluetooth v3.0 with HS is not widely supported, we keep only the 802.11 part of the connection which means that the selected wireless connection for this project is Wi-Fi.

From the early beginning, Wi-Fi seemed to be the best solution. First of all, almost every house is equipped with a Wi-Fi router able to cover all the house area. Secondly, nowadays all mobile phones support Wi-Fi including the low-end series. Moreover, company facilities are common to have large grids of wireless routers covering all the company's spaces under the same network. In those latter cases, the switching between the routers will also require another connection like Bluetooth, but we will explain how it was solved in the most efficient way in section 5.2.8. Summing up, WiFi provides networking with a large effective range of reception supporting a maximum data transfer rate of 24Mbit/s. Based on the WiFi connections, next step was to make a decision regarding the communication protocol that would be used throughout the connections. The three main options were the UDP, TCP/IP and HTTP.

In the beginning, the idea of using UDP seemed ideal. A connectionless bi-directional communication would be a great advantage in cases of frequent network unavailability. There would not be any overhead of reestablishing the connection. This approach, though, had two disadvantages. First, there is no way of detecting whether there is any receiver alive, thus there is no way of consistently knowing which mobile devices are connected. In

cases that the receiver is started or terminated normally, the receiver could send a state message to the other end reporting either that it is alive or that it is about to be terminated. On the other hand, in case the receiver is unexpectedly terminated, there is no way for the sender to detect the state of the receiver, unless timeouts are used, which is only a hypothesis of unavailability. In addition to this, the most important disadvantage is that it is not trustworthy, regarding the integrity of the data exchanged through it. There is no assurance that the data sent will be delivered with no data loss. This fact can be overlooked when sending small length data multiple times per second. The corruption of one message does not have any impact on the outcome. However, the proposed toolkit will support file transferring. In that case, the loss of one byte would result to file corruption which cannot be bypassed.

Having in mind the previous analysis, TCP/IP was selected. This protocol is connection-oriented creating one socket per connection between two ends. If any of them is terminated gracefully, or not, the socket gets closed informing the other party for the disconnection. In this way it guarantees the consistency of the communication. Moreover the protocol itself assures the data integrity. In case any transmission error takes place, it is handled by communicating with the originator of the message for retransmission.

5.1.2 Exchanged data format

As discussed in the previous section, the connection type will be TCP/IP over Wi-Fi so next step is to standardize a common message format. With the use of TCP/IP, arouse the idea of using HTTP requests over it. The Proxy would be the HTTP server and the mobile device the client. This architecture would benefit the mobile device when sending data or files to the proxy but the server would not be able to send requests to the client. In order to overcome this issue, there was the option that both the mobile device and the Proxy run on a separate HTTP server. Though, the idea was rejected due to its complexity and the maintenance overhead that it introduced. As a result, we were lead to simple TCP/IP sockets with raw data streams.

The utilization of a data stream indicates that byte arrays are sent over the socket. Regardless of the message or its contents, every data has to be translated into byte arrays or, else named, raw data. These are the data that will be transmitted and received by the other end. The problem is that the receiver by default cannot identify the beginning or the ending of a message. It receives a stream of data and it is responsible to divide and decode it

in order to extract the messages contained. The sender is responsible for creating the message to be transmitted in the correct format for the sender to be able to decode it. It is a two-way agreement. The format that was chosen for our system will be explained in the rest of the section.

The requirement was to find a way to form the messages, maintaining the balance between the low formatting overhead and the readability. The formatting overhead is the extra information added to the initial message in order to maintain the formatting. It would be ideal to use a format that already exists, so as to be compatible with other applications if needed in the future. JSON (JavaScript Object Notation) standard was finally chosen for our case. It is a text-based open standard for human-readable data exchange. It is a solution widely adopted by thousands of systems, mostly web-based. Its best feature is that it can provide Object-Oriented structure to the data. The basic structure of a JSON object is provided in Figure 14: Example JSON object. As the example from Figure 14: Example JSON object indicates, a JSON object is a string-indexed array of data like a Hash Table. Its supported values include all the basic types of variables, another JSON object or a list of other JSON objects. This gives us as much flexibility as needed, as it is not only easily readable by humans, but also an open standard leveraging its compatibility with other systems.

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumber": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

Figure 14: Example JSON object

This example forms an opportunity to explain the formatting overhead. The overhead is defined as the length of a message in bytes, once the sum of all the length of all the values is subtracted. In this particular example the overall length of the message is 410 bytes and the sum of the length of the values is 70 bytes. This results in an 83% of formatting overhead, which means 340 bytes. Those, are broken down to 90 bytes (26%) that form the sum of all the length of the keys, 173 bytes (51%) of white spaces and the rest 77 bytes (23%) that are the necessary quotation marks, commas etc. In our case to achieve the lowest percentage of formatting overhead it was decided to remove any white space, except for those contained in the values, of course. Also for the commands that are sent frequently, like the accelerometer's readings, the length of each key was set to one. Hence, "x", "y", "z" is quite adequate. Also, in the frequently used commands there is no use of complex values such as the arrays or other JSON objects. If those changes are applied to the above example, the overall key length is dropped down to 13 bytes, since there are only 13 keys and the white spaces are completely removed. So, the new overall length of the message, after the proposed changes, is 160 bytes, significantly smaller than the initial of 410 bytes. The new message includes 90 bytes of content data that form the 56% of the overall message length and the formatting overhead is 44%. In fact, this is an important decrement and will be even more obvious, since, as mentioned before; there will not be so many fields of values per message. Consequently the values themselves will have as small length as possible shrinking the overall size of the message to be sent. After the presentation of all the proposed contents of a JSON object formed by the message, there will be an analysis of the formatting overhead through the structure of a common case message.

5.1.3 Message contents

The goal of the proposed message format is to keep the length of the value field as small as possible. The use of enumerations, instead of actual data like strings, was the best practice. Six types of enumerations were defined to encapsulate a great amount of data in just a sequence of numbers. Hence, only the parties that have the enumeration definitions are able to decode the actual meaning of each message. Since our data will not be encrypted for the sake of speed, the use of enumerations can at some level obfuscate the message, making it hard for eavesdroppers to understand the sniffed messages. The six enumeration types are listed in Figure 15: Proposed Protocol enumerations providing an overview of all the supported commands.

A quick overview of the enumerations makes it clear that the enumerated data are not the actual data collected from the device's sensors or any device info. They are the data contained in each message, used to indicate the type of the message and its expected contents. When the enumeration values of the message are combined together, they reveal the purpose of the message and the data source of the content. Thus, the receiver identifies the expected data fields that are contained in the message. For example, any new sensor value is reported through a message where the *Operation* is equal to *StateUpdate* and the *ContentType* equals to the value of the originating sensor (ex. The accelerometer would be *AccelReading*) followed by the sensor data as the content of the message.

```
enum Operation{
    Request = 0,
    Response,
    StateUpdate
};

enum RequestType{
    SensorSelection,
    GestureSelection,
    HardwareInfo,
    FolderBrowsing,
    FileRetrieval,
    FileUpload,
    SoundControl,
    NfcMessageWrite,
    FileOpen,
    ReestablishSocket,
    FileDelete,
    NfcMessageWriteToServer
};

enum Sensor{
    Accelerometer,
    Orientation,
    Proximity,
    Tap,
    AmbientLight,
    Compass,
    Gyroscope,
    Light,
    Magnetometer,
    Rotation,
    Gps,
    Nfc
};
```

```
enum ContentType{
    DeviceLockStatus = 0,
    ProfileStatus,
    AccelReading,
    OrientationReading,
    ProximityReading,
    TapReading,
    AmbLightReading,
    CompassReading,
    GyroReading,
    LightReading,
    MagnetometerReading,
    RotationReading,
    GestureRecogition,
    NfcDetected,
    NfcLost,
    NfcMessageReceived,
    PositionUpdated,
    ChargingStatus,
    SoundStatus
};

enum Gesture{
    Cover,
    DoubleTap,
    Hover,
    PickUp,
    Shake,
    Swipe,
    TurnOver,
    Twist
};

enum SensitivityLevel{
    UnusedSensibility = 0,
    High = 1,
    Medium = 2,
    Low = 6
};
```

Figure 15: Proposed Protocol enumerations

When designing this format, an important issue that came up was how to categorize the collected data in order to minimize the number of fields included in the content part in each message. For example, the accelerometer needs 3 integers to provide the values of the

3 axis, the NFC needs a string value to report the ID of the tag and the proximity sensor requires a boolean value to state if something is within range or not. This diversity led us to enlist all the different data sources and commands that we were interested in, along with the type of the data that each one required. The above enlistment was helped to notice that many commands share common data fields. The outcome was the definition of 8 content fields for the most commonly used commands; one Boolean field (“b”), two String fields (“s”, “ss”), one Enumeration field (“e”) and four Real fields (“i”, “x”, “y”, “z”). As far as only the commonly used commands are important, from the performance perspective, the rest infrequent commands can be greedy with their content key lengths and will be allowed to support lists or other JSON objects as values. Note that any of the proposed fields that is unnecessary for a given command formation, is not added to the JSON object, keeping the message clean and as small as possible. Also, in practice no more than 3 of the above fields are used at the same time. The complete structure of the proposed message format is depicted in Figure 16: Proposed message format.

The previous description of the structure of a common case message, will be used as a reference to calculate the minimum and maximum length and the formatting overhead of a common case message. The minimum message does not contain any contents apart from the Operation and the Content Type or Request Type. That is a JSON object with two fields like the following *{“o”:1,”r”:2}* whose overall length is 13 bytes. The maximum common case message is created by sensors that provide three floating point values one for each axis. Each floating point value’s length is at most 64bit (8Byte) so the JSON object would be formed as follows *{“o”:2, “c”:10,”x”: (floating point value), “y”: (floating point value), “z”: (floating point value)}*. In the worst case scenario where all three floating point values occupy the complete 64bit range each, the overall message length is 53 bytes from which 21 bytes (40%) are the formatting overhead. Even though 40% overhead can be considered significant, the overall length of the message is so small that even when all the sensors are set to the maximum sensitivity, sending 30 samples per second per sensor, the maximum bandwidth required is less than 20Kbyte/s.

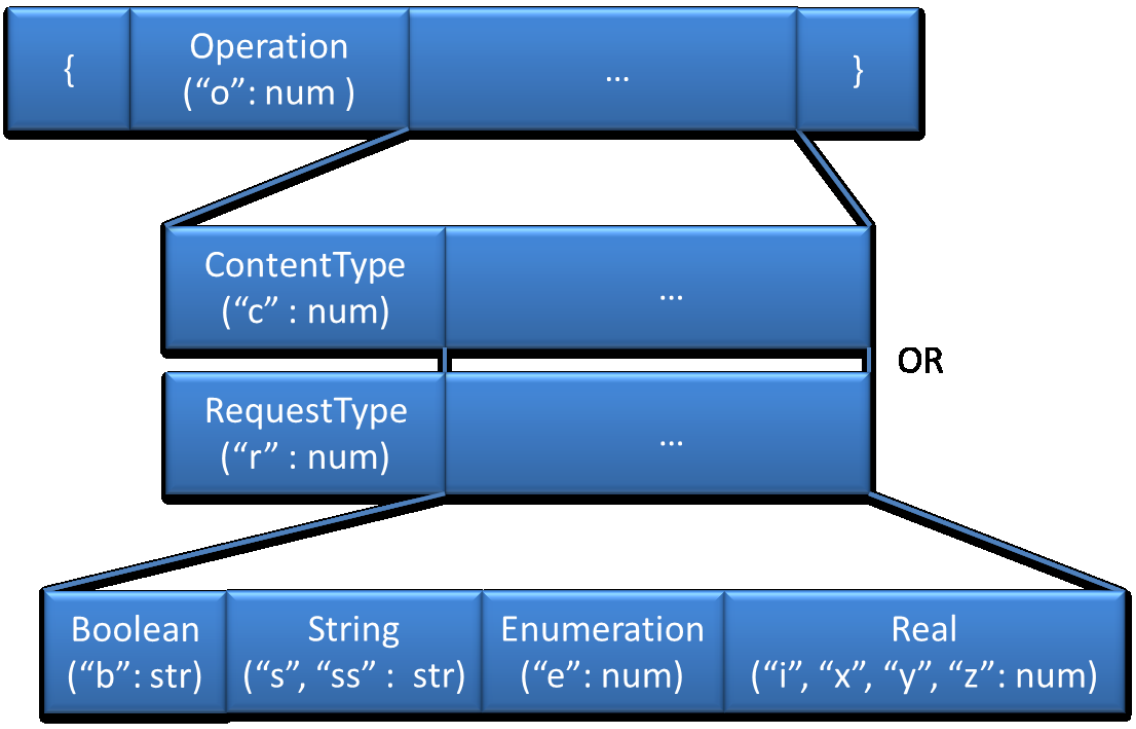

Figure 16: Proposed message format

5.2 Smartphone Harvester mobile application

In this section the SmartphoneHarvester mobile application will be presented. It is the starting point of all the work done, considering that it is the tool that provides all the functionality at the mobile device side. It is installed and executed at the remote device and is always connected with a dedicated Proxy that runs on a computer. It continuously exposes the collected data and it is always available to receive requests from the Proxy it is connected with. To achieve continuous availability, it is designed to operate even when the application is at the background. Event though, it can be used as a front-end application. If the operating system does not support multithreading, the application is used like any other frond-end application. Moreover, if there is no screen available it gets executed as a console application excluding the graphical interface.

Moving on to the application architecture and its integral parts, SmartphoneHarvester architecture can be divided into 3 main modules. The first one is the Socket Handler module to provide the interconnectivity with the proxy. The Graphical User Interface is the second, exposing only limited and necessary functionality to the mobile device owner. Last is the Core module that comes to bind the above two together. The Core module is the most important one, as it is a unification of many distinct sub-modules each

one responsible for a different functionality of the mobile device. A quick enlisting of those modules starts with the Sensor Manager, which is a low level layer to encapsulate sensor control and new value retrieval. Those functionalities are exposed either directly to the SmartphoneHarvester Core, to support raw sensor values reporting, or, indirectly, to the enabled Gesture Recognizers. Gesture Recognizers get enabled or disabled by the Gesture Manager. Continuing, the Location Manager exposes GPS data and the File System Manager provides access to the file system both for read and for write operations. These two operations are also shared by the NFC manager using NFC technology by reading from and writing to NFC tags or other NFC-enabled devices. Next, the Hardware & Device info manager was created to extract information regarding the device itself for identification or context extraction. Moreover, the Audio Manager captures the microphone input to extract the noise level and finally the Command Executer module supports the execution of a small vocabulary of commands on the mobile device. All the aforementioned Core's modules are orchestrated by the main Core module through which all the exposed data, gathered by the above Managers, are translated into transmittable objects/messages, using the protocol described in section 5.1, and are, later on, sent to the Proxy. This architecture is pictured in Figure 17: SmartphoneHarvester mobile application architecture. Each different part will be analyzed along with the connections with all the other parts, to present how all the modules create one unity, the SmartphoneHarvester.

5.2.1 Core

The in-depth analysis of the SmartphoneHarvester application will start by presenting its core, more specifically the orchestrator module, which, from now, on will be addressed as the Core. It is the module responsible for the interconnection of all the other modules. It is the first one that gets created when the application gets executed, instantiating all the other modules, thus building everything around it. Every sub-module that is instantiated gets directly connected only with the Core. In this way, every module isolates its specific operations reporting only to the Core. With this practice, the system complexity is reduced, increasing the system's scalability and maintainability. The communication of the sub-modules with the Core is bi-directional. The sub-modules send information updates to the Core while the Core is able to place requests at the sub-modules.

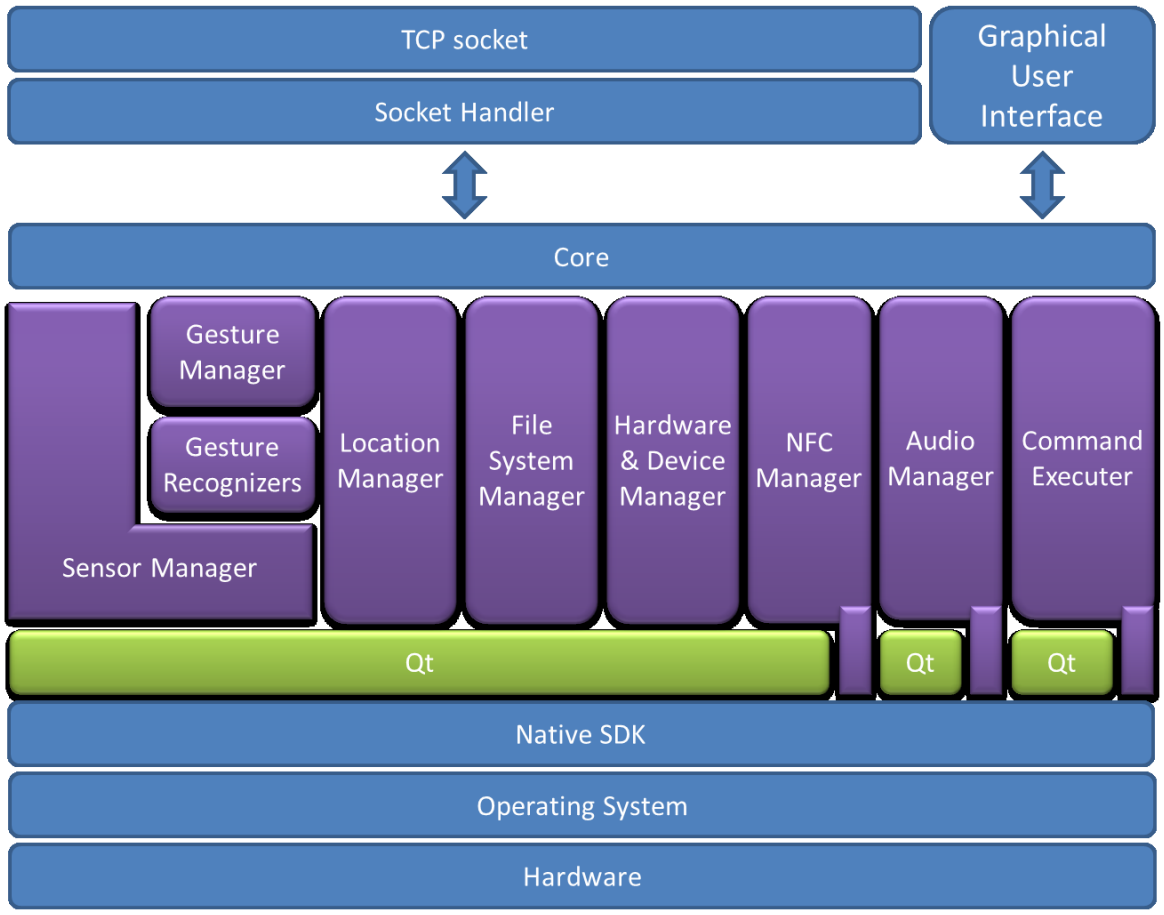

Figure 17: SmartphoneHarvester mobile application architecture

Any data that are extracted by the sub-modules are delivered to the Core. If they are not handled immediately, they get vanished. No logging of data takes place because the system was designed as a real-time reporting system. Thus, it is mandatory to transmit the captured data to an external application. The Socket Handler is used by the Core to be the medium for sending the created messages to the Proxy. Any state report triggered by a value change from a sensor, is delivered to the Core and packed as a message. This message includes the Operation field which equals to the StateReport and the ContentType field. The ContentType field is selected according to the specific sensor that has generated the data. The initialization of the message is succeeded by its transmission through the Socket Handler.

The above procedure takes place between the Core and the Proxy and is bi-directional. The Proxy can send a message whose OperationType equals to Request. According to the RequestType, the Core either acts without response or replies with a message whose OperationType is Response. The response message contains the same

RequestType value as the request message, along with the contents that need to be sent to the Proxy. For example, the Proxy sends a request to retrieve the browsing result from a folder in the mobile device. This message would have the OperationType set to Request, the RequestType to FolderBrowsing and will include as content a string with the path of the folder. This message will be received by the Core and will be routed to the File System Manager. When the File System Manager has the list of the browsing results it will be returned to the Core. Then, the Core will form a message whose OperationType is Response, including the RequestType set to FolderBrowsing and the content with a list of strings for the browsing results.

By now the overview of the information flow through the system should be clear. Thus, in the following sections we will proceed by presenting one by one the sub-modules that get connected with the Core.

5.2.2 Socket Handler

Before proceeding with the Core's internal components, it is important to describe the module through which the Core communicates with the Proxy. That is the Socket Handler module which encapsulates the TCP/IP socket communication along with the translation of the incoming and outgoing messages. The Core, instead of dealing directly with the TCP/IP sockets and the decoding of the messages, uses a Socket Handler instance per connection.

Each message exchanged through the communication socket, is formatted according to the proposed JSON object. Any received message is deserialized by the Socket Handler, exposing the data contained in the JSON object into an instance of a C++ class named Exchangeable Object. Thus, the Socket Handler translates the message into a C++ class instance of Exchangeable Object. The fields are identical to the fields contained in the received message but they are made accessible to the rest of the system. This Exchangeable Object is then sent to the Core triggering actions. Reversing the direction of information exchange, in case the Core needs to send a message to the Proxy, it creates an Exchangeable Object assigning to it the values needed to be sent. This object is handed over to the Socket Handler which serializes it into a JSON object with the proposed format.

This encapsulated approach made the information exchange easy, giving this method many advantages over the rest. So, no other module needs to be aware of the proposed communication protocol. Moreover, the communication type is encapsulated, which means that in future work support to other communication types, like Bluetooth,

could be added without affecting the rest of the modules. Finally, one very important advantage can be identified in cases where the connection is dropped and reconnection takes place. In those cases, the only action that the Core needs to make is to replace the old instance of the Socket Handler with the newly created for the new connection. The execution continues unaffected. For the rest of this thesis, whenever there is information exchange between the Core and the Proxy or vice versa, it will be handled by the Socket Handler, both to the SmartphoneHarvester side and to the Proxy side though it will not be mentioned.

5.2.3 Sensors

The main functionality of the SmarhphoneHarvester toolkit is based on the exposure of sensor data. The sensors are able to provide information about the body movement, the surrounding environment of the user or the user's location. In our system their functionality is divided into a number of modules. The GPS functionality is handled by the Location Manager and the NFC operations are held through the NFC Manager. The rest of the sensors include inertial, environmental and directional sensors and are handled by the Sensor Manager module. Before proceeding with the analysis of this module it would be useful to enlist the supported sensors.

- *Accelerometer*: provides the direction and the amount of the forces applied to the device in 3D space. Returns a triplet with floating point values x, y, z, each one representing the forces applied to each axis.
- *Ambient Light:* a high level light sensor encapsulating the light intensity into light intensity states. Those states include the levels Dark, Twilight, Light, Bright and Sunny.
- *Compass*: returns the azimuth of the device measured in degrees from the magnetic north in a clockwise direction based on the top of the user interface.
- *Gyroscope*: returns the rate of angular change measured in degrees per second in 3D space. The values are contained in a triplet of x, y, z.
- *Light*: a light sensor reports integer values of the environment's light intensity starting from zero at no light conditions and increasing.
- *Magnetometer*: returns magnetic flux density values along 3 axes measured in teslas.
- *Orientation*: A high level sensor that reports the side of the device that is currently on top, providing the value as an enumeration between the six different positions a device can take (see Figure 9: Device's orientation values).

- *Proximity*: A proximity sensor mainly placed at the front side of the mobile device. It reports a boolean value indicating whether an object is close to the phone's front view (true) or not (false).
- *Tap*: detects light finger taps applied on the mobile device, reporting whether the tap was single or double. Its report also contains the direction of the tap including an enumeration whose value is selected by one of the six different directions in 3D space (Left, Right, Vertical Up, Vertical Down, Horizontal Up and Horizontal Down).

The sensors mentioned above, were encapsulated by the Sensor Manager. The encapsulation includes three operations for each sensor. The activation and deactivation of the sensor are essential for their control and are followed by the detection and the reporting of their state changes. The activation of all the sensors is held through one function, which receives as input a value of an enumeration identifying the sensor. It has been implemented in this way to keep a reference count of the requests received for the activation of each sensor. If the sensor reference counter is already greater than zero, the sensor has already started so the start command will be ignored, but the reference counter will be increased. All the reference counters are subsequently used when a deactivation command regarding a specific sensor is received. The reference counter will initially be decreased and, if and only if the new value equals to zero, the sensor will actually be deactivated. This prevents a sensor that is in use by other components to be deactivated.

Any new value that is extracted by the sensors is reported to the Sensor Manager and redirected to the Core and the activated Gesture Recognizers.

5.2.4 Gestures

The Gesture module is divided into two subsystems. The Gesture Recognizers are the module directly connected with the Sensor Manager, retrieving sensor data and generating high level gestures if recognized. The second module is the Gesture Manager which is connected to the Core module. Through it, the Core can request the activation or deactivation of each Gesture Recognizer. It is also responsible for receiving the results from the activated Gesture Recognizers and translating them into text through the use of a predefined vocabulary. Thus, the transmission of the gesture from the Core to the Proxy is made easier.

Special focus will be given to the Gesture Recognizers which translate the mobile device physical movements in 3D space into gestures. In the literature there is extensive research on the field of gesture recognition through sensors. Complete studies have been conducted for the creation of recognition algorithms. Those algorithms have been used either to recognize the predefined gesture vocabularies, or to support the creation of custom gesture vocabularies The recognition of a custom gesture is more demanding than that of the predefined ones. It requires generic motion recognizers that log the captured information. The available logged data are mainly accessible via a sliding window mechanism that exposes a set of previous and current logged data. Those data create a sequence of movements that form the custom gesture. For a custom gesture to be set as valid, the user needs to be able to reproduce the exact same movements consistently. This feature gets proven at the training period of the system. During that period, the users are required to execute the same gestures repeatedly for the system to extract movement patterns thus creating the custom gestures. The main advantage of this technique is the freedom provided to the user to decide the movements that are more convenient to be reproduced and utilized.

Yet, this approach introduces a number of disadvantages when the above procedure gets executed on a mobile device. Before the first use of the system, the user is obliged to go through the system's training procedure. As a result, a period of time is required for the system to be available, a fact that makes the system less robust and ready – to – use. Also, during the training period, sophisticated classifiers are required to be used to achieve the best movement pattern recognition, which, in most cases, demand enhanced device capabilities and processing power that are not always available. Moreover, the support of custom gestures can confuse the user. If the user is free to generate any chosen gesture and associate it with controls, there is an increased probability that there will not be any consistency over the cognitive association of gestures with the final control. Thus, the system will become hard to use and less intuitive. The lack of consistency can also result to gesture overlapping and conflicts. Two, or more, defined gestures could have only a small variance regarding the required movement to be identified, which could result to false recognitions. Those conflicts are especially increased in cases of long gesture vocabularies. The user is able to create an arbitrary number of custom gestures which produces a long gesture vocabulary. The bigger the variance of gestures defined is, the harder it gets for the user to be able to remember and reproduce each gesture.

The above considerations were the reason why the SmarhponeHarvester has been decided to support a predefined vocabulary of gestures. Thus, no training period would be required for the system, making the toolkit ready to use as soon as acquired. The recognition should take place in real time through the use of lightweight recognizers. The predefined vocabulary of gestures would be assistance for the real time recognition since each gesture would be optimized for performance. The selected gestures must have diversity at the mobile device movements recognized. Because of the diversity, the system's concentration will be transferred on the performance rather than on the gesture distinction. Moreover, the selected gestures have to be convenient and user friendly when executed through mobile devices. It is necessary to require only natural and non-intensive movements of the hand when holding the mobile device. Also, even though the gesture vocabulary has to be small, the included gestures have to be able to cover a variety of applications.

Those requirements were fully met by Qt's implementation of gesture recognizers through the sensors of a mobile device. Since this implementation is still in development phase and is to be integrated in a future minor version of Qt 5, the source code was acquired and further developed. The initial vocabulary included the following gestures; Cover, Double Tap, Hove, Pick up, Shake, Slam, Turn over, Twist and Whip. The Slam and Whip gestures were rejected because they required too much physical movement which could be both tiring for the user and unnatural to execute. The new gesture that was introduced by SmartphoneHarvester is Swipe. Also a number of fixes and algorithm alterations were made to the preexisting recognizers to meet our standards. The list of the final gesture vocabulary supported by SmartphoneHarvester will be presented.

- *Cover*: It is generated when the mobile device is facing upwards and an object is placed cover the mobile phone at a maximum distance of 4cm.
- *Double Tap*: A quick double tap anywhere on the device.
- *Hover*: like *Cover* gesture, it is produced when the mobile device is facing upwards and an object is placed over the mobile phone. The difference is that, in this case, the object has to be left above the device for 3 seconds before the Hover gesture gets recognized.
- *Pickup*: The mobile phone is facing upwards and is left still on a surface. The user grabs the device and, with a quick movement, brings it to vertical position.
- *Shake*: The intense movement of the device back and forth, up and down or right and left for 3 times in a very short period of time. The gesture includes the axis of movement.

- *Swipe*: A quick movement of the device over an axis. Includes the direction of the movement in the 3D space.
- *Turn over*: Once the device is facing downwards and is close to an object, it is considered as turned over. For example a mobile phone is placed on a table facing downwards.
- *Twist*: The device has an initial orientation. A quick change of the orientation to another value and then back again to the initial orientation constitutes a twist. The intermediate orientation is used to extract the direction of the gesture. The direction values are the Front, Back, Left and Right twists. For example a mobile phone's top is facing upwards. The user twists left, rotating the device 90 degrees counterclockwise for one second and then restoring the orientation to its initial value. This is a Twist gesture with left direction.

5.2.4.1 Location Manager

The simplest module included in SmartphoneHarvester is the Location Manager. Its main responsibility is to handle the GPS functionality of the mobile device. It can toggle the GPS power state and set the location update interval. Thus, it can be setup to retrieve one sample per minute which, for example, could be used to detect whether the owner of the mobile phone has left a building or even the general area of a building to automatically enable the alarm system of the smart house. If the interval time is not set or set to 0 the position's updating will be done as soon as possible, which varies from device to device. Upon receiving any GPS data, it redirects to the Core information containing the latitude, longitude and altitude.

5.2.4.2 NFC Manager

The technological background of the NFC was introduced in section 4.2.2 and the NFC manager was created based on it. By the time the NFC Manager gets created, the target detection is initiated. The term target is referred either to an NFC tag or to an NFC-enabled mobile phone. In case a target is found, a three state sequence is always executed. The target first enters the range of the NFC field emitted by the device the SmartphoneHarvester is running on. Consequently, the data of the target are read or, else said, the NDEF messages are retrieved. Those messages are decoded using the NDEF specification and their records are extracted. For now only Text and URL records are supported but this is subject to change in the close future. When the target exits the NFC effective range, the sequence is

completed. For each of the above states, a corresponding event is sent to the Core. Any message records retrieved are also included in the event that is sent to the Core. Moreover, the NFC Manager accepts requests for writing given data to the NFC tag that is currently in the field's range. For now it does not support any point-to-point connections.

5.2.4.2.1 *Message Sender*

This module is responsible for sending custom text blocks to the Proxy, an operation that is triggered by and associated with an NFC tag. In more detail, along with the default NFC's read/write, an extra functionality was decided to be added. It is an emulation of the write operation triggered by a NFC read operation supporting unlimited text length. Given the desired text block, it is able to send the text block to the Proxy, triggered by an NFC read operation. Imagine that the graphical user interface has a text area in which the user can add arbitrary text with no limitation over the length or the type. When the composition is completed and an NFC tag gets read, the text gets associated with the ID of the recognized NFC tag and the pair is redirected to the Core and later to the Proxy. Thus, we have a message sending system triggered by NFC read operations integrated into our platform. An example of its usage can be found in section 6.1.1.

5.2.4.2.2 *File system Manager*

A great advantage of our toolkit is the support of file system access. The storage capacity of the mobile phone is growing, making the case of the 8GB common. It was a great opportunity to take advantage of the on-device storage considering that a smartphone is carried along with the user. Thus, the files related to the user could be stored on the mobile phone and be available anywhere. Reversing the operation, it is very easy to imagine the case where the user is travelling and is using the mobile phone as a means to view or create presentations and download media or images. When the user returns at the office, the newly created files will be downloaded automatically to the office environment's backend. The list of examples that prove the importance of supporting such operations is very long. All the file system functionality can be provided through three operations; browse the contents of a folder, retrieve a number of selected files and upload files to a selected path. All three are triggered by only one incoming request from the Proxy.

In the case of the file browsing, it only requires as input the path of the folder that is to be browsed, along with an optional input which defines whether the browsing operation

that will take place will be recursive or not. If the optional input is missing, the recursion of the operation is automatically disabled. The response to this request is a list of absolute paths that contain the files and the folders that were discovered. The information returned to the proxy, is the absolute path of the entry and a boolean value indicating whether the entry is a folder or not. The “true” value means that the entry represents a folder. More information about the entries, like the creation date or the size, is not included since it was evaluated as unnecessary data.

Proceeding with the actual data exchange, data can be transferred from the mobile device to the Proxy and vice versa. At this point, we faced a challenge. Take for example the situation where the Proxy sends a request to the mobile device, asking to retrieve some files from the mobile device. The request contains a list of absolute paths representing the selected files. The request is received by the Core module and is redirected to the File System Manager for further handling. The problem arouses on how these data will be transferred to the Proxy. They are binary data and our default protocol of transfer includes the use of JSON objects. The practice of transferring data packets through JSON objects poses an extra overhead. To include file data in a JSON object, the data need to be converted from a byte array to an array of 64base characters. The receiver, by its turn, will have to convert the file data back to a byte array form. From a technological side of view, our system is programmed to use TCP/IP connections which, by definition, are used as a reliable stream to transfer files. Not using the capabilities of TCP/IP connections would be an important waste of effort, and processing power, while it would result to very poor transfer rates. To solve all the above issues, a separate File Retriever module was created to be used from both the SmartphoneHarvester application and the Server.

5.2.4.3 File Retriever

The solution was brought by creating new TCP/IP sockets and establishing dedicated connections for each transfer request. This functionality can be easily explained through a step-by-step example of a file transfer. We will start by explaining an example of file retrieval and then one of file uploading to show both ways of the operation.

The Proxy has already requested the browsing of a specific folder and has already retrieved the list of files and folders contained in the specified folder. Between the files in the list, a group of files is selected and are about to be requested for retrieval from the mobile phone to the Proxy. When all the absolute paths of the files are available, the JSON-based message is formed to make the request. The type of the operation is file retrieval and

also includes the list of the absolute paths of the files. Prior to the message transmission, the Proxy creates an instance of the File Retriever. The File Retriever, upon its instantiation, creates a new TCP/IP server at the same IP address as the main Proxy server and an available Port number, which is provided by the operating system, is assigned to it. Through this new step, the message that is about to be sent gets extended to include also the IP address and the Port number of the File Retriever server. The message is transmitted to the remote device and handled by the Core.

The Core extracts the contents of the message and passes them over to the File System Manager requesting as well the initiation of the file retrieval. Initially, the File System Manager creates a simple socket and binds it at the given IP address and Port number. If the socket is successfully connected, the File System Manager is directly connected with the File Retriever at the Proxy and the operation can proceed. The File System Manager at the mobile device iterates one by one the file path list and executes the following operation for each file. It initially retrieves from the file system the length of the file specified by the path. Then, follows the creation of a message whose Operation is Response having as Request Type the File Retrieval and containing the path of the file and its length. This is the introductory message received by the File Retriever before each file transfer, stating “who I am” and “what my length is”. Having retrieved the file information, next step is the file data transfer. The local file is opened for reading and its data are transferred through data chunks of 1MB. Each chunk is sequentially copied to a Byte Array. This Byte Array is given as input to the socket thus it is transmitted. The above operation is repeated till the entire data set of the file is transferred.

Back to the Proxy, the operations taken to handle the above data will be shown. Upon receiving a message as response that contains the path of one file and its size, it enters the state of file construction. In that state it creates the local file that will be used to store the received data. Also as the overall length of the file is known, the Proxy is aware of the number of Bytes that are expected to be read from the socket till the file is complete. After the file gets completed, it returns to the previous state of expecting the introductory message for a file and so on. The Proxy, from the beginning of the procedure, knows how many files have been requested, so, as soon as it receives all the pending files, it closes the server and destroys the instance of File Server. Likewise the File System Manager at the mobile device closes the opened socket by destroying it and returns to its normal execution.

This approach has many advantages. Each session of file transferring is executed independently without affecting the rest of the system. Because of their isolation, multiple

simultaneous sessions of file transferring can be active. Also, the transferring rate of the file is maximized since the file is transferred through a stream, not in packets. In this way, there is no overhead from breaking the file into packets and reassembling it back to normal at the other end of the connection.

The exact same operation reversed is held in case the Proxy wants to send data to the mobile device. The only difference lies in the initiation of the operation. The Proxy, once again, initiates one TCP/IP server sending a request message of File Upload to the mobile device. It contains the IP address and Port number of the server along with the number of the expected files. Once the mobile device gets connected to that server, the aforementioned operation is held with the roles of the sender and the receiver swapped.

5.2.4.4 Shared Folder

Having implemented the three file system operations, browse, retrieve and upload, the file system support was utilized in a number of scenarios. One problem was identified. Our toolkit is multiplatform so the File System Manager has to deal with differently formatted file systems. For instance, Symbian devices have file system path addressing that resembles the Windows machines, (E:\) but any other Linux-based or iOS-based devices utilize the classic root folder addressing of Linux. Also, it became important for some applications to store data on the But it was not comfortable to remember the different formatted absolute paths for all such applications.

For this reason, two “shared” folders were created. They are folders with virtual specified paths, though internally translated into platform depended paths. The paths “~” and “~/Office” were introduced. If, for example, the Proxy places a request of folder browsing on one of those two, when retrieved by the File System Manager of the mobile device, it is first translated into its real path depending on the platform that the SmartphoneHarvester is running on. Then, the request gets executed. The file retrieval and file uploading work alike using the above two paths.

To create the association of the paths, when the SmartphoneHarvester is executed it creates the two folders if not existent. Table 2: Shared Folder info shows the association of the virtual paths with the actual paths depending on the operating system. iOS is the only case of relative path since its file system cannot be accessed with full privileges unless it is at the same folder or a children folder of the application’s root folder. That is because of the creation of a “Sandbox” for each application for security reasons.

	Symbian	Linux-Based (Android)	iOS
~	E:/Ami/Shared	/mnt/sdcard/Ami/Shared	./Ami/Shared
~/Office	E:/Ami/Office	/mnt/sdcard/Ami/Office	./Ami/Office

Table 2: Shared Folder info

5.2.5 Audio Manager

Through the literature review, we have seen a number of context aware systems. In those systems the main practice was that the more data we have the better the outcome of the context recognition will be. This concept is contributed by the Audio Manager. It is a small module that once it gets activated, it continuously captures sounds. The sound input is processed and normalized to generate sound levels. Reporting values from 0 to 1 is the medium through which a system can identify the user's surrounding noise. This, for example, could be used to adapt the alarm levels when there is too much noise around the user or when an important event takes place and it is essential that the user realizes it. When the audio capturing is started, it can be set to one out of four levels of sensitivity affecting the reporting output. If sensitivity is unused or set to High, then by default the noise level captured by the mobile device will be sent directly to the Proxy. If sensitivity is used, the normalized range between 0 and 1 is divided into ranges. Then, the reporting to the Proxy will take place only if the sound is varying enough to move to another range. The Proxy will be informed the moment the sound level changes range. If Medium sensitivity is used, the range is divided into 10 ranges, 10% each. If Low sensitivity is used, the ranges are four, 25% each.

5.2.6 Command Executer

One of the forward thinking features of SmartphoneHarvester toolkit is that it provides the ability to go beyond its non-obstructing operation. The Proxy is able to place a request to the mobile device to open a specific file in its file system using the default file viewer of the device, depending on the file type. Fox example, the Proxy requests the opening of an image located at a specific path at the mobile phone. When the SmartphoneHarvester receives the request, it will try to open the file specified by the path with the default viewer of the device. If no such viewer is supported the request is ignored.

5.2.7 Hardware & Device Information Manager

Having a system which adapts according to the platform and the device on which it gets executed, it is essential to provide a method of retrieving the actual data that triggered the adaptations. For example, SmartphoneHarvester can detect the supported sensors through which it selects the supported gesture recognizers. An analogous adaptation could be required by an end-application that uses the toolkit. In that case, the SmartphoneHarvester must be able to send the supported sensor list to the Proxy when requested. Generalizing the above example it would be useful to make available the complete list of the supported capabilities of the mobile device along with the device specific information.

This led to the creation of the Hardware & Device Information Manager. When the SmartphoneHarvester gets executed, it initializes its information. It starts by retrieving device data that include the manufacturer name, the model and the unique identification number of the device. It continues with the identification of the operating system and its current locale settings, followed by the identification of the charging state of the device and two additional mobile phone specific features, such as the lock state of the device and the current selected profile. Moving to the hardware information, it queries the device and retrieves the list of the supported sensors. Using the sensor list it creates the list of the gesture recognizers that will be enabled. The GPS and NFC support is queried to the responsible modules, the Location Manager and the NFC manager respectively. The above set of information is stored and can be retrieved when the Proxy sends a request for Hardware Info to the Core. The response is created by the Core module containing all the extracted information by the Hardware & Device Manager.

Apart from the extraction of the above static information, the Hardware & Device Manager is responsible for identifying and reporting changes at the state of the lock status, the charging state and the currently selected profile. Whenever change in a state occurs, it is reported to the Core and the message gets redirected to the Proxy.

5.2.8 Network Interface Manager

A challenge that was brought up was how to automatically handle cases of network loss. When targeting mobile devices, the system should be designed to handle issues that are caused by the mobility nature of the usage of the mobile devices. One of the issues that are highly possible to occur is the loss of Wi-Fi reception for a period of time. For example, the user has the mobile device and goes out of the building for lunch at a nearby restaurant.

The Wi-Fi reception is lost thus, the SmartphoneHarvester gets disconnected. It would be ideal if the SmartphoneHarvester could automatically be reconnected to the Proxy when the network gets active once again.

The Network Interface Manager was created to handle such cases. It observes changes of the underlying active Network Interfaces of the mobile device. When the network connection is lost, the Network Manager will detect that the Network Interface got deactivated and will inform the Core about the disconnection. Thus, the Core will know that the Proxy is unreachable at the moment and will destroy the inactive instance of the Socket Handler that was in use till then. Once the network is reestablished, a Network Interface will get activated and the Network Manager will detect it and inform the Core. The Core will then create a new Socket Handler and will try to reestablish the socket connection. Thus, as soon as the user returns to the building after the lunch, the network will be reestablished and the SmartphoneHarvester will automatically be reconnected.

5.2.9 GUI

The SmartphoneHarvester is a toolkit and is mostly meant for background processing during its execution. In a few cases though, it is necessary to be used as a front-end application. For this reason, a Graphical User Interface (GUI) is provided. At the early stages of toolkit's development, the GUI was feature-rich providing the state toggling of any sensor and gesture recognizer. This initial version can be seen in Figure 18: Initial GUI design.

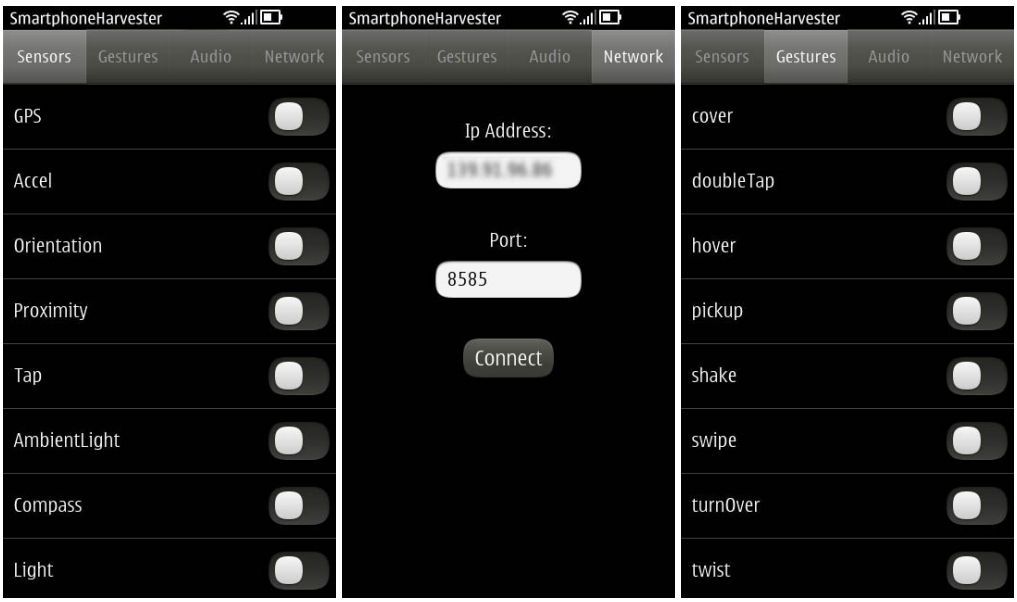

Figure 18: Initial GUI design

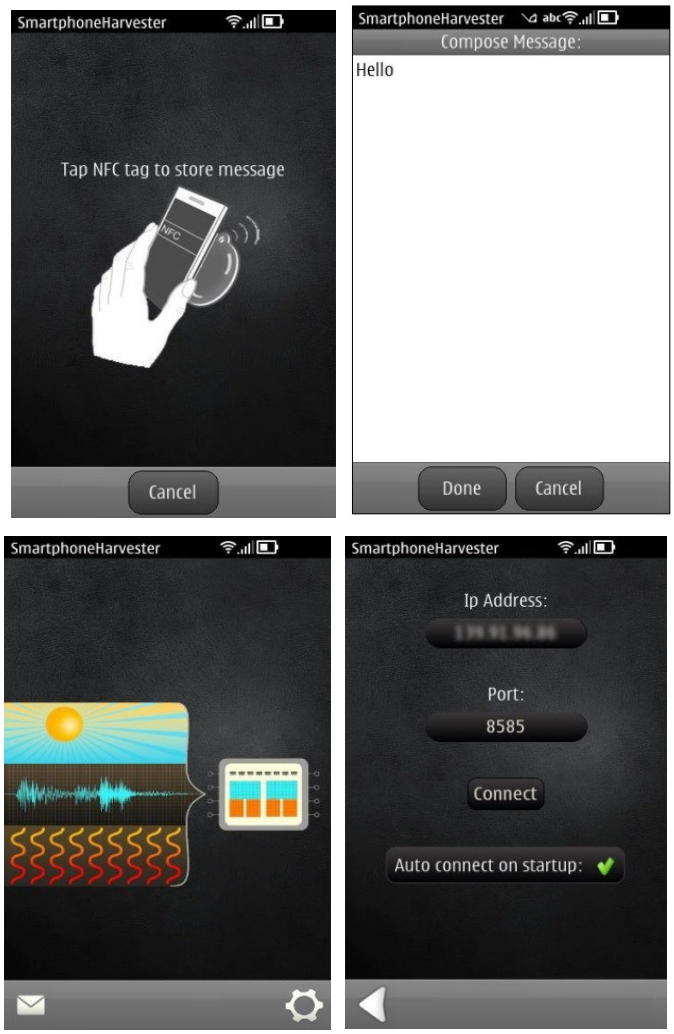

Figure 19: Final GUI

After completing all the tests the GUI was turned into a feature-less user interface because when the toolkit is used, everything is handled remotely. The only functionalities kept were the Network Setup view and the ability to compose messages in the form of a text block, which could be later sent combining the use of NFC and the aforementioned NFC message sender module. The final version of the user interface is depicted in Figure 19: Final GUI.

5.2.10 Completing the gaps in Qt's compatibility

Throughout the implementation phase, the Qt framework was extensively used for the implementation of all the modules. Yet, even though it is functionally complete for the platforms officially supported, the Android platform is currently supported through the use of the third-party Necessitas project. Being third-party suggests that no reassurance can be given for complete API porting to the Android platform. That was the case for three

operations needed in three of our modules. The NFC, the audio capturing and the command execution were not supported on Android so they had to be implemented using platform specific tools and APIs.

The Android operating system is based on Java programming language. Android's full featured APIs can be accessed through Java programming language, though it is possible to write parts of an application in C/C++ through its Native Development Kit (NDK). As an integral part of the NDK, the Java Native Interface (JNI) is the framework that enables Java code running in a Java Virtual Machine to call, and to be called by, other code portions written in other languages like C and C++. This functionality was used in our toolkit to implement the aforementioned missing functionalities. The implementations are written in Java programming language and are linked to a C++ class exposing the data to the rest of the toolkit.

The creation of multiple implementations for the same functionality, depending on the target operating system, was based on the Private Implementation software pattern (Pimpls). According to this pattern, when a public class needs to be defined and implemented, a second private class will be defined which will be holding all the implementation. The public class will provide the interface to the rest of the application, whereas the functionality will be provided by the private class contained in, and used by, the public class. This pattern encapsulates the actual implementation and its private members into a class that is only visible by the public class. In C++ this is implemented by creating a forward declaration of the name of the private class in the header of the public class and adding a private friend member, whose type is the forward declared class. In the source file, that uses the #define macro, the private implementation header file can be included and instantiated. Any call to a function of the public class is redirected to the analogous function call of the private implementation. Using this pattern, multiple private implementations can be created, one for each operating system for instance. Thus, when the application is compiled it includes only the implementation that corresponds to the targeted operating system.

A general note here is that the iOS implementation of the toolkit lacks NFC support and command execution. The NFC is not supported as, by now, is not adopted by Apple. Moreover, no command execution was successful during our tests, so it remains as a good suggestion for future work.

5.3 SmartphoneHarvester Middleware Proxy

In the previous sections, the SmartphoneHarvester mobile application was presented along with the definition of the communication protocol. The SmartphoneHarvester's modules were examined one by one through examples where the modules were using the Core to communicate with the Proxy. This section will focus on the other side of the connection, the SmartphoneHarvester Middleware Proxy which will be referenced as the Server. A brief overview of its modules and its functionality was introduced in section 4.3.1 where the system architecture details and design decisions were analyzed. Based on the proposed final system design, the Server's two main modules are the Broker and the Proxy. Both those modules contain implementations of middleware services to bridge the exposed information from the remote mobile devices to the middleware. An overview of a more in-depth Server's architecture is shown in Figure 20: Broker application architecture.

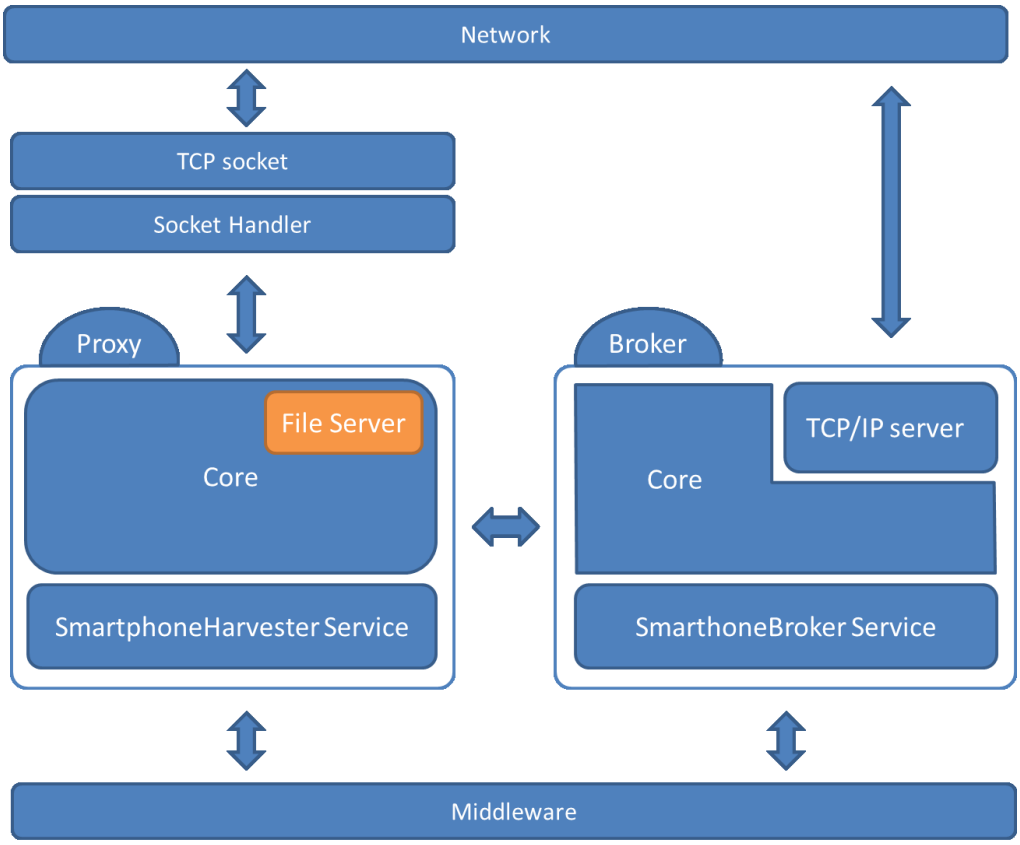

Figure 20: Broker application architecture

5.3.1 Proxy

The analysis of the Server will start by presenting the Proxy module which was extensively referenced at the previous section. It is the part of the Server that handles the communication with the SmartphoneHarvester mobile application and exposes those data to a middleware service. Hence, multiple instances of this module can coexist since there will be one instance per mobile device connected to the Server.

The Proxy module is directly connected to the SmartphoneHarvester mobile application so the Proxy had to be conformed to the communication protocol and message format proposed, as described in the section 5.1. This process was trivial since the Socket Handler module that was used at the mobile application, has been designed to be a reusable component. By using the already implemented Socket Handler module, the socket functionality was already encapsulated and the message serialization and deserialization was also included. Thus, the Proxy, using an instance of the Socket Handler, covers the complete intercommunication with the remote device.

The Socket Handler instance, later used by the Core module, is created upon new successful connection of a mobile device to the TCP/IP server hosted by the Core module. The server's IP and the Port number are provided by the Broker upon the initialization of the Proxy. Upon accepting the first incoming connection, the server gets deactivated. Thus, it preserves the one – to – one association of a Proxy and a mobile device.

The instance of the Socket Handler is connected with the Core module of the Proxy. This Core module is the orchestrator of the Proxy. It is the component that redirects the messages received by the mobile device to the dedicated instance of the middleware service (Service) and vice versa. Any message received by the mobile device is processed by the Core, and depending on its type, triggers an operation in the Service module. That could be a state update or a response to a request placed by the Service module. The state update is sent through Service module as an event to the middleware clients and the response returns the result of the request to the caller client. Another part of the Core module is the File Receiver component that was introduced in section 5.2.4.3. One File Receiver instance is created for each file exchange session.

Having established the procedures through which the mobile devices exchange messages with the Proxy, the next step is the exposure of those data to the middleware. For this purpose, a Famine service has been defined using the IDL language, which is the interface of all the functionality provided by the service. Most of the information exchanged by common messages is defined through enumerations within the service definition, making

the service more comprehensible. The supported operations provided by the service include functions for querying the state of the device, functions for placing requests to the remote device and a number of predefined events for exposing any state change received by the mobile device. Practically, the complete functionality of the SmartphoneHarvester mentioned in the previous section, is transferred to the service definition as well. The above are implemented by the Service module which gets registered to the middleware upon the Proxy's instantiation and is accessible via the interface defined by the SmartphoneHarvester Famine service definition (advice appendix section 10 for the service definition written in the IDL).

5.3.2 Broker

Having defined the Proxy module which will be responsible for handling the mobile device and exposing it to the middleware, the system analysis will proceed with presenting how the Proxy instances are created and initialized. That is held by the Broker. The Broker module is responsible for handling the newly connected devices and for informing the middleware clients about the new connection. Part of this module is a TCP/IP server which is constantly available and constitutes the entry-point of the mobile devices to the system. Thus, for a SmartphoneHarvester mobile application to connect to a Broker, the only prerequisite is to know the Broker's IP address and the Port number. Once the remote device gets connected, a TCP/IP socket is created. The first message sent to the remote device is a request for device information retrieval. As soon as those data are acquired, the Broker creates and initializes a Proxy instance. This initialization is crucial since the Broker passes the IP address and Port number, at which it will receive the connection of the mobile device, to the Proxy instance. The Port number is randomly selected from a list of available ports provided by the operating system. Also, the Broker provides the context with which the Proxy will be registered to the middleware. The context is a concatenation of the device manufacturer, the device model and its unique identifier all of which are contained in the device information already retrieved.

The Proxy is considered initialized when its server is alive at the given address and the Service module is successfully registered to the middleware with the context that was provided. As soon as the initialization is completed, the Proxy reports back to the Broker for the success state. Consequently, the Broker sends a message to the mobile device, which is still connected with the Broker, to request the reestablishment of its connection with the Server, though, this time, at the IP address and Port number provided to the new Proxy

instance. Thus, the mobile device is disconnected from the Broker and is connected to its dedicated instance of Proxy module. When everything gets executed successfully, the Broker, via its Service module, sends an event that the new Proxy is created and registered at the above selected context.

The Broker's Service module implements the SmartphoneBroker service (advice appendix section 10 for the service definition written in the IDL). This service is responsible for providing, when requested, the list of the connected devices. This list is updated every time the Broker reports the connection of a new mobile device through the aforementioned procedure. Moreover, a Proxy instance reports its state to the Broker, in case it is terminated when its associated device gets disconnected. The Broker, on its turn, sends through its Service module an event, which includes the context of the service that is about to be unregistered, probing the unregistering of the specific Proxy from the middleware.

6 Towards Multimodal Applications

The SmartphoneHarvester toolkit was introduced in the previous sections. Its design and capabilities were thoroughly analyzed so, by now, the overview of the proposed system should be clear. Having the SmartphoneHarvester toolkit, the next goal was to make it of use in real systems to prove the requirements set by the Motivation & Contribution chapter 3. Summarizing the key goals, our toolkit needs to be multiplatform, feature-rich and integrated into the Ambient Intelligence Environment. Two applications were created based on the proposed toolkit which will be presented in the following sections. The scenario of "A day at Work" includes a set of applications created to integrate the smartphone usage into the work space environment. It is followed by the analysis of the SmartCube project, a multimodal system for controlling Smart Home environments through the use of smartphones and smart objects.

6.1 A day at Work

The enhancement of the office environment is a subject of active research as reviewed in section 2.2.3. This research field is considered important since the hours spent at work cover a great percentage of the human's life. Thus, any change applied to this part of a person's life can have immediate consequences on the individual's everyday life. Considering the above, a set of applications were created to integrate the use of smartphones in the office environment through the use of SmartphoneHarvester toolkit. The smartphone has become constituent part of everyday life. It is carried along, to most of the activities executed by a person. Thus, its continuous presence can be exploited to provide the user a comprehensible means of controlling the environment and interacting with it. Utilizing this idea, combined with the toolkit's capabilities, the "A day at Work" scenario was created as a set of applications working together. The scenario will be described, followed by the analysis of each application created to implement the scenario.

"Bob has just arrived at the entrance of the office building. He unlocks his mobile device and touches it to the NFC tag placed at the entrance. The door unlocks and Bob walks in, entering the main hall. A TV screen placed at the main hall welcomes him stating the number of messages his coworkers had left for him at his absence. On the way to his office, he passes by John's office. John is not currently there so Bob decides to leave him a message. Bob opens the SmartphoneHarvester application at his mobile phone and composes the message for John. As soon as the message is ready, he taps the mobile device to the NFC tag placed in front of John's office thus sending the message. Bob reaches his

office and logs in to his personal computer. First checks the messages left by his coworkers during his absence. Next, opens his documents folder to access the presentation that he created the night before using the mobile phone. The time passes and it is time to go to the meeting. Before leaving his office, he uploads to his mobile phone the files that will be used at the meeting. Those files include a presentation, a video and an image. Bob reaches the meeting room and takes his place at the meeting table. He places his mobile phone on the table, touching with the mobile device the NFC tag in front of him. The mobile phone gets associated with his position at the table and an interaction area appears in front of him on the table. The files that were previously uploaded to the mobile phone are now automatically retrieved and shown at the interaction area associated to Bob. The same procedure is held by the other meeting attendants, Jack and Mary. Bob, using the special infrared-emitting pen, selects the presentation at his interaction area and opens it. The presentation appears at the main TV screen to be visible by all. During the presentation Bob uses the pen to move the slides back and forth through the toolbar in the interaction area. At the middle of the presentation he stands up and starts walking around the room continuing the presentation. Now, the slides change by swiping the mobile device to the right over the air. At the end of the presentation he performs a twist back gesture to close the presentation, swipes to the right to select the next file, the video, and twists front to open the file to the main screen. During the video play, Jack wants to make a suggestion so Bob swipes the mobile device to the left to pause the video. As soon as the conversation is finished, the video is resumed by a right swipe. At that point, Bob returns to his seat and closes the video using the toolbar shown at his interaction area. Next Mary takes the lead and opens a spreadsheet to the main screen. Just before the end of the meeting, Mary selects to share the spreadsheet with Bob and Jack. The spreadsheets file is copied to the areas of Bob and Jack and is transferred to their mobile phones. As soon as the transfer is completed, the spreadsheet file is opened on the mobile device, thus Bob and Jack can keep the results for later usage. The meeting finishes and the attendants log out by shaking their mobile devices."

6.1.1 NFC Message Server

During the development of our system, the NFC operations were studied. This study has resulted to the proposal of a hybrid solution regarding the NFC write operation. The NFC tags have a maximum content capacity which is varying from 44 Bytes to 2K Bytes. This capacity range was taken into account for the cases of note taking or messaging to other people. The overall text length of all the messages must not exceed the aforementioned

limited capacity, restricting the number and the size of the stored messages. Such a limitation makes the system useless. This issue was solved by the combination of the NFC message sender functionality, analyzed in 5.2.4.2.1, and the creation of a message server as a client to the SmartphoneHarvester service. The NFC message server (server), upon its execution, resolves the Broker service which is registered by the context “office”. Hence, this context groups all the mobile devices used in the office environment. The next step is to request from the Broker service the complete list of the mobile devices already connected. For each mobile device connected, this server resolves the SmartphoneHarvester service associated with the mobile device. If any new device gets connected to the Broker after the aforementioned initialization, the event that is send by the Broker is handled. Thus, the new device’s service instance is resolved as well. The NFC read event, in which this server is interested, is produced when a message block is sent via the NFC message sender on the mobile application. The server stores the text block associated with the id of the NFC tag that was touched. The backend of the storage functionality was implemented by using a lightweight MySQL database provided by the Qt’s Offline Storage API.

```
7  module Office {
8      interface NfcMessageServer {
9
10         struct Message{
11             string tag;
12             string originContext;
13             string message;
14         };
15
16         typedef sequence<Message> MessageSeq;
17
18         void AddTagToContextAssositation(in string tag, in string contextName);
19
20         void AddMessage (in Message messageInfo);
21
22         MessageSeq GetNewMessagesPerTag(in string tag);
23         MessageSeq GetNewMessagesPerContext(in string contextName);
24
25         long PeekNumberOfNewMessagesPerContext(in string contextName);
26         long PeekNumberOfNewMessagesPerTag(in string tag);
27
28     };
29 };
30
```

Figure 21: NFC message server Interface through IDL language

The messages were made accessible by implementing a dedicated service for the NFC message server. A client is able to query the number of messages or to retrieve all the messages that are associated with a specific tag. For simplicity reasons, at the time a client retrieves those messages, they are deleted from the server. Thus, the server only keeps the messages that are not yet delivered to a client. This approach allows the server to keep its

database as small as possible, making the server robust for large systems. Moreover, for the same reasons, the service allows the association of a NFC tag id to a SmartphoneHarvester context. Hypothesizing that one mobile phone belongs to only one user, a NFC tag can be associated with the user's device and consequently with the user himself. This results to linking NFC tags directly with users. So when a user leaves a message through a NFC tag, this message is directly related to another user.

6.1.2 Smartphone Door Control

The Smartphone Door Control application enables the user to open a door by touching an NFC tag through a mobile phone. In our case, the NFC tag was placed at the entrance of the office building for opening the entrance door. It is implemented as a client of the SmartphoneHarvester services instances. Like the NFC message server, upon its execution it retrieves the list of the connected devices from the Broker service and resolves all their service instances. Any new device connection is also handled. The information that is acquired by the SmartphoneHarvester services is the events raised when an NFC tag has been read. In parallel, this application is client to the DoorControl service which is already implemented for the facilities where the system was tested. Also, it implements the SmartphoneDoorControl service. Its purpose is to raise an event whenever the door is opened via NFC by a mobile device interaction. The event includes the context associated to the mobile device that opened the door.

The id of the NFC tag that is placed at the door is associated with the name of the door through a configuration file before the Smartphone Door Control execution. Thus, as soon as an NFC tag is recognized, this application uses the id of the tag to find the associated door's name. If no name is found, then the event is ignored. If the association exists, the door's name is send to the DoorControl service requesting the door to be unlocked. The list of associations of tags and doors can be as long as needed. In this way, this application manages to handle the door control of a complete building.

6.1.3 Welcome Screen

The Welcome Screen application is the screen that welcomes the user to the building. It is client to the above two new services that were just defined, the NFC message server and the Smartphone Door Control. As soon as it receives an event that the entrance door was opened by a specific mobile device, it queries the NFC message server for the number of pending messages associated with the context of the mobile device. Meanwhile, the application creates a borderless window shown on top of any other application running

at the same screen, which contains a welcome message. If the number of pending messages is greater than zero, below the welcome message, indicates the number of the messages (Figure 22: Welcome message on screen).

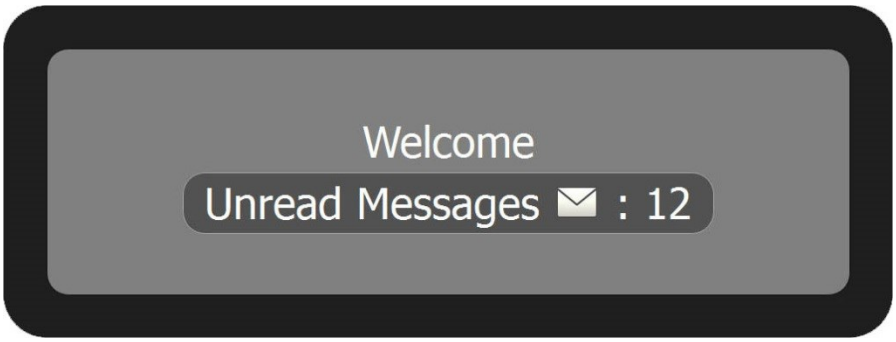

Figure 22: Welcome message on screen

6.1.4 Mobile at Work Assistant

The MobileAtWorkAssistant application is the tool through which a number of operations, that are included in the scenario, get executed. It is the application that will be running on the computer of all the employees, providing essential functionalities for the proposed scenario. Its basic views can be seen in Figure 24: Mobile At Work Assistant views and will be analyzed one by one.

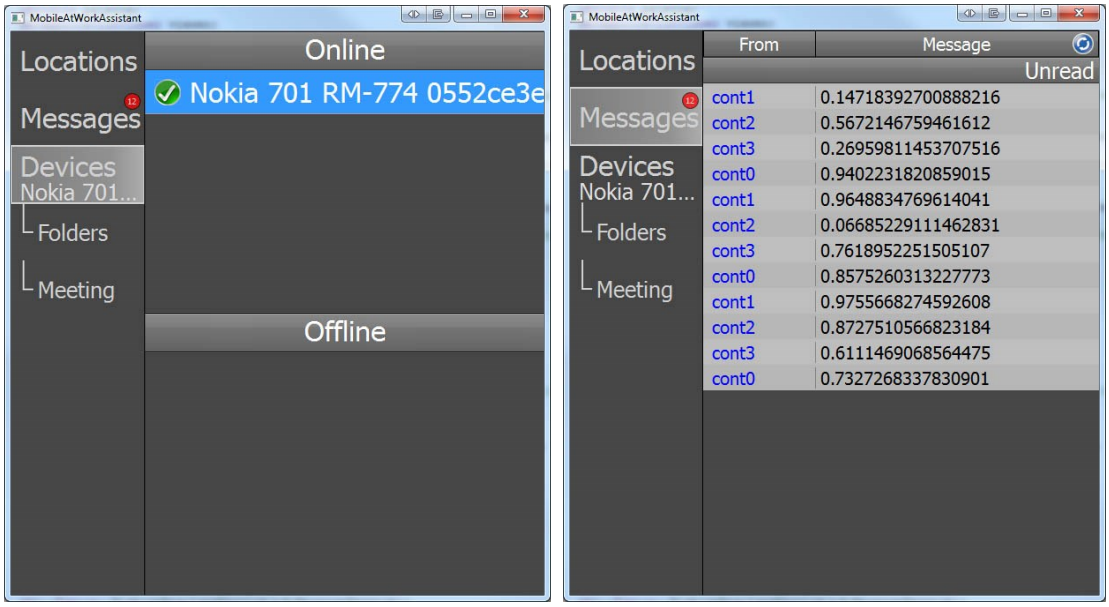

Figure 23: Mobile At Work Assistant views (1/2)

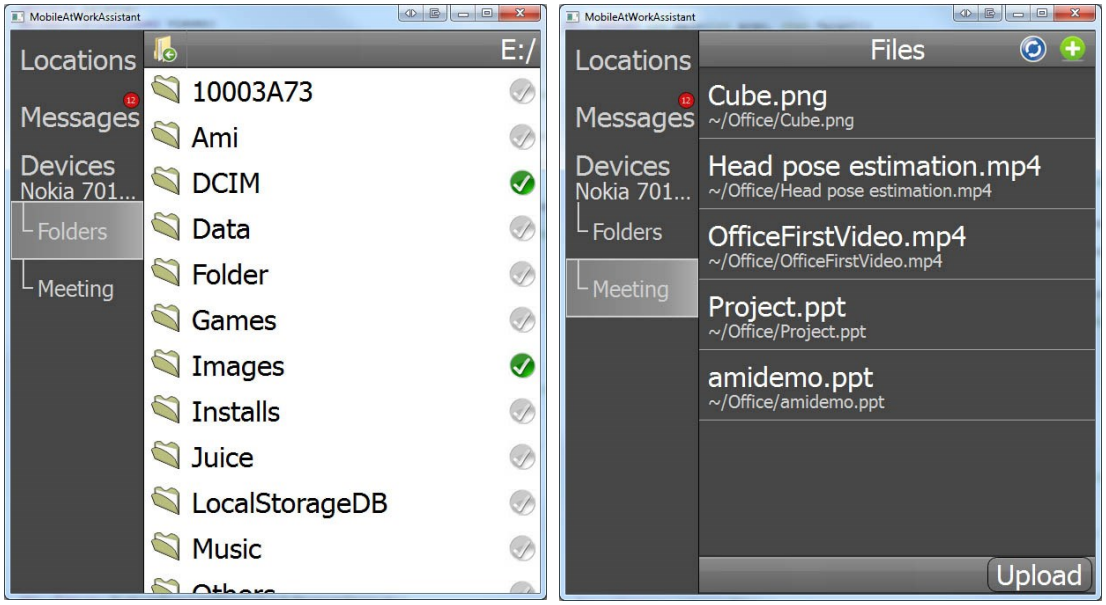

Figure 24: Mobile At Work Assistant views (2/2)

This application is dependent on three services, the SmartphoneHarvester service, the NFC message server and the Smartphone Door Control service. It is created as a front-end application, being client to the above services. In this way, it will support the association of connected mobile device to the user of the application. Also it will allow the browsing of the mobile device file system supporting folder selection. The selected folders will be monitored and any new files added to them, will be automatically retrieved by the user's personal computer. Moreover, the pending messages left for the user will be able to be viewed. Finally, the application is related with the preparation of the user to attend the meeting. It will be used by the user to view the files already uploaded to the mobile phone during a previous meeting. The list of the uploaded files can be edited by removing or adding new files. Each view contained in Figure 24: Mobile At Work Assistant views will be presented in more depth and will be referenced by the name shown in the figures.

- Devices view

The Devices view is the default view upon startup. It shows the complete list of all the connected devices as retrieved by the Broker service. The Broker service provides the service context of each mobile phone, which is the name of the mobile device concatenated with its unique identification number. So the Devices view is practically populated by the list of available contexts. For the sake of readability, the contexts, before being added to this list, are edited to remove the unnecessary punctuation. Through the list of the contexts of the device, the user selects the preferred mobile device. The selection is made by clicking on

the tick, placed on the left of the device name at the entry of the list. This step is very important since it results to the association of a mobile device with the application. The association will be used in later views of the application where it device specific information need to be shown. One important fact is that through this operation, more than one mobile device can get associated with the application. Though, the following views will be accessed for one device each time. To switch the target device, the user returns to the Devices view and clicks the desired entry of the list. By selecting one device to operate on, the name of the device is added to the Devices tab label on the left of the graphical user interface. Any change at the preferred selected devices is consistently stored and is available after the termination and re-execution of the application. This fact applies to all the views that follow.

- Messages view

The Messages view is utilized to present the pending messages for the user. The messages are retrieved from the NFC Message Server, provided an id of a NFC tag. Since the application will be executed at the computer of each user, the NFC tag will be the one placed at the office of each user. Moreover, the devices that get selected and ticked in the Devices view are associated with that NFC tag id. This association of the context of the device with the NFC tag id is sent to the NFC message server enabling the retrieval of messages.

The message retrieval described above can be triggered by two events. It is either automatically initiated by a specified time interval, or when the user unlocks the door at the entrance through the Smartphone Door Control application. Either way, the Message view gets updated by retrieving the messages. If another view is currently selected, a red circle is placed at the tab's label of the Message view, indicating the number of the unread messages. Inside the Message view, the user can go through the list of the unread messages along with the previously received messages that are already read. When the mouse hovers above each list entry, a button is shown on the left of each entry. For the unread messages, this button sets the message as read, whereas for the already read messages it is used to permanently delete the message.

- Files view

The Files view is used to select the folders of the mobile device that will be checked for changes. Any new file that gets added to those folders, upon the execution of the synchronization, will be transferred to the folder located at the personal computer of the

user. This folder path is set at the Locations view. The synchronization is set to be executed at the time the user unlocks the entrance door through the Smartphone Door Control application.

Every time the user selects the Files view, it is a prerequisite to have one remote device selected by the Devices view. If it is the first time the Files view is shown for the selected device, the service related to the device is resolved. Through the resolved service, the application makes browsing requests whose results are consequently visualized. The result list includes only the folder entries that are found, because this view is created to select the already checked folders. The visualization of the folder list is constructed by two integral parts, the address bar placed at the top and the list of the browsing results filling the rest of the window. The address bar, as it indicates, shows the current path. In case the mobile device supports multiple drive letters (C:/, E:/ etc.), the address bar provides a selection of the available drive letters when clicked. Moving to the list entries, each entry represents a folder. When clicked, the browsing is continued in that folder and the address bar gets updated with the new path. This view supports the operation of moving up and down in the folder hierarchy. Each folder entry has a tick on its left. If this tick is clicked, the folder referenced by the list entry is added to the list of the watched folders. Likewise, the un-ticking of a folder results in its removal from the watched list.

- Meeting view

The scenario states that before the user leaves the office for the meeting, the necessary files that will be used in the meeting are uploaded to the mobile device. This functionality is handled by the Meeting view. Assuming that a device is selected by the Devices view, as soon as the Meeting view is selected, a list of files will be shown. Those files are the contents of the folder "~/Office" on the mobile device. It is the virtual path that was introduced at the description of the SmartphoneHarvester toolkit implementation in section 5.2.4.4. The file list shown by the Meeting view can be edited. Files can be deleted from the list which results to the actual deletion of the file from the phone at that specific path. Furthermore, files can be added to the list by pressing the green circle button. When pressed, a file view, like the one implemented for the Folders view, is shown but this time the contents are populated by the local file system of the user's computer. Any file desired, can be added to the list of the files for the meeting by double clicking on it or by using the keyboard to select it. By the time the new files are added, the user presses the upload

button and waits for the transfer to be completed. After this process the user is ready to attend the meeting.

6.1.5 Presentation Table

The Presentation Table is the multimodal system that was created to enhance the procedures held in a meeting, through the use of smartphones equipped with the SmartphoneHarvester toolkit. It involves the use of NFC technology for user login. File sharing is enabled between the application and the mobile devices of the attendees. Each file can be manipulated through gestures, originating from mobile devices or pen interaction.

The analysis of this application will start by describing the preexisting infrastructure in the physical meeting room where the application will be executed. The meeting room consists of a meeting table and a TV screen placed at the one end of the table. Above the table there is a projector that is used to display information on the table. Practically the table is turned into a big screen where anything can be shown since both the projector and the TV screen are connected to a computer. The users can interact with the projected info by simulated mouse events. These events are generated through infrared light emitting pens which are tracked by a camera with infrared filter. The blobs generated by the infrared light are extracted through the camera and are either translated into mouse drag or mouse click, depending on the operation performed by the user. Also RFID antennas are placed under the table in front of each user. Thus if an RFID card is placed on the table, above the specified areas, it will be recognized. This setup constitutes the hardware infrastructure.

The software infrastructure is based on the Famine middleware and its already implemented services. The list of the services used, starts with the Table RFID Reader service. This service is responsible for reporting the recognition of any RFID card or tag that is placed over an antenna. Upon the recognition of a card or a tag, it sends an event that contains the id of the card and the name of the antenna that recognized it. Moreover, the Multimedia Viewer service creates a window in which, upon request, it can open and show files that are included in its list of supported file types. This includes image files, videos and Microsoft Office related files like presentations through PowerPoint and spreadsheets through Excel. Each file that is successfully opened can be manipulated through commands send to the service. The vocabulary of the commands is accessible by the clients of the service and its enabled set is altered depending on the file type that is currently opened. Examples of such manipulations are the pausing of a playing video or the progress of the

slides of a presentation. Finally, the third service is responsible for the camera, continuously reporting the detected blobs.

The camera service was directly used by an already implemented client included in the software infrastructure. This client translates the data received by the camera service into mouse events through the Tangible User Input Output (TUIO) and the Windows 7 multitouch support. When the light pen's button is pressed and the pen is dragged, a mouse dragging will be generated. If the button is clicked, a mouse click will be generated. Note that since the multitouch backend of Windows 7 is involved, the aforementioned approach can support multiple mouse events at the same time.

Based on the above infrastructure, the Presentation Table was created. The goal was to support common tasks held during a meeting using the attendees' mobile phones. Meanwhile, other operations, analogous to the controls provided by the mobile phone, will be supported by other technologies. This will result in a multimodal system through which the SmartphoneHarvester toolkit can be used in parallel with other interaction techniques. The final application proposed is focused on two main operation concepts; the user login/logout, and the file manipulation.

- Graphical User Interface

Before proceeding with the analysis of the underlying functionality, the Graphical User Interface will be presented. It is the interface that will be projected on the table, with which the users will interact. Its main component is the User Area. The User Area is the replicable component which contains all the data and functionality that are provided to each user. Each logged in user has a dedicated User Area on the table that is placed in front of the attendee's seat, showing a rectangle of components. Since the users are distributed around the meeting table, the User Areas are rotated accordingly. Moving inside the User Area, at the center there is a floating component representing the Toolbar. The Toolbar contains the controls available for the currently opened file. On the right side of the User Area is placed the File Container. It is a rectangular area depicting the list of files provided by the user. Each file is represented by a File Thumbnail which consists of an image that corresponds to the file type. Below the image, a label is showing the name of the file. A File Thumbnail, when clicked, reveals a context menu with four operations that can be applied to the file. The four operations are Open, Share, Close, Delete. All the operations that are related to any of the

above will be presented. Figure 25: Presentation Table interface of User Area illustrates the above description.

Figure 25: Presentation Table interface of User Area

- Login operation

The first operation that a user executes when coming to a meeting is the login to the application. In our case, that is the creation and assignment of a User Area to the user. The user gets prepared for the meeting by uploading the necessary files to the mobile device through the MobileAtWorkAssistant application. By walking into the meeting room, the user takes a seat at the meeting table and places the mobile phone in front of him. This action triggers the creation of a User Area to be associated with the user. For the triggering to take place, two parallel technologies were used. A number of NFC tags associated to each seat are placed on the table in front of each seat. The RFID antennas already mounted beneath the table at each seat constitute the alternative way if the NFC is not supported by the mobile device of the user.

For the above to be implemented, alike the aforementioned applications that were introduced, the Presentation Table makes use of the SmartphoneHarvester service in combination with the SmartphoneBroker. When the application gets executed, it retrieves the list of all the connected devices by the Broker service and resolves all the services of the available mobile devices. Any new connection of a mobile device is also handled and automatically resolved. All the resolved services are internally used by the NFC Watcher

module which is responsible for collecting the NFC events generated by the devices. Since the NFC tags that are placed at the table are unique, the Presentation Table application has a predefined configuration for the association of NFC tag ids to a user's position on the table. So, as soon as a mobile device successfully recognizes one of the predefined NFC tags, it is known that a user holding the specific device is at the meeting table and has interacted with the specific area of the table. This will assign the mobile device to the selected area on the table, if no other user is already assigned to it already. In this case, the NFC Watcher disposes the resolved service and requests from the Device Associator to initiate the login procedure for the selected device at the selected position of the table.

In case the mobile device does not support NFC, an RFID tag can be attached to it. Thus, if the mobile device is placed on the table, the RFID tag will be recognized by an RFID antenna. The RFID Watcher was created as an analogous to the NFC Watcher. Upon the recognition of an RFID tag by an antenna, the RFID Watcher receives a message from the RFID tag id that was recognized and the name of the antenna which recognized it. In the application's configuration file an association of all the names of the antennas with specific positions at the meeting table is included. Moreover, the configuration file has to contain the id of the RFID tag associated with the context of the SmartphoneHarvester service of the mobile device. This approach is not flexible since the above information has to be defined before the execution of the Presentation Table though it was implemented for compatibility purposes for the mobile devices that lack the NFC support. In the years to come, most of the mobile devices will support NFC so this approach would not be needed.

Once the login procedure gets triggered, the application already knows the position of the user around the table and the context of the mobile device that will be in use. The position is used by the application to rotate appropriately the User Area created for the user. The context of the mobile device is used to resolve the device's registered service and associate it to the User Area. Once the service is resolved, the mobile device gets initialized for the operations that will be required. The application requests the mobile device to start the Shake, Swipe and Twist gesture recognizers. It is followed by the request of file retrieval of all the files contained in the "~/Office" folder. Each file that is successfully retrieved is added to the File Container of the User Area and is represented by a File Thumbnail. Upon the completion of the file transfer, the login procedure is finished.

- File Interactions

Once the login procedure is completed, each user has a User Area with all the files needed. The user can interact with the files either by the infrared light emitting pen or by gestures generated by the use of the mobile device.

- File Selection

Initially the user is requested to select a file from the File Container, for instance the third. To support this operation through gestures generated by the mobile device, it was decided to support a file iteration procedure. Such approach is very common in Ambient Intelligence Systems when the input is limited. In this context, one repeated input message is required to select one by one each file in the File Container sequentially. This input message was selected to be a Swipe gesture. If the Swipe has direction towards the right, the next file is selected whereas the left direction results the selection of the previous file. Note that the selected file is represented through a blue rectangle around the File Thumbnail. On the other hand, the file selection through the use of the light emitting pen is simpler. All that is needed is to apply a click on the File Thumbnail and the file is selected.

- File Open / Close

Once the desired file is selected, the user can proceed to opening the file. This will result to a request to the Multimedia Viewer service to open the file. If no other user has a currently opened the file, the Multimedia Viewer will show the file in the big screen. Likewise, the Multimedia Viewer is requested to close the file when no longer needed.

To support such operations through gestures, it was important to select a distinct gesture. Thus, it was selected to associate the Twist gesture with direction to the Front for the file opening and the Twist Back for the file closing. The use of this gesture, will not conflict with the aforementioned Swipe gesture. To execute the operation through the pen, the user first clicks on the File Thumbnail to open the context menu of the file and subsequently selects the Open or Close operation.

- File Manipulation

The file manipulation refers to the requests placed to the Multimedia Viewer regarding the opened file. By the time the file is opened, a floating Toolbar is shown in the User Area. The Toolbar contains a row of buttons with the available operations that can be applied to the opened file. Since the close operation can be applied to any file opened, the Close button is always existent. In the case of a video file, the additional buttons are the Play

and Pause Button or for a presentation are the Next and Previous buttons (see Figure 26: Different Toolbars in Presentation Table application). The buttons are directly accessible through the use of the pen by clicking on them. Meanwhile, the Swipe gestures are used to execute the above operations through the mobile phone. The Swipe Left would be the analogous of pressing the Pause or Previous button and the Swipe Right as pressing the Play or Next button. The same Swipe gestures as the file iteration were used to create a more user friendly interaction with limited vocabulary. Thus, when the User Area is at the state of file selection, the Swipe gestures are used to iterate the files. As soon as a file is opened, the state changes and the Swipe gestures are now used for file manipulation. When the file is closed, the state returns to the file selection once again.

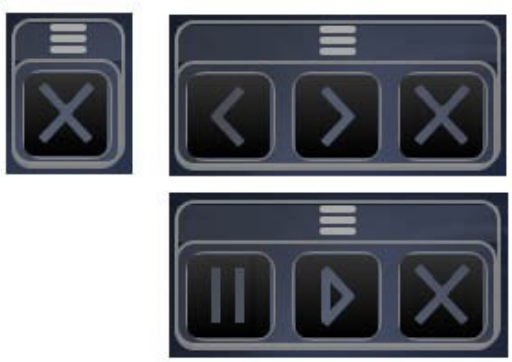

Figure 26: Different Toolbars in Presentation Table application

- File sharing

In a meeting, it could be common request for the users to share their files with the other attendants. The Presentation Table supports such an operation. This operation is accessible through the context menu created when a File Thumbnail is clicked by the pen. It was decided not to assign an analogous gesture control to this, since it is an infrequent operation which cannot be reversed. So the intention was to eliminate any chance of false triggering this operation by false gesture recognition.

The sharing operation gets executed in 3 steps. First the file selected is copied to the User Area of all the other attendants. When the copy finishes, the new file is added to the File Container. Second step is the transferring of the file to the mobile devices of the attendants. The file is selected to be stored at the path specified by "~/Office". After the completion of the transferring, it is requested by all the mobile phones to open this new file with their default viewer associated to that file type.

- File Delete

This is the simple operation of deleting the file from the File Container. The File Thumbnail is removed locally though the file is not deleted from the mobile phone.

- Logout

At the end of the meeting, each user needs to be logged out from the application. For this operation, many alternatives were proposed. The first idea was to execute the same procedure followed for the login. If a user places the phone on the table, the logout procedure is triggered. This approach was rejected since it would be irritating for the users not to be able to leave the mobile phone on the table during the meeting (in case the RFID recognition is used). Also, it conflicts with the reconnection procedure that was implemented for the devices that have lost connection. If a mobile device has lost network connection during the meeting, the User Area is not destroyed. The user is required to touch again the NFC or the RFID tag. In case the smartphone is the same as the one registered to the User Area, the mobile device is associated with the User Area once more.

The ideal approach would inquire the existence of a complete facility of Ambient Intelligence. Hence, a building that can provide the position of each user in the building and the user's current action. Thus, by detecting that the user has left the meeting room, the system could automatically trigger the logout operation. The smartphone accelerometer data could be used in this case to detect whether the user is walking or not. Combined with the detection of the user in the building it could be interpreted as "the user is walking away from the office". The above infrastructure was not yet available, so it could not be used.

The finally selected operation was as simple as possible. It was selected that a user would be logged out if two Shake gestures were recognized within a very small period of time. Even though it is not ideal since it requires intense movements from the users, it narrows down the possibilities were the logout operation could be falsely triggered.

6.2 SmartCube – home appliances control through a Cube

The SmartCube project was created to present the usage of SmartphoneHarvester toolkit to create multimodal systems for Smart Home control. The basic interaction technique applied was the control of the house through hand gestures. The gestures are

recognized either by smartphones or by a Smart Object developed for this project, the SmartCube. The goal was to support a number of controls over the house which would be triggered by the aforementioned devices. Also, through this project, the multiplatform support of the SmartphoneHarvester is broadened. The SmartCube's heart is a Raspberry Pi, which includes the embedded Linux to the SmartphoneHarvester's platform support list. This new addition increases the number of appliances that can be benefited by the use of our proposed toolkit.

6.2.1 Raspberry Pi

The Raspberry Pi is a credit-card-sized single-board computer developed in the Cambridge University targeting the education of children on computer science. Even though this was the initial intention, the board has become very popular because of its capabilities and interconnectivity. A CPU of 700 MHz, SDRAM of 512 MB, hardware accelerated graphics, HDMI output and a number of low-level peripherals are ideal for creating embedded solutions for robotic automation, smart home control and entertainment. The most important feature that was used in our project was the low-level peripherals. Through the GPIO (General Purpose Input Output) pins it provided digital input and output where its simplest application can be to control LED lights. Also the I^2C bus (Inter-Integrated Circuit) can be used to accept a number of external hardware connected in parallel. In terms of power consumption, it requires 5 volt input with 700mA. This amount of power can be provided by a pack of four rechargeable batteries of 1.2V each, for an hour of normal use. The aforementioned features are accessible and programmable through the custom Linux build, the Raspbian "wheezy" which is an optimized version of Debian with LXDE windowing server.

6.2.2 SmartCube design

The SmartCube is a Smart Object created by a Plexiglas cube in which a Raspberry Pi is embedded. A small development board was used to solder an accelerometer sensor, a magnetometer sensor, 4 LED lights and the input for power supply through batteries. This board was designed to be pluggable to the pin interface provided by the Raspberry Pi enabling the above features. Furthermore, the network connection was provided by the use of a USB WiFi dongle. The Raspberry Pi got mounted on a removable Plexiglas chassis located in the middle of the SmartCube. On the other side of the chassis, the battery pack is mounted as well. After the cube closure, a power switch was placed at the removable side of the cube allowing the toggling of the cube's power state. In Figure 27: SmartCube initial

design, this setup can be seen as initially designed, before the implementation. This primary design was extended upon its implementation only by adding labels at four sides of the cube. Each label corresponds to the different functionality that the cube will be supporting for the house control. Those labels were added to the sides of the cube from the inside. When the lights inside the tube turn on, the pictures printed at the labels are made clearer.

Summarizing the final capabilities of the SmartCube, it is a network-enabled smart object that can be portable because of the use of batteries and WiFi network connection. It contains an accelerometer sensor and a magnetometer sensor to detect movement and is able to emit light. From the software side of view, the SmartCube runs a lightweight Debian distribution.

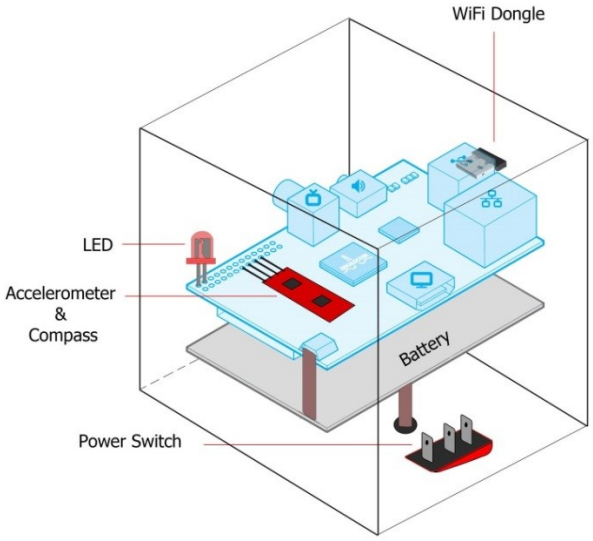

Figure 27: SmartCube initial design

Figure 28: SmartCube final implementation

6.2.3 Home interaction

The goal of this project is to control a smart home environment through multimodal interaction. The interaction will be executed through the use of the SmartCube, which was designed and implemented in the previous section, and smartphones. The rest of the section will first present the home appliances that will be controlled, followed by the supported interaction techniques.

The project will be applied to a Smart Home environment. This implies that the environment includes appliances that can be remotely controlled. The controls are provided by already implemented services that are all available through the Famine middleware. The list of the controlled devices will be also a list of the Famine services that were used.

- *Lights*: At the ceiling there are 9 lights mounted. Each light consists of Red, Green and Blue lights which can be controlled to provide a complete color range. The Light service provides the ability to set the color of each light. The color is set by providing its RGB value to the service along with the name of the specific light. Thus, any light can be separately controlled.
- *Television*: The television is controlled through TVService. The service supports the basic operations that are needed for controlling a TV. It can toggle the TV's power state, change the channel to the next and previous, and control the sound by increasing or decreasing the sound level.

- *Window Shade*: The room contains one window shade which is controlled by the Window Service. Through the service, it is possible to open or close the shades automatically.
- *Door*: The entrance door of the room. It is controlled by the Door service which can open or close the door when requested.

The above controls will be executed through the interaction with the smartphones and the SmartCube by the use of the SmartphoneHarvester toolkit. The SmartCube server that will be running on a computer will be responsible for receiving the data from the devices. Through those data, it will be controlling the aforementioned home controls being client to all the above services. In our example, all the controlled devices are included into one room.

6.2.3.1 SmartCube Interaction

By the time the SmartCube gets powered on, a red light inside the cube starts lighting. As soon as the cube is initialized and ready to use, the light turns to green. By now, the SmartphoneHarvester application is running and is connected with a Proxy. The SmartCube server resolves the service of the newly connected SmartCube. When the service is ready, the server requests the necessary gesture recognizers and sensors to get started. The gesture recognizers that will be used are the Twist and the Shake, whereas the sensors used are the Orientation sensor and the Compass.

The Orientation sensor will be providing the side of the cube that is currently up. Each side is associated with one control, which is depicted by the label shown at the cube's sides. So the orientation of the cube is used to select the currently controlled home device.

- *Light*: The user holds the SmartCube from above and rotates it. A clockwise rotation will increase the light intensity of the room, whereas the counterclockwise rotation will decrease the light intensity. By twisting the cube towards any direction, the system selects a subset of the lights, depending on the twist direction. If any rotation is now executed, the changes will be applied to the selected lights. If the cube is shaken by the user, the light selection will return the complete room to its default lighting.
- *Television*: The user rotates the cube to place the television label to the top. If the cube is shaken by the user, the TV will toggle its power state. A twist of the cube on the right

or on the left will select the next or the previous channel correspondingly. By rotating the cube, the user can control the sound volume alike the light intensity procedure.

- *Window Shade*: The user rotates the cube to place the window shade label on the top. By a twist of the cube backwards and forwards, the window shade will fold and unfold respectively.
- *Door:* The user rotates the cube to place the door label to the top. A twist of the cube backwards and forwards, will make the door open and close.

6.2.3.2 Smartphone Interaction

The controls presented for the SmartCube can be supported by a mobile device as well. As soon as the mobile device is connected and its service is registered, the SmartCube server will enable the necessary gestures and sensors. The sensors are the Compass, Proximity and the gesture recognizers are Hover, Turnover, Swipe, Twist and Shake.

Using a smartphone as a controller is much different from the use of the SmartCube. The active service selection is made by iterating the services one by one. So, a twist to the right will select the next service to be controlled and a twist to the left will select the previous service. If the device is turned over, any selected service is deselected. Thus, the mobile device deactivates the control. To regain control, the service selection procedure is executed once more. The service specific interactions defined will be presented.

- *Light*: once the light service is selected, the user rotates the device clockwise to increase the light intensity and counterclockwise to decrease the light intensity. The selection of the light subsets is done by swiping. The direction of a swipe is used to select the light subset that will be controlled. The selection returns to its default by shaking the device.
- *Television*: The power state of the television is toggled by placing the user's hand in front of the face of the smartphone for 3 seconds (hover gesture). The rest of the controls are operated through swiping. Swiping up and down controls the volume. Swiping right and left controls the TV channel.
- *Window Shade*: a swipe upwards will fold the shades and a swipe downwards will unfold the shades
- *Door*: a swipe towards back will open the door and one towards front will close it.

A list of all the controls and their correspondence to interaction is provided in Table 3: List of interaction correspondence between SmartCuve and smartphone.

Command	SmartCube	Smartphone
Service selection	Orientation	Twist left/right
Light intensity	Rotation	Rotation
Light group selection	Twist (left, right, back, front)	Swipe (left, right, back, front)
Light default selection	Shake	Shake
TV power toggle	Shake	Hover
TV channel next	Twist right	Swipe right
TV channel previous	Twist left	Swipe left
TV volume up	Rotation clockwise	Swipe up
TV volume down	Rotation counterclockwise	Swipe down
Shade fold	Twist back	Swipe up
Shade unfold	Twist front	Swipe front
Door open	Twist back	Swipe back
Door close	Twist front	Swipe front

Table 3: List of interaction correspondence between SmartCuve and smartphone

7 Evaluation

7.1 User Evaluation

7.1.1 Set-up and Participants

The evaluation process was held inside FORTH's AmI Facilities in a room that contained a meeting table and a big TV screen at the one end of the table. One projector is placed over the table, projecting on the table's surface. The assessment took place during the daylight making the projected information not visible. Thus the projector was replaced by a computer screen that got positioned on the table. This setup was used for the user experience assessment. Moreover a laptop computer got utilized for the gesture recognition rate evaluation. The mobile device that was used throughout the process was a Samsung Galaxy SII mini with NFC support running Android 2.3.6.

A total of 11 volunteers participated in the evaluation, 2 women and 9 men. Their age varied from 25-30. All of them were experienced or expert computer users and each one had in possession a smartphone prior to the evaluation. Even though they were familiar at operating a smartphone, they had never utilized it as a means of executing gestures.

7.1.2 Evaluation Goal & Process

The goal of the evaluation is divided into two sections. First, the recognition rate of each gesture will be measured. Secondly, the user experience will be assessed through the use of the Presentation Table application (section 6.1.5) whose interaction is held completely via gestures generated by the mobile phone. The process that will be described was executed once per person to avoid interference with other participants. Thus the results are clearly objective. Each person was advised to think out loud stating their experience and thoughts.

For the first step of the evaluation, a dedicated application was developed. This application will be referenced as the gesture evaluator. It goes through sequentially the complete list of supported gestures, showing on the screen the currently selected gesture. The sequence is as follows: Cover, Hover, Pick Up, Shake, and Swipe (Left, Right, Front, Back, Up, Down), Turn Over, Twist (Right, Left, Front, Back). At the successful recognition of each gesture, the user is informed of the success and it progresses to the next. Any failure is not shown, even though it is recorded in the log files along with the successful tries. During this process, all the gesture recognizers are enabled. This approach was decided in order to log

the conflicts between gestures. That is, the gestures that get falsely recognized in the process of executing a specific gesture.

The evaluation process starts by briefly explaining to the participant the concept of mobile human-computer interaction. It was followed by informing the users about the goals of the evaluation. Next, the instructor shows and explains each gesture by the use of the aforementioned gesture evaluator application that iterates the gestures and logs the tries. By the end of the gesture sequence, the user is requested to execute the same operation through the gesture evaluator. Thus, the user gets familiarized with the gestures. The evaluation process continues with the user experience assessment through the Presentation Table. It got described to the user that the previously executed gestures would be as the only means of interacting with the meeting application. The interaction contains 11 steps which were executed by the instructor first and consequently by the users. The user success rate for each task was recorded by the instructor. Finally, each user was requested to fill a system usability questionnaire.

7.1.3 Results

7.1.3.1 Gesture Recognition Rates

The presentation of the results will begin with the gesture recognition rates that were extracted by the first step of the evaluation. Through the Gesture Evaluator application, the users executed sequentially all the gestures. In case the user was not successful within two tries, the instructor was giving tips. The tries executed for each gesture were logged whether they were successful or not. Each log entry contained a timestamp that was measured in milliseconds. The timestamp was used to determine if one user movement has generated multiple recognitions. Any pair of gestures that differ in time of recognition less than 800ms, they are hypothesized to be part of the same user's movement. Thus the gesture conflicts and overlapping will be extracted. Hence, all the supported gesture recognizers are enabled in parallel.

Examining the data retrieved by the above process, we can identify tendencies. 10 out of the 15 different gestures were successfully recognized by all the participants by their first try as it can be seen in Table 4: Gesture recognition rates. The gestures that repeatedly had false recognitions were mostly because of the required movement that had to be executed. We will focus on the case of Swipe and Twist Right/Left gestures. Note that all the

participants were holding the mobile device by their right hand with the screen of the device facing at the user.

- Swipe: This gesture was considered by the users that require too intense physical movement to get recognized. Though, as stated by a participant, may be the intense movement is acceptable since it is avoids false positives. Apart from that, the Swipe gestures in the enlisted six directions were successfully recognized and easily adoptable by all users. The worst recognition rates are encountered at the Swipe Down with an average of 2.45 repetitions before success. Interestingly enough, in the 55% of those cases of false recognition, a Swipe Back was generated instead. Moreover the Swipe Front and Back had intermediate results since, as stated by the users, the forward and backward horizontal movement was not physical. Especially in Swipe Back gesture, the most common issue was that the users move horizontally forward their hand as a preparation, which resulted the recognition of a Swipe Front.

Gesture	Average Repetitions to Success	Deviation
Cover	1	0
Hover	1	0
Pick Up	1.18	0.3
Shake	1	0
Swipe Right	1.18	0.33
Swipe Left	1	0
Swipe Front	1.45	0.66
Swipe Back	1.55	0.7
Swipe Up	1	0
Swipe Down	2.45	1.3
Turn Over	1	0
Twist Right	2.64	1.45
Twist Left	1.73	0.93
Twist Front	1	0
Twist Back	1	0

Table 4: Gesture recognition rates

- Twist Right/Left: The Twist Right and Left were the gestures worst performed. This fact can be partially explained by the way the users were holding the device. By holding the mobile device with the right hand and the screen facing the user, a twist to the right or left requires the rotation of the whole hand. Also, by rotating the

mobile device 90 degrees on the left or right though a twist movement, it was seen that the users were also rotating subconsciously the device backwards. 8 out of the 11 participants have reproduced this, resulting to recognition of a Twist Back instead of a Twist Right of Left. The Twist Front and Back where 100% successful probably because the users had become already familiar with the Twist concept.

Moreover, by the results we see that two gestures prevalent in false recognition cases. Swipe and Shake detect movements thus if they are used in conjunction with Twist, special case has to be taken. If Twist Right and Left gestures are to be utilized with Swipe gestures, it is advisable to not use Swipe Up and Down since the later were among the prevalent false recognitions. Regarding the Shake gesture, it is possible to get recognized during the Twist execution since Shake is also capturing oscillation movements.

Among the general comments of the users are included the following;

- The Hover and Cover gestures have to switch naming to be more intuitive
- The Twist back has been considered inconvenient by 2 participants.
- All users agreed that after a very short training, they were able to execute successfully most of the gestures.
- Also a participant stated that the Pick Up gesture was not intuitive.

7.1.3.2 User Experience – Presentation Table

The second step of the conducted evaluation was to assess the user experience in a system that is solely controlled by mobile-based gestural interaction. For this purpose, the Presentation table shown in section 6.1.5 was used. Its functionality only contained the gestural interaction leaving out the pen interaction. This step of the evaluation was executed immediately after the gesture recognition rate step. Thus, the users were familiar with the gestural system and were ready to use them in a real application.

For this step, a total number of 11 tasks were defined. The participants were requested to execute each step sequentially. Meanwhile, the instructor kept notes of the participant's success in each task. The task execution is considered Successful if the participant was able to execute the task through at most 2 tries without any assistance. If the participant failed in the second try, assistance was given. A successful execution through assistance is considered Partial Success. Any case of more than 3 failed tries was considered a Failure.

Before the participant started the execution of the steps, the complete procedure was shown by the instructor. That was because the goal was not to evaluate the Presentation Table for its design and intuitiveness in general, but to assess the use of gestures through mobile devices as a means of interacting with it. In Table 5: Presentation table user tasks, the task list is presented.

User Task List	
Task 1:	Place the mobile device on the Table. Unlock the phone and touch the NFC tag. Select the presentation file named amidemo.ppt.
Task 2:	Open the presentation file and Go to the 3^{n} slide whose title is «New Potential».
Task 3:	Go to the second slide.
Task 4:	Close the presentation.
Task 5:	Select and Open the video file.
Task 6:	Pause the video execution.
Task 7:	Continue the video execution.
Task 8:	Close the video
Task 9:	Select the image file.
Task 10:	Close the image file.
Task 11:	Logout from the system.

Table 5: Presentation table user tasks

To evaluate the results of the process, the success rates of each task from all the users were combined. To generate a single metric of the overall user success rate, one point was given to each Successful task, half point to each Partially Successful task and finally no point to the Failures. The sum of all the above, divided by the overall number of all the tasks of all the users gave us the required value.

Number of Successful tasks (TS): 86, Number of Partially Successful tasks (PS): 27

Number of Failed tasks: 8, Total Tasks (TA): 121

Total success rate = (TS + (PS*0.5))/TA => (86 + (0.5*27)) / 121 = 82%

By the above procedure, the user success rate is calculated to be 82%. If the user success in each individual task is examined (see Diagram 1: User Success Rate in Presentation Table), it is made clear that each task contained Twist gesture had increased failures. That was every time it is was requested to the user to open or close a file. This comes in partial contradiction with the participants since three of them preferred the Twist gesture over the Swipe especially when the user is sitting. Moreover, the selection of the logout method turned out to be wrong. The reproduction of two Shake gestures in a small amount of time requires too much physical effort. Instead it was proposed by 4 users that it would be more convenient it the logout process was triggered by turning over the mobile phone and leaving it to the table. Moreover, the use of the Swipe gestures, even though they were considered too intense, had better results. Especially, as stated by the participants, if the meeting attendee is standing up.

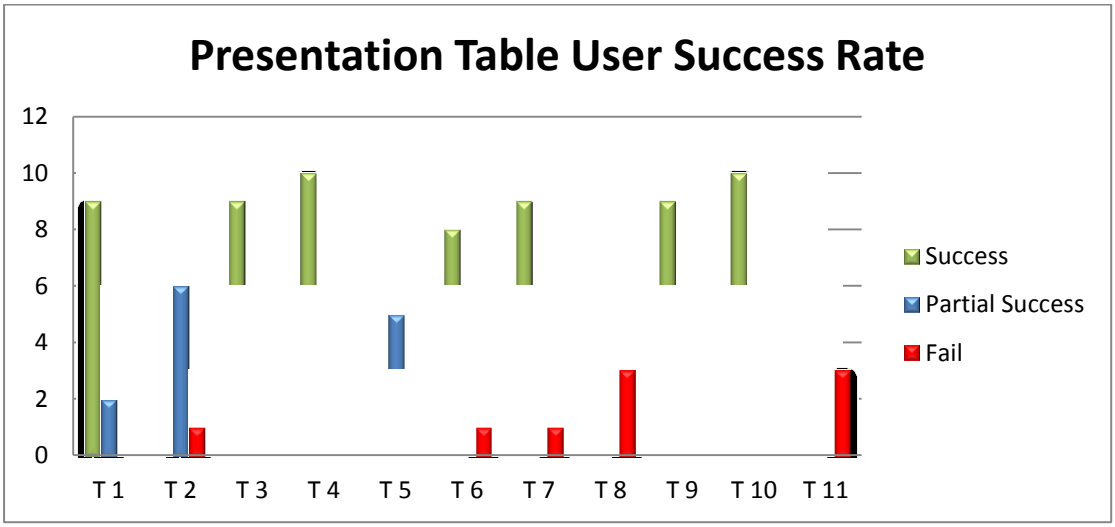

Diagram 1: User Success Rate in Presentation Table

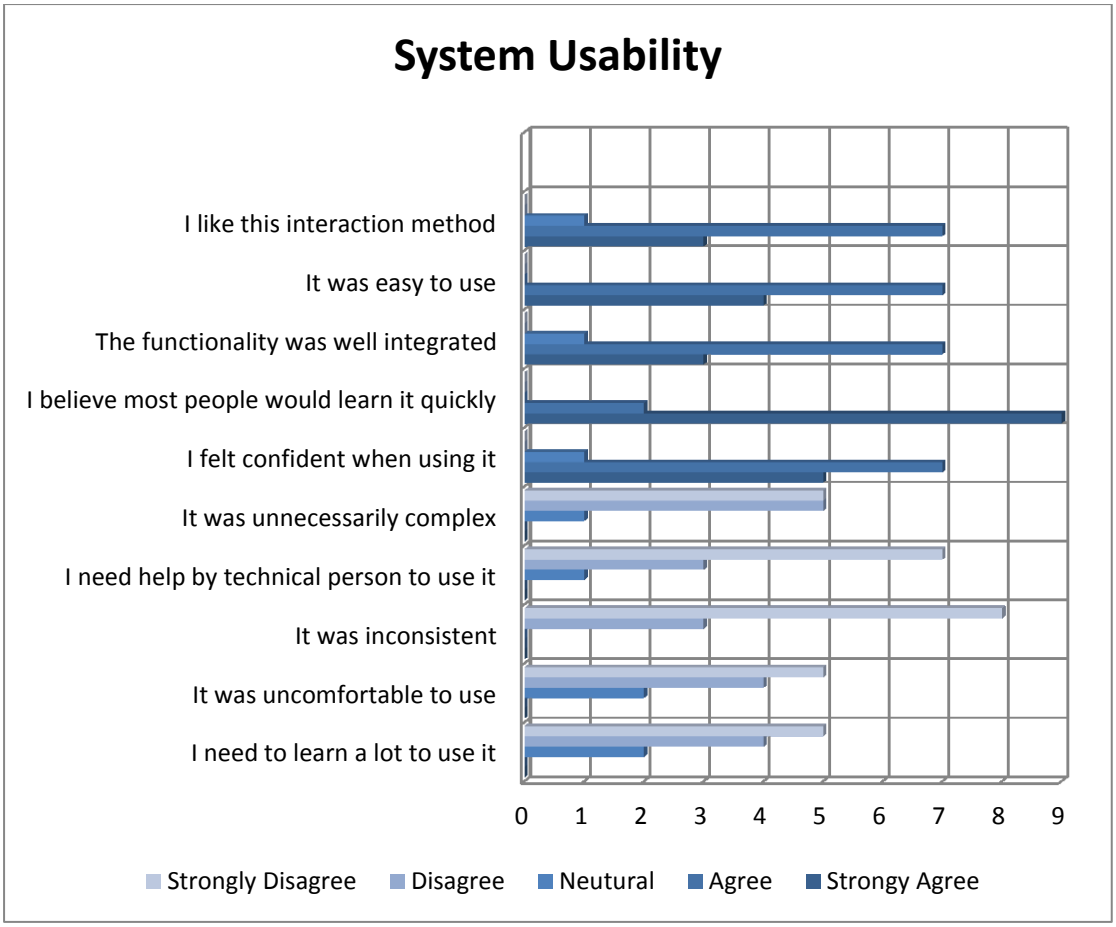

Diagram 2: Questionnaire results

Finally each participant has filled a questionnaire regarding the system usability through the proposed gesture-based interaction. By examining the results of the questionnaires through Diagram 2: Questionnaire results, the system usability was measured to be 84.77%. The majority of the participants liked the proposed interaction and would likely use it again. It is very interesting the fact that the interaction method was identified as easily adoptable and quick to learn. On the other hand, the acceptance of this interaction method could be wider if it was less uncomfortable because of the intense Swipes and the unrecognized Twists.

7.2 Multiplatform Support

As stated multiple times in this thesis, a very important feature of the SmartphoneHarvester toolkit was its multiplatform support. In this section we will provide a number of screenshots extracted from the actual usage of the toolkit in different platforms (Figure 29, Figure 30, Figure 31, Figure 32). The screenshots will be used as a proof of the application's execution in the different platforms. This section is completed with a table

stating the support of each toolkit's feature dependent to the operating system it is running at.

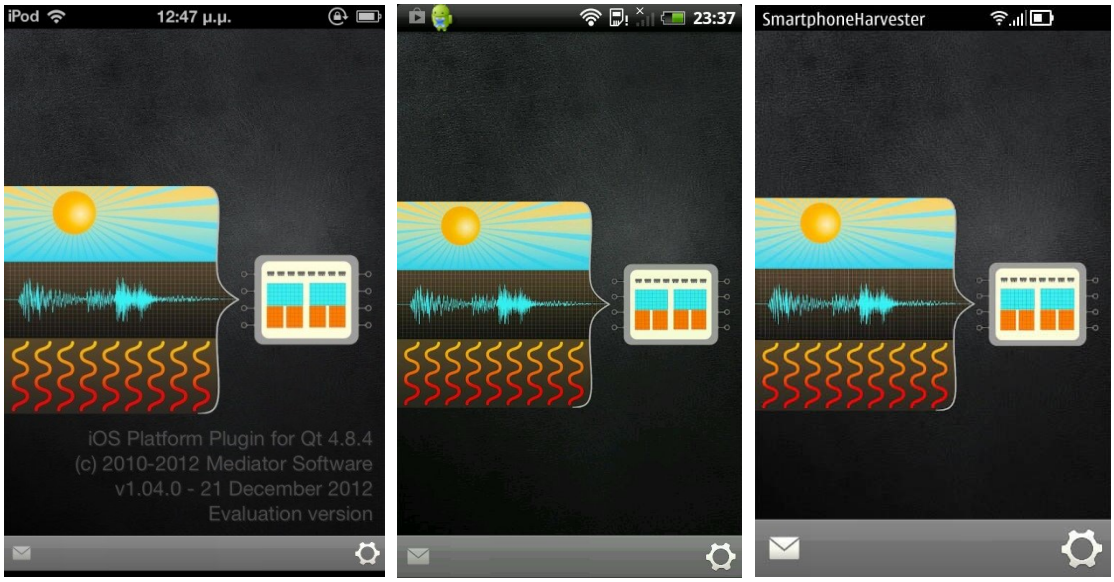

Figure 29: Multiplatform Support (iPod, HTC Desire HD, Nokia 701)

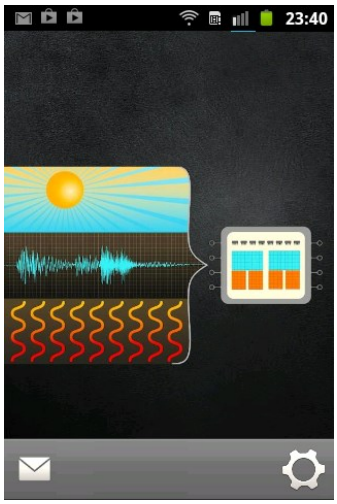

Figure 30: Samsung Galaxy SII mini

Figure 31: Meego emulator

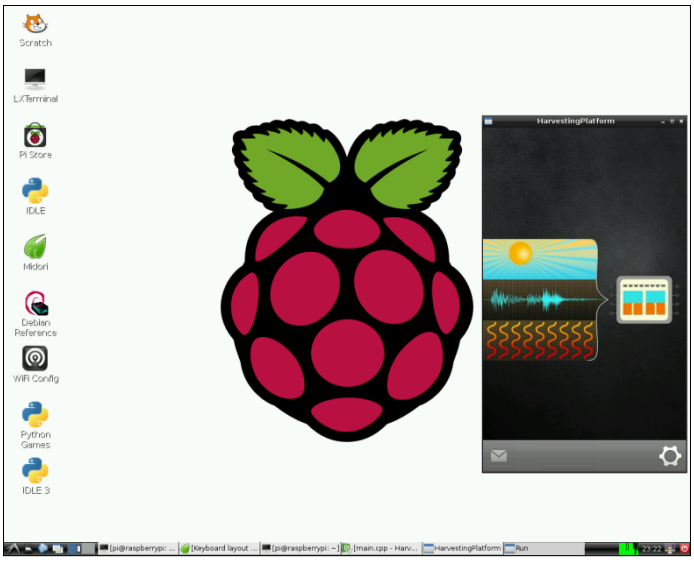

Figure 32: Raspberry Pi Raspbian "wheezy"

Resource	Android	iOS	Symbian	Meego	Linux
Sensors					
Gestures					
GPS					
NFC					
Sound					
File System					
Command Execution					
Device info					

Supported	Partially Supported	Planed to be supported	Not Supported

Figure 33: Multiplatform Feature Support

8 Conclusion

8.1 Summary

This thesis has presented an approach of integrating the smartphones into the Ambient Intelligence environments. The exposure of the feature-rich smartphone capabilities was the goal in order to further exploit them through external applications, always in a Smart Environment.

The SmartphoneHarvester toolkit was designed and implemented. Prior to the actual implementation, an extensive research was conducted to establish a solid understanding of the related work approaches. Based on the obtained information, the system's design and architecture has been defined in regard to the multiplatform support and its usage in Ambient Intelligence environments. This design created the foundations of the Service Orientated Architecture that had to be followed as imposed by the use of the Famine middleware.

Having established the basic architecture on which to build upon, the implementation of the SmartphoneHarvester toolkit was presented. The implementation started by the definition of the communication protocol between the SmartphoneHarvester mobile application and the Proxy. It was defined as a text based protocol encapsulated into JSON objects for easy manipulation. Based on this protocol for information exchange, the SmartphoneHarvester implementation was analyzed. The Core module was firstly presented because it would be the central module with which the rest of the application components would communicate. The analysis was continued the Socket Handler for the connection and message exchange handling. Most of the exposed resources were handled by the Sensors module containing all the sensors, the GPS, the NFC and the gesture recognizers. The rest data resources were handled by the File System manager, the Audio manager, the Command Executer and the Device info manager. To the other side of the connection, the Proxy application was created to interconnect the mobile device (smartphone world) with the middleware (Ambient Intelligence world). The Proxy analysis contained the description of the defined Famine services that would expose the information.

When designing the toolkit, a number of goals were set. To explore whether those goals where achieved, two applications were created. The SmartCube project had as a result the creation of a Smart Object which through the SmartphoneHarvester toolkit and the

appropriate client application would be able to control Smart Home appliances through physical interaction. Moreover, the “A day at work” scenario was presented as set of applications that integrate the use of smartphones into everyday operations that are taking place in an office environment.

This thesis contains parts that are only technical, though the final usage of the application will be made by simple end users. Thus it was interesting to see their opinion on the proposed enhanced interaction through their smartphone. For this reason an evaluation took place with 11 participants that had as a goal first to measure the gesture recognition rates along with the user ability to successfully reproduce the proposed gestures. Next, the Presentation Table application was used to conduct an evaluation regarding the user acceptance and the system usability when the interaction means is restricted only to the gestures generated by the SmartphoneHarvester toolkit. The results were encouraging since the user success rate on the executed tasks was 82% and based on the users statements the system was useful, the users were likely to reuse the system on other situations and the training needed was minimal. The downside was that some of the gestures required too much physical effort though through the statements it was accepted as a way of not generating false positives though more “sensitive” algorithms.

8.2 Contributions

To examine the contributions of this thesis, we will go one by one the goals that were set in section 3. Each goal that was achieved will be presented, justifying the statement with the methodology that was used in each case. Consulting the section 3, the objectives are divided into two categories. Those that have to be met regarding the user experience and those that have to be met for the developers. The enlisting will start by examining the objectives for the developer experience, followed by the user experience goals.

- *Support complete integration with ambient intelligence environments*: The integration of the toolkit in the Ambient Intelligence environments was made through the Famine middleware. Any smartphone gets connected with the Server and the Server registers a service instance of SmartphoneHarvester to the middleware.

- *Easily exploitable by other applications:* the middleware clients are able to make use of the smartphone capabilities through a Service Oriented Architecture.

- *Have comprehensible Application Programming Interface (API)*: Any Famine service, in order to be used by other applications, it has to define its own interface through the Interface Description Language. So the service functionality is expressed in a higher level language, independent from the programming language that will be used. Also the SmartphoneHarvester service contains clear and intuitive functionalities. Most of the data that could cause confusion were defined as Enumerations. Thus, the complete list of values is always available to the developers and provides a broader and more in depth overview of the complete service functionality.

- *Compatible with multiple programming languages to cover a respectable range of developers*: The SmartphoneHarvester, by being interconnected with the middleware, it inherits the advantages of the middleware. An arbitrary number of clients can be developed in the language of their choice, as long as that programming language is supported by the middleware. In our case, the Famine supports C++, C#, Java, Flash and Python which are more than adequate to cover a large number of developers.

- *Provide remote control of its components*: The SmartphoneHarvester service includes functions that place requests to the mobile device, either waiting for an answer or not. For example, the request of the enabling of a sensor or the request for file retrieval. Those examples constitute the remote controlling of the smartphone

- *Is easily extendable for future additions:* As presented in section 5.2 (the smartphoneharvester implementation), all the modules that sum up the SmartphoneHarvester are isolated by each other. So the addition or the removal of any of the modules would not affect the rest of the application. Moreover, the communication protocol can be easily extended by the addition of some new enumeration values. Finally, the Famine service can be extended, keeping the compatibility with the previously created clients as long as no functionality is removed.

- *Provide abstraction across the different mobile operating systems:* The abstraction was handled by 2 aspects. First, the usage of the middleware is providing seamless interaction to the clients regardless the smartphone. Secondly, on the mobile application implementation, Qt framework was as en internal middleware for the toolkit. A porting layer between the various operating systems and a high-level API in

C++ programming language. Thus the application was written once and was deployed into 5 different platforms.

- *Support functionalities that could satisfy the requirements of different research fields:*
    - *Context Aware systems:* The context aware systems mainly target at acquiring as many data as possible in order to have accurate activity or personality recognition. The proposed toolkit supports inertial sensors (accelerometer, gyroscope), geo-location sensors (magnetometer, GPS, compass), environmental sensors (Light, Ambient Light, Proximity, Orientation). Moreover, device data are made available that include a Uniqie Intentification Number for the identification of a device. In CenceMe [6] that is done by keeping the MAC address through in our proposed way it is more comprehensible. Consiquently, the operating system is provided along with the current language, the lock status, the charging status, the profile selection and two lists regarding the supported sensors and gesture recognizers. The sensors can be used for activity recognition. The device status for application adaptation. For example, the text language of an application could change according to the currently selected language of the phone. Also the profile state could be used to deactivate non-important alerts when for example the smartphone is in Silent mode.
    - *Gesture-based systems*: By the support of the 7 gesture recognizers, the toolkit can be used to create gesture-based systems. For example, the I'm Home [40] project could also be implemented through our toolkit moving the decision making to the computer rather than on device.
    - *NFC systems*: Both NFC read and write operations are supported covering the only prerequisite for an NFC-based system.

- *Introduce the availability of technologies that are existent though remain unused:* Throughout the reviewed projects in literature, none of them was utilizing and exposing the file system of the mobile devices. We have proposed the complete file system exposure through Browse, Retrieve, Upload and Delete commands. All the file transfers are held in the most effective way by creating dedicated connections between smartphone and Proxy to transfer the file data. Finally the Delete operation is supported only for the protected folders that were introduced by this toolkit. The virtual folders

with path "~" and "~/Office" can be used by any application to store data without having to depend on the underlying file system structure.

From the users' point of view, the system should;

- *Incorporate functionalities that can be easily used:* The mobile application, when it comes to the user interaction, contains only 3 specific operations. First comes the gesture recognition through physical movements. Consulting the user evaluation that was conducted through the results in system usability questionnaire, the users stated that the proposed interaction method was by 95% easy to be learned quickly and by 84% easy to use. Moving along to the graphical user interface, it provides only two functionalities. That is the network setup which only requires an IP address and a Port name and the composing of the message to be left through an NFC. The messaging functionality is self-driven. By the time an NFC tag is touches, the message is sent, the message box is cleared and the UI returns to its idle home screen.

- *Be intuitive with low cognitive effort:* Here the above statements can be applied as well. Also by 93% the gesture movements were considered consistent to what they denote.

- *Be non-intrusive:* The application that was developed was designed to be working at the background throughout its life. Thus it would not affect the user experience throughout the default smartphone usage. This goal has succeeded in Symbian and Android though it failed in iOS. In iOS platform, the proposed toolkit could not be used at the background since its network connection was instantly closed. This could either be a platform specific limitation, or an omission of the toolkit's implementation.

- *Support multiple current and future platforms to include the biggest market share slice:* A very important feature of SmartphoneHarvester toolkit is that tested on Android, iOS, Symbian, Meego, Linux. This could cover most of the smartphone market share, though this list can be expanded by the evolution of Qt framework. Qt framework is the "middleware" that was used for the multiplatform support so the supported by it platforms are inherited to our application as well. For example the BlackBerry 10 platform is already supported by Qt but our application was not tested lacking a deployment device.

- *Seamless interaction across platforms:* The User Interface was designed to be consistent between the platforms so the 2 interactions regarding the GUI are seamless. As to the gesture recognition, the only difference to the user experience is that on each device the inertial sensor sensitivity varies. In most of the case, the variance is small thus negligible. Never the less, one of our five deployment devices (HTC Desire HD) has much less sensitive inertial sensors which resulted to requiring to apply more physical effort to execute a gesture.
- *Be easy to install:* This step was not investigated into this thesis. Though the deployment on Android and Symbian because of the openness of each platform will be seamless. On the other hand, the iPhone application ,to be used to mobile devices other than the one used for development, needs to be uploaded to Apple's App Store, pass their Quality Assurance and then be available. This procedure was not followed and is left for future work.
- *Be easy to configure:* The only configuration required in our toolkit is the one time setup of the IP address and Port number of the running Proxy that will be connecting the mobile device to the middleware. This information are stored and used upon the execution of the application till they get manually changed.

So, reviewing one by one the goals that were set, almost 100% of them are achieved.

8.3 Future Work

Based on the conclusions, the evaluation and the experience that was gained during the implementation, a small number of key aspects can be identified that are already planned to be addressed.

The most important future addition will be to complete the support of all the toolkit's features throughout all the platforms. This includes the Audio Manager and NFC Write operation module in Android. For the iOS platform, Command Executer and the Device Information Manager have to be implemented to achieve seamless developer experience in all platforms.

Regarding the gesture recognizers, they first need to support adaptation to the device's sensor sensitivity levels. Moreover, further evaluations have to take place with different mobile devices to extract patterns in the false gesture recognition. This will help us

evolve the gesture recognizers eliminating the conflicts between simultaneously recognized gestures.

The NFC specification defines the NDEF message types, which have to be separately supported by any NFC enabled application. In our case we support only Text and URL messages to be ready from an NFC for simplicity.

Ultimately, the mobile application has to become an integral service of the mobile device. On Android platform, as soon as the screen is locked, the network connectivity is disabled. Also on iOS the application is connectable only when it is in the foreground. This is why this toolkit has to be redesigned to become a background service in all the supported platforms. Thus it will be always running, continuously providing services.

9 References

1. Han L., Jyri S., Ma J., Yu K., Nokia Research Center (2010), “Research on Context-Aware Mobile Computing”
2. Hasu T. (2010), ”ContextLogger2 - A Tool for Smartphone Data Gathering”
3. SQLite3 site
4. Bl´ azquez Gil G., Berlanga A., Jos´ eM.Molina (2012), “InContexto: Multisensor Architecture to Obtain People Context fromSmartphones”
5. Azizyan M., Constandache I., Choudhury R. R. (2009), “SurroundSense: Mobile Phone Localization via Ambience Fingerprinting”
6. E. Miluzzo, N. Lane, K. Fodor et al., (2008) “Sensing meets mobile social networks: the design, implementation and evaluation of the CenceMe application,”
7. J. Chon and H. Cha, (2011) “LifeMap: a smartphone-based context provider for location-based services”
8. A. Dey and G. Abowd, (1999) “Towards a better understanding of context and context-awareness”
9. P. Prekop, M. Burnett, (2002) “Activities, Context and Ubiquitous Computing”
10. Robert Neßelrath, Chensheng Lu, Christian H. Schulz, Jochen Frey, Jan Alexandersson, “A Gesture Based System for Context – Sensitive Interaction with Smart Homes”
11. D. J. Cook, J. C. Augusto, V R. Jakkula, (2007) “Ambient Intelligence: Technologies, Applications, and Opportunities”
12. J. Bohn, V. Coroama, M. Langheinrich, F. Mattern, M. Rohs, (2005) “Social, economic and ethical implications of ambient intelligence and ubiquitous computing”
13. R. Lucky, (1999) “Everything will be connected to everything else.”
14. M. Weiser, (1991) “The Computer for the 21st Centrury”
15. E. Zelkha, B. Epstein, (1998) "From Devices to "Ambient Intelligence": The Transformation of Consumer Electronics", (http://www.epstein.org/brian/ambient_intelligence/DLR%20Final%20Internal.ppt)
16. M. Charalampidou, S. Mouroutsos, G. Pavlidis, (2012) “Identifying Aspects of Ambient Intelligence Through A Review of Recent Developments”
17. E. B. Snaders, (2002) “From User-Centered to Participatory Design Approaches”
18. D. M. Dubois, (1998) “Introduction to Computing Anticipatory Systems, International Journal of Computing Anticipatory Systems”

19. A. Butz (2010) “User Interfaces and HCI for Ambient Intelligence and Smart Environments”
20. Kettebekov, Sanshzar, and Rajeev Sharma (2001). "Toward Natural Gesture/Speech Control of a Large Display."
21. M. Vassiliou, V. Sundareswaran, S. Chen, R. Behringer, C. Tam, M. Chan, P. Bangayan, and J. McGee (2000), “Integrated Multimodal Human-Computer Interface and Augmented Reality for Interactive Display Applications”
22. S. Oviatt, (2002) “Multimodal Interfaces”
23. Garrett, Jennifer Tumlin, et al. (2011) "Using Speech Recognition Software to Increase Writing Fluency for Individuals with Physical Disabilities."
24. E. J. Pauwels , A. A. Salah , R. Tavenard (2007) “Sensor Networks for Ambient Intelligence”
25. H. Do, H. Silverman, (2007) “A fast microphone array SRP-PHAT source location implementation using coarse-to-fine region contraction (CFRC)”
26. G. Galatas, S. Ferdous, F. Makedon (2013) “Multi-modal Person Localization And Emergency Detection Using The Kinect”
27. D. Grammenos, Y. Georgalis, N. Partarakis, X. Zabulis, T. Sarmis, S. Kartakis, P. Tourlakis, A. Argyros, and C. Stephanidis, (2009) “Rapid Prototyping of an AmI-Augmented Office Environment Demonstrator”
28. N. Kubota (2007) “Human Detection and Gesture Recognition Based on Ambient Intelligence”
29. M. Karem and M. Schraefel, (2011) “A taxonomy of Gestures in Human Computer Interaction. In Transactions on Computer-Human Interactions”
30. V. Buchmann, S. Violich, M. Billinghurst, A. Cockburn, (2004) “FingARtips – Gesture Based Direct Manipulation in Augmented Reality”
31. P. Stefan, (2009). “Ubiquitous Computing Smart Devices, Smart Environments and Smart Interaction”
32. L. Chittaro, (2010) “Distinctive Aspects of Mobile Interaction and their Implications for the Design of Multimodal Interfaces”
33. S. Love, (2005) Understanding Mobile Human-Computer Interaction.
34. K. Huang, (2009) “Challenges in Human-Computer Interaction Design for Mobile Devices”
35. S. Kratz, F. Hemmert, M. Rohns, (2010) “Natural User Interfaces in Mobile Phone Interaction”

36. J. Kjeldskov, K. Cheverst, M. de Sá, M. Jones, R. Murray-Smith (2012) “Research Methods in Mobile HCI: Trends and Opportunities”
37. A. Wilson, N. Oliver, (2005)”Multimodal Sensing for Explicit and Implicit Interaction”
38. A. Dix, J. Finlay, G. D. Abowd, R. Beale, (2003) “Human Computer Interaction. 3rd Edition”
39. M. Johnston, L. F. D’Haro, M. Levine, and B. Renger. (2007) “A Multimodal Interface for Access to Content in the Home”
40. T. Westermann, (2010) “I’m Home : Smartphone-enabled Gestural Interaction with Multi-Modal Smart-Home Systems”
41. F. Razzak, D. Bonino, F. Corno: (2010) “Mobile Interaction with Smart Environments through Linked Data”
42. B. Schilit, N. Adams, R. Want, (1994) “Context-Aware Computing Applications”
43. G. Chittaranjan, J. Blom, D. Gatica-Perez, (2011) “Who's Who with Big-Five: Analyzing and Classifying Personality Traits with Smartphones”
44. H. Turner, J. White, C. Thompson, K. Zienkiewicz, S. Campbell, D. C. Schmidt, (2011) “Building Mobile Sensor Networks Using Smartphones and Web Services: Ramifications and Development Challenges”
45. K. A. Yuksel, S. H. Adali, (2011) “Prototyping Input Controller for Touch-less Interaction with ubiquitous Environments”
46. L. Jing, Y. Zhou, Z. Cheng, T. Huang, (2012) “Magic Ring: A Finger-Worn Device for Multiple Appliances Control Using Static Finger Gestures”
47. I. Oakley, J. Park, (2007) “A motion-based marking menu system”
48. T. Park, J. Lee, I. Hwang, C.Yoo, L. Nachman, J. Song, (2011) “E-Gesture: A Collaborative Architecture for Energy-efficient Gesture Recognition with Hand-worn Sensor and Mobile Devices”
49. G. Shen, C. Peng, Y. Li, C. Shi, Yongguang Zhang, Songwu Lu, (2012) “Dita: Enabling Gesture-based Human-Device Interaction using Mobile Phone”
50. J. Ruiz, Y. Li, (2011) “DoubleFlip: A Motion Gesture Delimiter for Mobile Interaction”
51. H. Ishii, (2008) “The tangible user interface and its evolution”
52. M. Kaltenbrunner, R. Bencina, (2007) “reacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction”
53. S. Jordà, (2010) “The Reactable: Tangible and Tabletop Music Performance”
54. E. W. Pedersen, K. Hornbæk, (2011) “Tangible Bots:Interaction with Active Tangibles in Tabletop Interfaces”

55. B. Schneider, P. Blikstein, W. Mackay, (2012) “Combinatorix: Tangible User Interface that Supports Collaborative Learning of Probabilities”
56. A. Richardson, (2012) “An Overview of the Pixel Ware Project at the Oriental Museum, Durham”
57. M. Kranz, D. Schmidt, P. Holleis, A. Schmidt, (2005) “A Display Cube as a Tangible User Interface”
58. B. Schiettecatte, J. Vanderdonckt, (2008) “AudioCubes: a Distributed Cube Tangible Interface based on Interaction Range for Sound Design”
59. D. Merrill, J. Kalanithi, P. Maes, (2009) “Siftables: Towards Sensor Network User Interfaces”
60. S. Burkard, (2012) “Near Field Communication in Smartphones”
61. R. Hardy, E. Rukzio, P. Holleis, M. Wagner, (2010) “Mobile Interaction with Static and Dynamic NFC-based Displays”
62. G. Broll, R. Graebsch, P. Holleis, M. Wagner, (2010) “Touch to Play - Mobile Gaming with Dynamic, NFC-based Physical User Interfaces”
63. G. Chavira, S.W. Nava, , J.C. Rolon, E. Rolon, T.G.Acuna, (2012) “SeTITaC: Services through Touch Interaction and Tagging Context ”
64. J. Jalkanen, (2005) “User-initiated context switching using NFC”
65. T. Pering, Y. Anokwa, R. Want, (2007) “Gesture Connect: Facilitating Tangible Interaction With a Flick Of The Wrist”
66. F. Resatsch, U. Sandner, C. Höchst, D. Michelis, T. Schildhauer, J.M. Leimeister, H. Krcmar, (2007) “Easymeeting – Prototyping Ubiquitous Computing in Offices”
67. R. Ballagas, M. Ringel, M. Stone, J. Borchers, (2003) “iStuff: A Physical User Interface Toolkit for Ubiquitous Computing Environments”
68. R. Ballagas, F. Memon, R. Reiners, J. Borchers, (2007) “iStuff Mobile: prototyping interactions for mobile phones in interactive spaces”
69. B. Kaufmann, (2010) “Design and Implementation of a Toolkit for the Rapid Prototyping of Mobile Ubiquitous Computing”
70. Y. Georgalis, D. Grammenos, C. Stephanidis, (2009) “Middleware for Ambient Intelligence Environments: Reviewing Requirements and Communication Technologies”
71. Grammenos D., Zabulis X., Michel D., Sarmis Th., Tzavanidis K., Argyros A. and Stephanidis C. (2011). Design and Development of Four Prototype Interactive Edutainment Exhibits for Museums

72. Grønbæk Kaj, Iversen Ole Sejer, Kortbek Karen Johanne, Nielsen Kaspar Rosengreen and Aagaard Louise (2007). Interactive Floor Support for Kinesthetic Interaction in Children Learning Environments
73. Murphy-Chutorian, E.; Trivedi, M.M. (2010). Head Pose Estimation and Augmented Reality Tracking: An Integrated System and Evaluation for Monitoring Driver Awareness.
74. Papagiannakis G., Singh G. and Magnenat-Thalmann N. (2009). A survey of mobile and wireless technologies for augmented reality systems
75. D. Grammenos, Y. Georgalis, N. Kazepis, G. Drossis, N. Ftylitakis, (2010) "The booTable experience: iterative design and prototyping of an alternative interactive tabletop"

10 Appendixes

10.1 List of Abbreviations

AmI: Ambient Intelligence

AmIE: Ambient Intelligence Environment

API: Application Programming Interface

GPS: Global Positioning System

GSM: Global System for Mobile Communications

HCI: Human Computer Interaction

Hi-Fi: High fidelity

HTTP: Hypertext Transfer Protocol

IR: Infrared

JNI: Java Native Interface

MEMS: Microelectromechanical systems

NDK: Native Development Kit

NFC: Near Field Communication

OS: Operating System

PImplS: Private Implementation software pattern

SNS: Social Network Service

SSL: Secure Sockets Layer

TUI: Tangible User Interface

UCE: Ubiquitous Computing Environment

UI: User Interface

UV radiation: Ultra Violet radiation

Wi-Fi: Wireless Fidelity

10.2 Famine Service Descriptions

```
module Driver{
    interface SmartphoneHarvester{
        enum Profile{
            Unknown,
            Silent,
            General,
            Loud,
            Offline,
            Powersave
        };

        enum OrientationDirection{
            Undefined,
            TopUp,
            TopDown,
            LeftUp,
            RightUp,
            FaceUp,
            FaceDown
        };

        enum AmbientLightValues{
            Indeterminate,
            Dark,
            Twilight,
            Normal,
            Bright,
            Sunny
        };

        enum TapDirection{
            X_Pos,
            Y_Pos,
            Z_Pos,
            X_Neg,
            Y_Neg,
            Z_Neg
        };

        enum Direction{
            Unused,
            Left,
            Right,
            VerticalUp,
            VerticalDown,
            HorizontalUp,
            HorizontalDown,
            Front,
            Back
        };

        enum Gesture{
            Cover,
            DoubleTap,
            Hover,
            PickUp,
            Shake,
            Swipe,
            TurnOver,
            Twist
        };
        typedef sequence<Gesture> GestureSeq;

        enum Sensitivity{
            UnusedSensitivity,
            Low,
            Medium,
            High
        };

        enum Sensor{
            Accelerometer,
            Orientation,
            Proximity,
            Tap,
            AmbientLight,
            Compass,
            Gyroscope,
            Light,
            Magnetometer,
            Rotation,
            Gps,
            Nfc
        };
        typedef sequence<Sensor> SensorSeq;

        struct Device{
            string id;
            string humanReadableID;
            string operatingSystem;
            SensorSeq supportedSensors;
            GestureSeq supportedGestures;
            string currentLanguage;
            boolean isLocked;
            boolean isCharging;
            Profile currentProfile;
        };

        struct RemoteFile{
            string name;
            boolean isFolder;
        };
        typedef sequence<RemoteFile> RemoteFileSeq;

        struct UploadFileInfo{
            string localAbsolutePath;
            string remoteStoreLocation;
        };
        typedef sequence<UploadFileInfo> UploadFileInfoSeq;
```

Figure 34: SmartphoneHarvester Service Interface (part 1/3)

```
        ////////////////////////
        ////    Methods     ////
        ////////////////////////

        boolean IsDeviceConnected();
        Device GetConnectedDeviceInfo();

        SensorSeq GetActiveSensors();
        GestureSeq GetActiveGestures();

        ////////////////////////

        void StartSoundLevelDetection(in Sensitivity sens);
        void StopSoundLevelDetection();

        void StartGestureRecognizer(in Gesture gestureId);
        void StopGestureRecognizer(in Gesture gestureId);

        //Sensitivity is used only at the following sensors:
        // Accelerometer
        // Gyroscope
        // Magnetometer
        // Rotation
        void StartSensor(in Sensor sensorId, in Sensitivity sens);
        void StopSensor(in Sensor sensorId);

        void BrowseFolder(in string folderAbsolutePath, in boolean isRecursive);
        //requires absolute paths for all the files
        //remoteContainerFolder: used when all files provided are children of a signle folder.
        //  To disable it provide an empty string
        void RetrieveFiles(in ami::StringSeq files, in string localDestinationFolder, in string remoteContainerFolder);

        void UploadFiles(in UploadFileInfoSeq files);

        //applicable only to the shared folder provided by this service.
        // Everything under directory "~".
        void DeleteFile(in string file);

        //The message will be written to the next NFC tag that will be touched
        void NFCWriteMessage(in string message);

        void RequestFileOpenAtRemoteTarget(in string filePath);

```

Figure 35: SmartphoneHarvester Service Interface (part 2/3)

```
        ////////////////////////
        ////    Events      ////
        ////////////////////////
        void Event_TapDetected(in TapDirection direction, in boolean isDoubleTap);
        void Event_ProximityReadingChanged(in boolean isClose);
        void Event_DeviceLockStatusChanged(in boolean isLocked);

        void Event_ProfileStatusChanged(in Profile profileId);
        void Event_OrientationChanged(in OrientationDirection orientation);
        void Event_AmbientLightChanged(in AmbientLightValues lightIntensity);
        void Event_GestureRecognized(in Gesture gestureId , in Direction direction);

        void Event_NfcTagDetected(in string id);
        void Event_NfcTagLost(in string id);
        void Event_NfcMessageReceived(in string id, in string message);

        void Event_CompassChanged(in double azimuth);
        void Event_LightChanged(in long value );

        void Event_AccelerometerChanged(in double x, in double y, in double z);
        void Event_GyroscopeChanged(in double x, in double y, in double z);
        void Event_MagnetometerChanged(in double x, in double y, in double z);
        void Event_RotationChanged(in double x, in double y, in double z);

        void Event_LocationChanged(in double latitude, in double longitude, in double altitude);

        //File system events
        void Event_FolderBrowsingEnded(in string rootFolder, in RemoteFileSeq selectedFiles, in boolean isRecursive);
        void Event_FileReceived(in string fileName);
        void Event_FileUploadingFinished(in UploadFileInfoSeq files);

        //////
        void Event_SoundLevelChanged(in double rmsLevel, in double peakLevel);
        void Event_ChargingStateChanged(in boolean isCharging);

        void Event_DeviceConnected(in Device newDevice);
    };
};
```

Figure 36: SmartphoneHarvester Service Interface (part 3/3)

```
7  module Driver {
8      interface SmartphoneBroker {
9
10         //contains all the successfuly registered services
11         ami::StringSeq GetActiveContexts ();
12
13         //emitted only when the new context has registered successfully
14         void Event_DeviceConnected (in string contextName );
15         void Event_DeviceDisconnected(in string contextName);
16
17         ////////////////////////////////////////////
18         //not for public use. Used internally
19         void SetContextAsAlive(in string contextName);
20         void Event_AboutToTerminate();
21     };
22 };
```

Figure 37: SmartphoneBroker service Interface

10.3 Evaluation Questionnaire

© Digital Equipment Corporation, 1986.

	Strongly disagree				Strongly agree
1. I think that I would like to use this interaction method frequently	1	2	3	4	5
2. I found the interaction method unnecessarily complex	1	2	3	4	5
3. I thought the interaction method was easy to use	1	2	3	4	5
4. I think that I would need the support of a technical person to be able to use the interaction method	1	2	3	4	5
5. I found the various functions in this interaction method were well integrated	1	2	3	4	5
6. I thought there was too much inconsistency in this interaction method	1	2	3	4	5
7. I would imagine that most people would learn to use this the interaction method very quickly	1	2	3	4	5
8. I found the interaction method very cumbersome to use	1	2	3	4	5
9. I felt very confident using the the interaction method	1	2	3	4	5
10. I needed to learn a lot of things before I could get going with this the interaction method	1	2	3	4	5