Design and Implementation of an Internet of Things Smart Meeting Application

Nikolaos – Stylianos Loumis

Master of Science in Technology Education & Digital Systems

From the

University of Piraeus

Department of Digital Systems

University of Piraeus

80, Karaoli and Dimitriou Str, Piraeus, 18534

August 2015

Supervised by: Dr Kostas Tsagkaris

AKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to Dr Kostas Tsagkaris for his understanding during this research and his willingness to step up in order for me to submit my thesis successfully. Furthermore, I would like to thank Assistant Professor Vera Stavroulaki for her motivation, knowledge, and her guidance throughout my research and writing of this thesis. I wish her and the new member of her family all the best to come.

Besides my advisors from University of Piraeus, I would like to thank Prof Klaus Moessner, Dr Stylianos Georgoulas, and Dr Michele Nati, who provided me an opportunity to join their team as intern, and gave me access to their research facilities.

I thank my family, friends, and co-workers for supporting me spiritually, especially during the hardest part of my life couple of years ago. I could not have made it without them.

Finally, I would like to thank the-soon-to-be-Dr Sylvia Agathou for her tolerance and support for me during my difficulties all these years.

DECLARATION OF ORIGINALITY

I confirm that the project dissertation I am submitting is entirely my own work and that any material used from other sources has been clearly identified and properly acknowledged and referenced. In submitting this final version of my report to the JISC anti-plagiarism software resource, I confirm that my work does not contravene the university regulations on plagiarism. In so doing I also acknowledge that I may be held to account for any particular instances of uncited work detected by the JISC anti-plagiarism software, or as may be found by the project examiner or project organiser. I also understand that if an allegation of plagiarism is upheld via an Academic Misconduct Hearing, then I may forfeit any credit for this module or a more severe penalty may be agreed.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

WORD COUNT

Number of Pages: 65

Number of Words: 12223

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

Abstract

During the past decades, a remarkable transformation in size and performance of portable electronic devices has taken place, leading to a new era where multi-core devices (phones, tablets, wearables) even in the size of a coin can offer ubiquitous computing capabilities to the end user. Meanwhile, broadband internet has become widely accessible, enabling users' and devices to be connected whenever, wherever. "Internet of Things" (IoT) is topic that emerged from this seamless connectivity that current technology has to offer. Devices and things are enhanced with intelligence by providing their data and by accessing information generated by others.

This study focused on the effects that IoT area can have in our everyday lives by examining a situation one can come across daily: a business meeting. It tried to prove that an Internet of Things Smart Meeting Application suite could help a meeting organiser to create, manage, and wrap-up a meeting, via his Android powered smartphone. Furthermore, an indoor localisation/navigation system was presented, that will escort users to the meeting venue using dynamically generated personalised invitations.

Περί ληψη

Κατά τη διάρκεια των περασμένων δεκαετιών έχει σημειωθεί σημαντική πρόοδος στο μέγεθος και την απόδοση των φορητών ηλεκτρονικών συσκευών, η οποία οδήγησε σε μια περίοδο όπου πλέον συσκευές πολλών πυρήνων (κινητά τηλέφωνα, τάμπλετ κ.α.), τόσο μικρές σε μέγεθος όσο ένα νόμισμα, μπορούν να προσφέρουν ηλεκτρονικές υπηρεσίες στους χρήστες ασταμάτητα. Παράλληλα, το ευρυζωνικό διαδίκτυο έχει γίνει ευρέως διαθέσιμο επιτρέποντας σε χρήστες και συσκευές να είναι συνδεδεμένοι οποτεδήποτε, οπουδήποτε. Το Internet of Things (IoT) είναι ένας κλάδος ο οποίος έχει δημιουργηθεί από την αδιάκοπη συνδεσιμότητα που η σύγχρονη τεχνολογία μπορεί να προσφέρει. Συσκευές και πράγματα εμπλουτίζονται με νοημοσύνη μοιράζοντας τα δεδομένα τους, αλλά και προσπελαύνοντας πληροφορίες που έχουν δημιουργηθεί από άλλους.

Η παρούσα εργασία επικεντρώθηκε στις επιπτώσεις που μπορεί να έχει στην ζωή μας ο κλάδος του Internet of Things χρησιμοποιώντας μια καθημερινή κατάσταση, όπως αυτή ενός επαγγελματικού μίτινγκ. Προσπάθησε να αποδείξει ότι μια σουίτα προγραμμάτων Έξυπνου Μίτινγκ (Smart Meeting Application) που ακολουθεί αυτά που επιτάσσει το IoT, μπορεί να βοηθήσει τον δημιουργό ενός μίτινγκ να οργανώσει, διαχειριστεί, αλλά και να ολοκληρώσει μια συνάντηση διαμέσου του κινητού τηλεφώνου του. Επιπλέον, ένα σύστημα εντοπισμού και καθοδήγησης εσωτερικού ΄χώρου παρουσιάστηκε, το οποίο αναλαμβάνει να συνοδέψει τους χρήστες στο σημείο συνάντησης. Αυτό το σύστημα χρησιμοποιεί προσωποποιημένες προσκλήσεις οι οποίες έχουν δημιουργηθεί δυναμικώς για κάθε χρήστη ξεχωριστά.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

Table of Contents

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

Table of Figures

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

Table of Acronyms

Acronym	Meaning
CVO	Composite Virtual Object
IoT	Internet of Things
ROI	Return on Investment
SMB	Smart Meeting Break
SMC	Smart Meeting Creation
SMR	Smart Meeting Recording
SMW	Smart Meeting Wrap-Up
UI	User Interface
VO	Virtual Object

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

1. Introduction

1.1 Background and Context

During the past decades a remarkable transformation in size and performance of portable electronic devices has taken place, leading to a new era where multi-core devices (phones, tablets, wearables) even in the size of a coin can offer ubiquitous computing capabilities to the end user. Furthermore, the constantly evolving microelectromechanical systems (MEMS) technologies have not only changed the dimension and precision of sensors but also their integration potential, allowing them to be installed in common mobile devices and enhancing the user experience.

Meanwhile, broadband internet has become widely accessible, enabling users' and devices to be connected whenever, wherever. "Internet of Things" (IoT) is topic that emerged from this seamless connectivity that current technology has to offer. Devices and things are enhanced with intelligence by providing their data and by accessing information generated by others. This creates a network of smart devices that communicate with each other and paves the path to develop situation aware applications and services. Those applications and services can upgrade and automate business processes augmenting efficiency and productivity while minimising the cost. However, the aforementioned devices and their created network introduce 2 basic concerns that should be taken under consideration:

1. The existing device heterogeneity, which derives from the great spectrum of devices that are connected to the internet. Minimising device heterogeneity has to be accompanied by enhancing the context awareness, the reliability, and the energy efficiency of internet services and infrastructures.
2. Provision of tailored services to different types of users in order to create new business opportunities.

This study suggests a solution that encapsulates a cognitive framework divided to 3 levels of reusable functionalities. Those are:

1. Virtual Objects (VOs) level. Virtual Objects consist of the combination of one natural object and its interconnected cognitive mechanisms, such as learning, processing, and decision making.
2. Composite Virtual Objects (CVOs) level. Composite Virtual Objects are using the services provided by the VOs. In other words, CVOs are cognitive mush-ups of the virtual objects, which target the service and application provisioning according to the users' requirements.
3. User/stakeholder level.

This dissertation describes an MSc project which aims the design, development, and implementation of an IoT Smart Meeting Application.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

1.2 Scope and Objectives

The scope of this use case is to provide meeting organisers with features and the capability of efficiently managing the whole meeting life-cycle from its organisation, to its execution and the meeting wrap-up. Furthermore, we aim to create one user friendly application to be used by meeting attendants to guide themselves towards the meeting venue.

1.3 Achievements

In addressing the issue of handling a meeting using cutting edge technologies for our benefit a Smart Meeting Applications suite was developed. All of the systems developed utilized various technologies including

- Java server application,
- Android applications
- MySQL databases
- QR code generation and scanning
- MQTT protocol
- GraphView library

1.4 Overview of Dissertation

This dissertation discusses the design, implementation, and operation of the Smart Meeting Application. It describes the process involved in developing a smart application that helps and saves time and effort to meeting organisers and participants.

This first chapter introduced the project by stating an existing problem, a methodology to alleviate it, and the scope of it.

The following chapter presents an analysis of the protocols and methodologies that were used during the life-cycle of this project.

Chapter 3 presents the project concept and the system requirements. The use cases that were taken under consideration will be broken down and explained in high details.

Chapter 4 outlines the design approach selected according to the aforementioned user requirements. High level design, database structure, and detailed functionalities are also part of this. Finally sequence diagrams help us understand the back end of the system.

Chapter 5 describes how the system was implemented and provides a detailed explanation of how it works. Furthermore, system screenshots are included.

Chapter 6 provides the conclusion of the dissertation.

2. State of The Art

2.1 Software Development Models

System software makes our lives easier, more efficient, and more productive in such a seamless manner, that we tend to forget how important it is. Nowadays, software plays, either in the foreground or in the background, an important role in almost every aspect of our lives, including crucial systems that affect our health and safety. This is why software engineering is currently under the research spotlight more than ever. Best practices of software design and implementation have to ensure that the actual software will have a positive impact on our way of living.

Software engineering literature includes various models of software development processes. Some of them define the ways of software development, while others are methodologies descriptions. In theory, the aforementioned two types of models should be similar or matched. However, in practise this does not happen: a software development team can understand the gap between what should be done and what is actually done by creating and following a software development process.

Waterfall model

Waterfall model was one of the first software development models that were introduces at the start of 1970s. The phases of this model are represented in a sequential way, like a waterfall, where one step leads to another. For each step of the model to start being implemented, firstly the previous one has to be completed. This sequential design offers a very high level view of what is taking place during the software development, and also enables the engineers to expect the following steps.

Model description

Initially system and software requirements are defined. Following that, a preliminary and detailed software design takes place. During the preliminary design, the units that will consist the software are defined, as well as their interconnections. The initial layer (the less fine grained one) includes the subsystems, while the second one includes the units inside subsystem and so on. During the detailed design, the internal structure of every software unit is defined, which practical corresponds to fragments of source code. This definition includes all the required components (algorithms, data structures etc.) in order for the remaining source coding that follows to be something simple to carry through with. The next step is the software integration, and the system testing, which will allow the whole system to be delivered to the client. The final phase is the maintenance of the complete system [1].

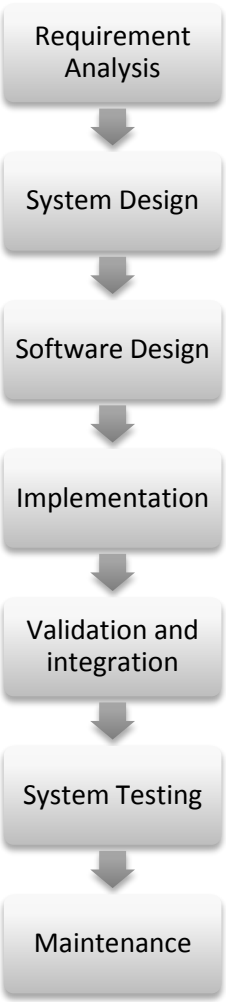


Figure 1: Waterfall model

A more detailed analysis of all the different phases before will be presented now:

- **Requirement analysis**: all the potential requirements of the system to be implemented are gathered during this phase. Those requirements consisted of the system capabilities and restrictions, and derive from the end user. Capabilities are extracted from the end user by consultation and then are analysed for their validity and potential integration. In the end, a requirement definition document is created, which will be the starting point of the next waterfall phase.
- **System and Software Design**: Before the start of the implementation of source code, it is highly important to understand what is going to be created and how it is going to look like. The requirement document that derived from the previous phase is studied during this stage, while the system design is forming. System design helps to clear out the hardware and the requirements of the system, while it provides input to the system's general architecture definition. The final design is the input of the next phase.
- **Unit Validation and Integration**: as defined earlier in this document, the system is initially divided in unit that are developed and tested for their functionalities. Those units are integrated to a unified system during this stage. Furthermore, validation

and operational tests take place in order to check the system aligns with the defined requirements. After the successful testing, the system is delivered to the client.

- **Operation and Maintenance**: This stage of the waterfall model practically never ends (or it is very long in terms of duration). Generally, some problems arise after starting to use the system. So, those issues are treated and solved after the system delivery. Furthermore, the majority of the problem do not appear at the same time, or immediately, but turn up at random times. This process is called maintenance. [2]

Disadvantages of the Model

From the moment the model was introduced, the waterfall model was highly criticised and sometimes not without a proper reason. In the section below, the disadvantages of the waterfall model will be underlined:

- As described in the section before, it is very important to gather all the possible requirements during the Requirement Analysis phase, in order to design the system properly. However, in real world environment, the requirement acquisition does not happen all in once. On the contrary, the client wants to add more even after the end of the aforementioned phase. This affects not only the implementation phase, but also the success rate of the system in a negative way.
- The more the requirement list gets bigger, the more of them rest unfulfilled, due to the fact the system implementation is already started. This leads to systems that are not completely user friendly and sufficient to the end user. The requirements that were left behind usually are covered with a newer version of the system, which raises the total cost of development.
- The whole project is not divided to phases in a flexible manner.
- The problems that occur during a phase are not completely tackled during that phase. In fact, many problems emerge after the ends of the phase which are related to. This leads to a badly structured system since the problems of one phase are not solved when the phase is still running.

2.2 Message Queuing Telemetry Transport (MQTT)

The Message Queuing Telemetry Transport protocol was initially developed by IBM in the late 1990s. It is a publish/subscribe messaging protocol that is highly used in the field of Internet of Things due to its low-bandwidth nature and high-latency environment tolerance. [3]

The systems that utilise MQTT are communicating via Topics. As seen in other protocols, topics ensure that a message will be delivered to the correct clients and is generally treated as a file path from the MQTT.

MQTT Advantages

- MQTT is open source, which makes its integration easier
- MQTT is power and bandwidth efficient by design, which leads to cost minimising
- It is highly flexible and scalable
- Provides QoS to its messages

Quality of Service

As mentioned before, the MQTT provides 3 levels of Quality of Service (QoS) that guarantee the delivery of a message. Messages may be sent to any QoS level, while users try to subscribe to a specific topic by specifying a QoS level. Practically, clients chose the maximum number of QoS that they will receive. [4] In the section below, all the MQTT Quality of Service levels are described:

- **0:** Message will be delivered exactly once, without any confirmation

Figure 2: MQTT Quality of Service 0. [5]

- **1:** Message will be delivered at least once, with confirmation required

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

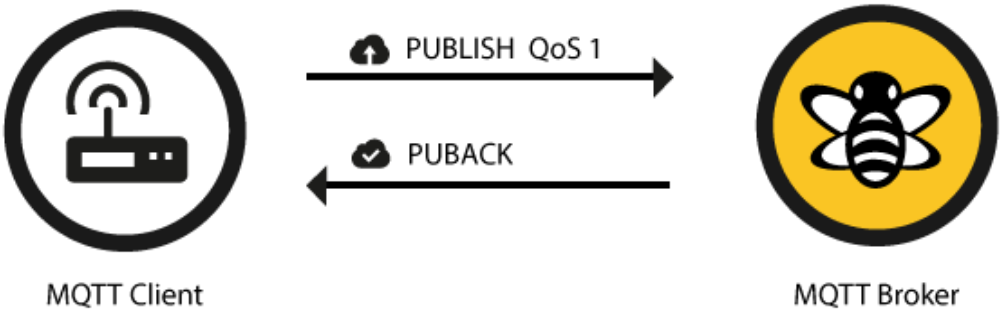


Figure 3: MQTT Quality of Service 1. [5]

- **2:** Message will be delivered exactly once, using handshake confirmation

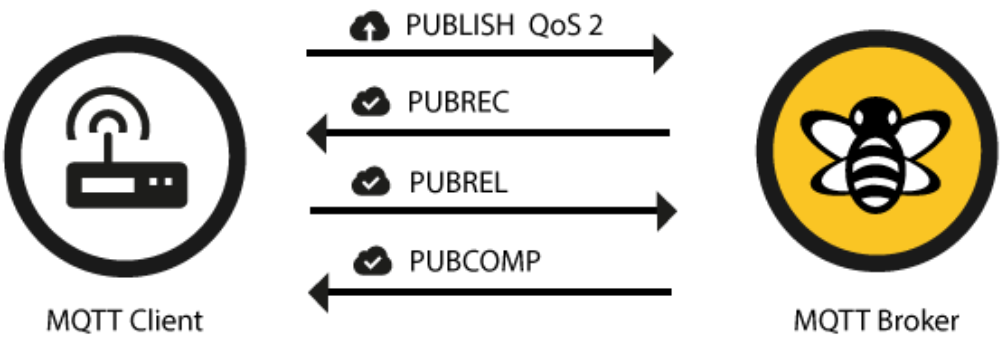


Figure 4: MQTT Quality of Service 2. [5]

The more we raise the quality of service, the more latency is introduced to the system accompanied with higher bandwidth requirements.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

3. Project concept and System Requirements

On this chapter we present the approaches utilised for our system, as well as the requirements that were extracted during the design phase of our Smart Meeting Applications suite.

3.1 Overview of Approach

Smart Meeting applications suite combines various techniques used by actual companies that develop Android applications for commercial purposes. One of our key factors to success is the strict utilisation of open source libraries. These libraries do not require any extra fee to be paid, avoiding a higher product cost.

This study demonstrates a way to, first of all, create a meeting, invite participants, and provide them information about the meeting venue and location. As participants arrive to the meeting premises, they can authenticate themselves by QR code scanning at general purpose Smart Panels and get directions to the specific meeting room without having to rely on other indoor navigation systems. Furthermore, additional functionalities, provided during the meeting, are presented. The end user can record the meeting from the best available portable device according to the specified preferences. Also, a revolutionary interactive way of pausing the meeting is introduced, using a real-time poll to question the attendants, without interrupting the meeting.

The major cost of operation is carried by the meeting host organisation. The Smart Panels may extend the overall expenses, but can be a great ROI considering the potential advertisement use. On the other hand, there is no additional cost for the meeting participants, as they will need to install a free open source application designed for their mobile smart device.

3.2 User Requirements

In this section, details about the interaction with the system will be presented. The various use case scenarios will be described providing additional information about the back-end processes when needed.

3.2.1 Create Meeting Use Case

The end user (meeting organiser) should be capable of creating a meeting specifying the date, time and location and indicate the invited visitors. The system should be capable of sending the invitation attaching a QR-Code, providing information related to the participant and the meeting itself.

The organising application should be supported from multiple devices and screen size to suit a wide range of potential users. Additional attention must be paid on the feedback provided by the system across the various stages of the meeting creation.

The meeting host will fill the details of a meeting including information such as:

- Title
- Description
- Visitors' details

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

- Venue

The system will validate the input data and if everything is matched to the database, it will create the new meeting. After that, using an external library it will generate dynamic QR-Codes containing information about each invited individual, as well as details about the actual meeting. When the QR-Codes are generated, the system should create personalised emails for all the attendants containing detailed information about the meeting and the corresponding QR-Code.

3.2.2 Location Guidance Use Case

The participants are able to find their way to a meeting venue by using a pre-installed system and the QR-Code received at the meeting organisation phase. The system reads the dynamically-generated QR-Codes and is able to show through a panel the direction that should be followed by the participant. In case the scanned QR-Code is not in the correct format, the user should be informed about that. Also, there will be additional feedback from the system to the end user, in case the scanned "meeting" is not available in the database.

The panel should also provide general information to the visitors using the World Wide Web, without compromising the system's security. A browser running in a sandbox environment will enable the users to surf the web, disabling all the potential harmful use.

Furthermore, an options menu should be accessible only from the system's administrators in order to configure the device on the fly.

3.2.3 Meeting Recording Use Case

The user is able to record the on-going meeting through the attendants' mobile devices. By sending a request via the organiser's mobile application, the system starts to gather information about the present devices' status. Later on, it processes all the data and selects the most appropriate device to do the recording. According to the organiser's preferences, the suitable matches may vary.

All the steps needed for the Smart Meeting Recording are run in the background without disturbing any of the meetings' participants during the meeting session. However, the recording device should inform the owner about the ongoing process and should be automatically cleared when finished.

Finally, the properties of the meeting should be dynamically changed by the meeting organiser, using his mobile application.

3.2.4 Meeting Break Use Case

The user should be capable of asking the meeting's attendants for their desire to have a break without interrupting the flow of the on-going meeting. The request will be sent from a mobile application specially designed for the meeting organiser, following the Google's official design guidelines that aim to a user friendly experience.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

After a given period of time, the system will analyse all the gathered data and send the feedback result back to the user who started the “Smart Meeting Break” procedure. Sending the request can be triggered at will, but also assisted by a background running functionality which can identify the potential need of a break.

Also, another way, more automated, should be provided. When the organiser decides he wants to monitor the meeting, all the devices located at the meeting venue, will start gather measurements and upload the data to a remote server. There, after a dynamically given amount of time, the probability of needing a break will be measured. Also, the estimated amount of remaining time will be calculated and sent back to the meeting organiser.

3.2.5 Meeting Wrap-Up Use Case

The system will gather all the data generated and provided by the participants and store them in a remote yet accessible folder. Thus, all the data will be available not only to the meeting attendants but to the rest of the invited persons who could not be physically present at the venue. Furthermore, the data could be accessed for any further offline analysis that may be viewed as useful.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

4. System Design

4.1 Introduction

This section describes the design of all the systems presented in this study. It analyses in depth how the various components interact in order to achieve the goals presented as system requirements. Furthermore, the high level system design and the components interactions are depicted on the figures below.

4.2 High Level System Design

This section of the design document shows a high level overview of the system and how the various system components interact with one another.

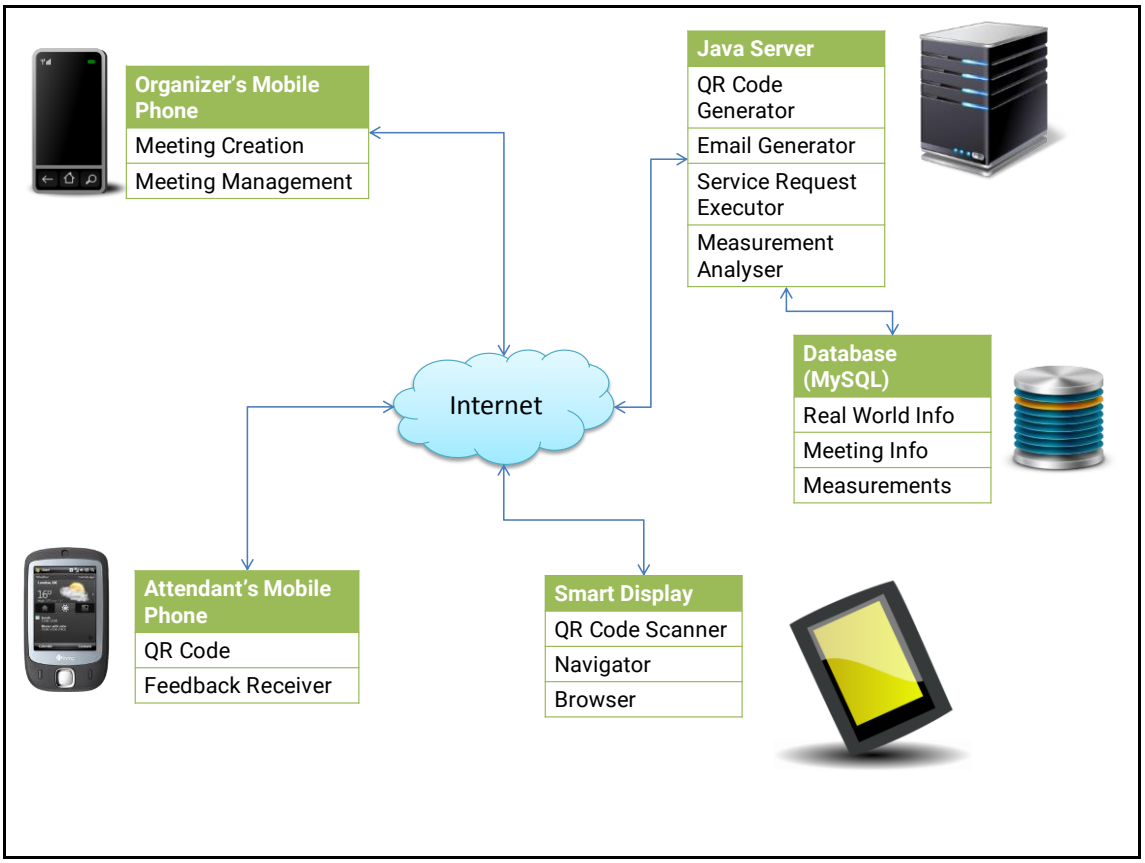


Figure 5: High Level System Design.

As shown in Figure 5 the meeting is created and initiated by a mobile application. This is an application which can be installed in any Android device and be used by the meeting organiser according to the meeting creation system requirements.

4.3 Component Design

This section contains the in-depth analysis of the system components and their relations that result to meeting system aims. Thus, the Java Server (CVO Management Unit), the Smart Displays (Smart Meeting Guidance Application), the visitor Android application (Smart Meeting), the organiser application (Smart Organiser) and the database are presented thoroughly.

4.3.1 Java Server (CVO Engine) Design

The Java server or CVO Engine is the most important component of our system. Everything runs through it, as for the sake of scalability and reusability of the components there is no direct communication between the applications to be presented in the next sections of this study.

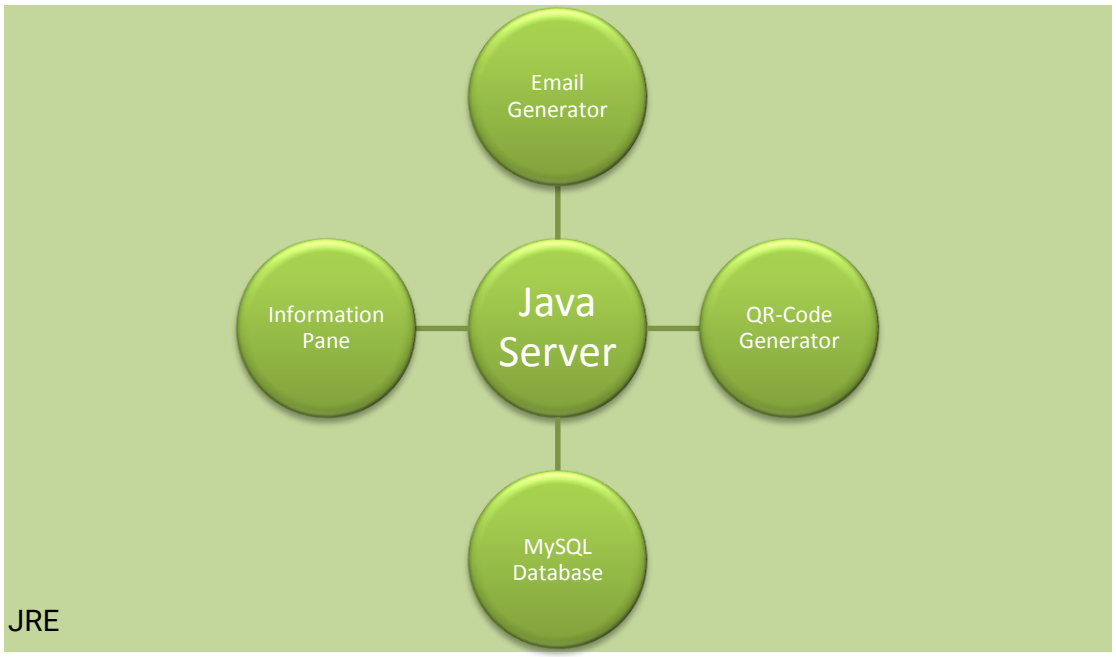


Figure 6: CVO Engine Design

As shown in Figure 6, the CVO Engine is responsible to provide information to the administrator for everything that is going on in the background. Furthermore, it is connected with an open source MySQL database where we store various important information. A detailed review of the database is available later in this section of our study. Additionally, our CVO engine can generate emails and attach QR-Codes that are generated on the go during the SMC process.

In order to achieve the functionality just mentioned, the following open source libraries are used:

- activation.jar
- commons-email.jar
- commons-io.jar
- commons-lang.jar
- javax.mail.jar
- mysql-connector.jar
- org.eclipse.paho.client.mqttv3.jar
- zxing-core.jar
- zxing-javase.jar
- zxing.jar

4.3.2 Database Design

In this section a detailed view of the databases used by the system is presented. Our system interacts with two (2) different databases, as shown in Figure 7 and Figure 8 where we can clearly see the tables created and also the relations between different fields.

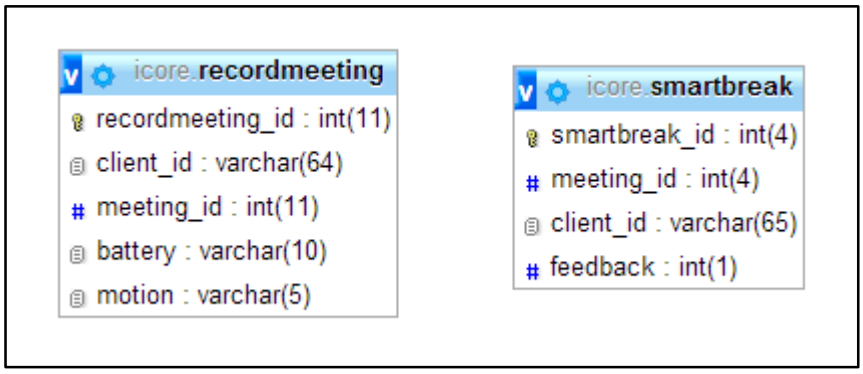


Figure 7: Measurements Database.

The first database depicted in Figure 7 is called “icore” and contains two (2) tables that are required for the Smart Meeting Recording (SMR) and Smart Meeting Break (SMB) use case scenarios.

Record meeting

- **recordmeeting_id:** auto incremental unique id. It is used as index for this table.
- **meeting_id:** The unique meeting id for which the Smart Meeting function is called. Use to distinguish the database recordings.
- **client_id:** This field contains the unique id that each Android device has. It is used to avoid the duplicates.
- **battery:** the percentage of battery that a device has for any given time.
- **motion:** this field stores the motion status of a device for the last thirty (30) seconds.

Smart Break

- **smartbreak_id:** auto incremental unique id. It is used as index for this table.
- **meeting_id:** The unique meeting id for which the Smart Meeting function is called. Use to distinguish the database recordings.
- **client_id:** This field contains the unique id that each Android device has. It is used to avoid the duplicates.
- **feedback:** contains the users’ response to the SMB question.

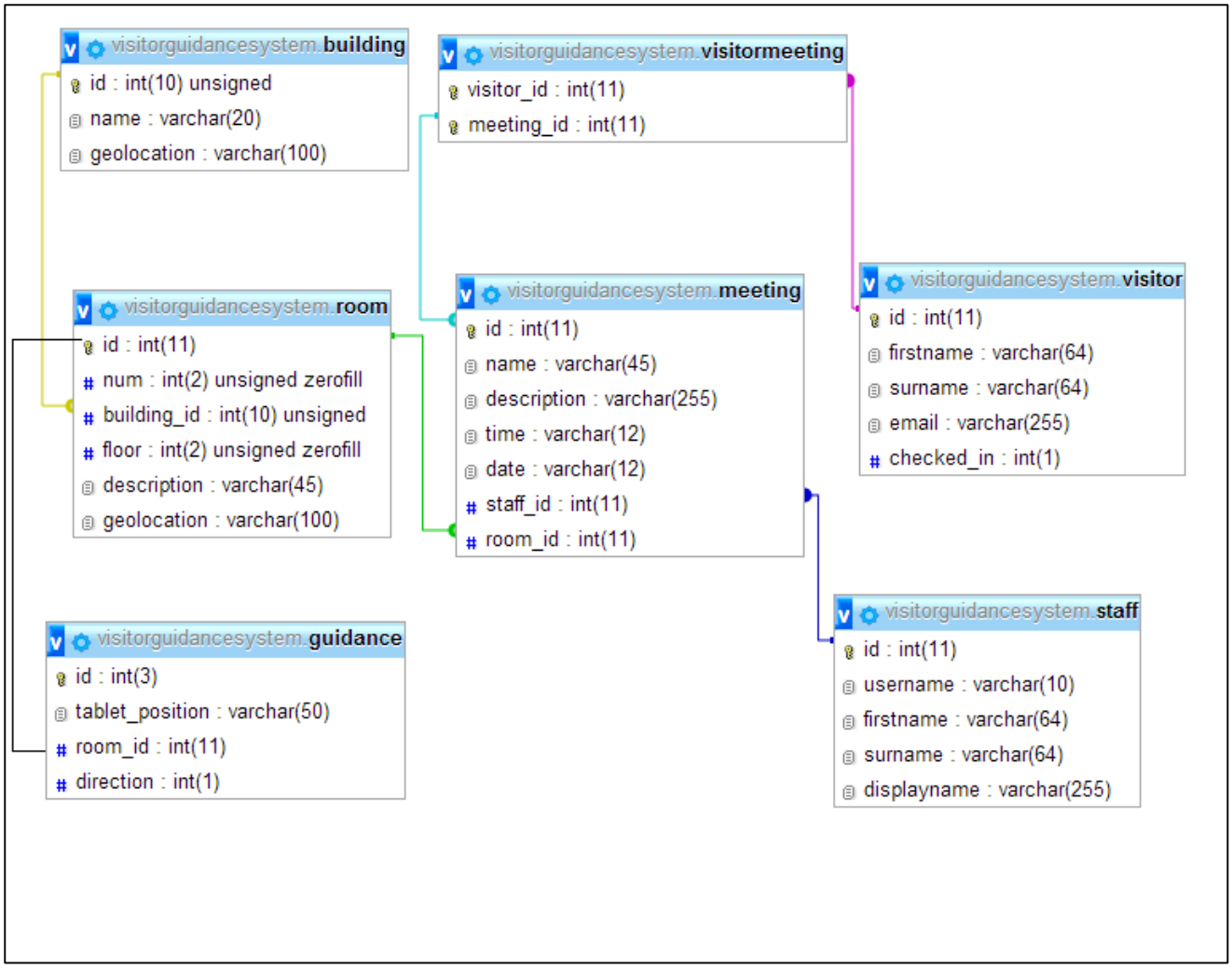


Figure 8: Meeting and Guidance Database.

The second database called "visitorguidancesystem" is more complex than the one described earlier in this section. This one contains seven (7) different tables that are described in depth below:

Building

- **id**: auto incremental unique id. It is used as the index for this table.
- **name**: the name of each building (i.e. BA, Library)
- **geolocation**: the location of the building presented in the form of a string.

Room

- **id**: auto incremental unique id. It is used as the index for this table.
- **building_id**: foreign key of the building table which corresponds to the building where the room is located to.
- **num:** the number of the room.
- **floor:** the floor of the room (i.e. 00, 01, 02).
- **description:** additional info about the room (i.e Test Room, Reading Room, MSc Lab).

- **geolocation**: the location of the room presented in the form of a string.

Guidance

- **id**: auto incremental unique id. It is used as the index for this table.
- **tablet_position**: the position of each tablet (i.e. entrance, Library)
- **room_id**: this field contains a foreign key from the "room" table.
- **direction**: this field contains a number in the range 0-8. Depending on the tablet position and the room id stored it provides the direction to the next hop. It serves as a routing table of our system. Zero (0) serves as to inform the user of his successful arrival to the meeting venue. Numbers 1-8 are used to guide the user as shown in the figure below:

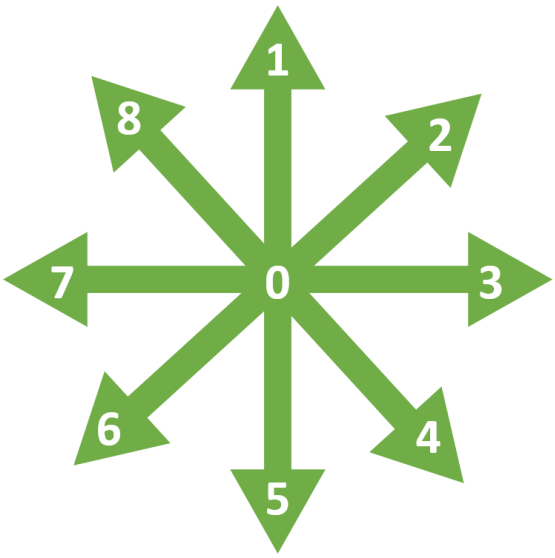


Figure 9: Directions mapping

Staff

- **id**: auto incremental unique id. It is used as the index for this table.
- **username**: the username selected from each staff.
- **firstname**: staff's username
- **surname:** staff's surname**.**
- **displayname:** the name to be displayed inside the application.

Visitor

- **id**: auto incremental unique id. It is used as the index for this table.
- **firstname**: the visitor's first name.
- **surname: the visitor's last name.**
- **email**: the visitor's email address, where the system will send the invitation with all the information needed.
- **checked_in**: the status of the visitor. Initially is set to zero (0), but when the users arrives to the meeting venue it is set to one (1).

Meeting

- **id**: auto incremental unique id. It is used as the index for this table.
- **name**: the title set by the meeting organiser (i.e. Progress Meeting).
- **description**: the detailed description of the meeting, provided by the meeting organiser.
- **date**: the date of the meeting.
- **time**: the scheduled time of the meeting.
- **staff_id**: foreign key that corresponds to the staff member who created the meeting.
- **room_id**: foreign key that corresponds to the room where the meeting will take place.

Visitor-Meeting

- This table contains the foreign keys from "visitor" and "meeting" table, in order to create the 1-1 relation needed for every visitor and meeting.

4.3.3 Smart Meeting Displays

The Smart Meeting Displays are tablets that run the Android OS and are mounted on the walls of a building. They are the cornerstone of the SMG use case scenario as will be described later in this section. Below we can see the components that consist the Smart Meeting Displays.

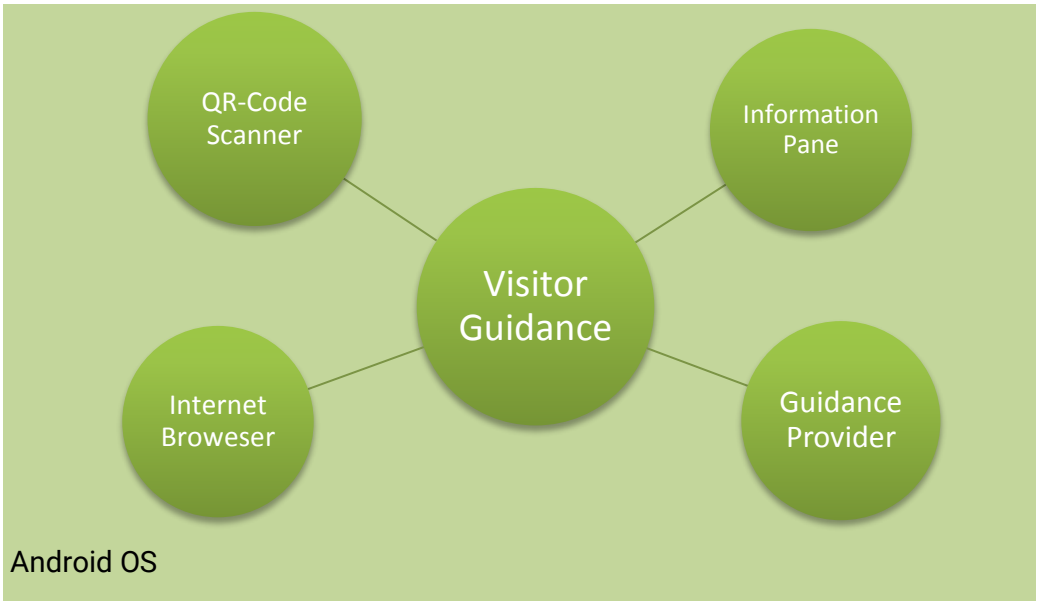


Figure 10: Smart Meeting Display Design

The Smart Meeting Guidance application that runs on the Smart Meeting Displays is a launcher application that overrides the default one. The front-end provides two (2) main functions to the visitor:

- Visitor Check-In
- Internet Browsing

When a user checks in, the display becomes a viewfinder of the hardware camera mounted on the front of the device. The user can scan his invitation QR-Code and check in to the building. If he wishes he can receive additional information about the direction he should follow in order to get himself to the meeting venue. On the other hand, when the user wishes to browse the internet, the panel transforms to a custom implemented web browser.

In the back-end of this application, the QR-Code scanning can be achieved with any appropriate android application, but the handling of the data captured is made by using the open source library zxing[1].

4.3.4 Smart Meeting Organiser

The Smart Meeting Organiser is also an application that runs on Android OS devices. It is optimised for smartphones and tablets to offer great user experience via state of the art UX/UI design.

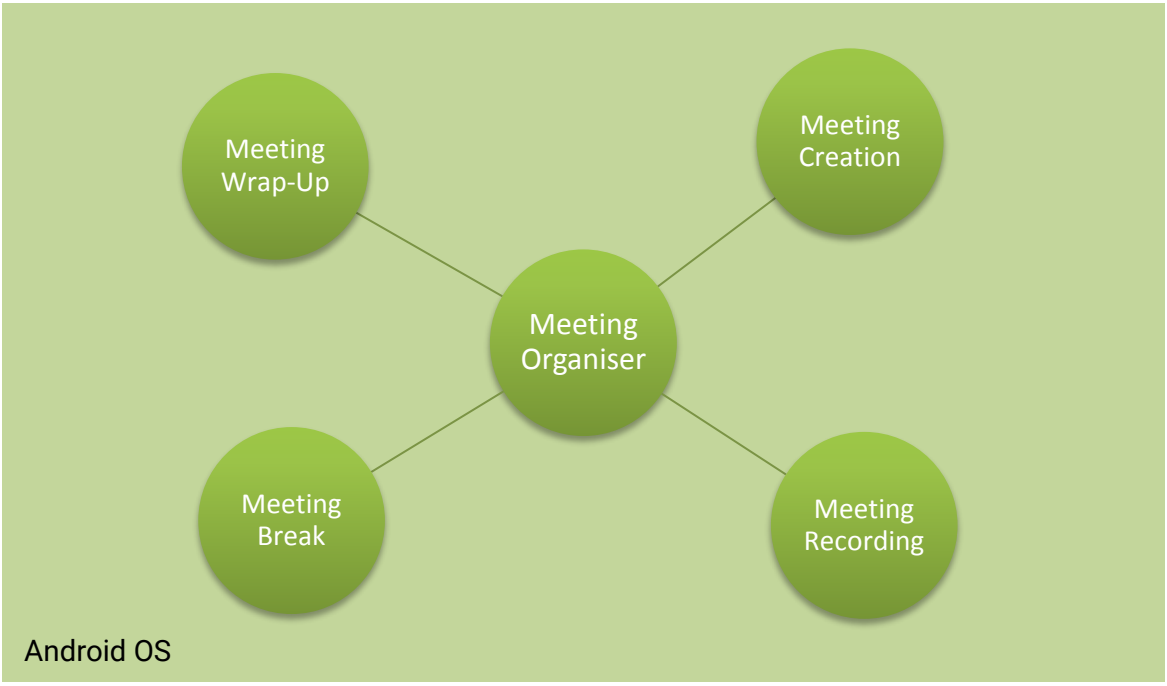


Figure 11: Smart Meeting Organiser Design.

As shown in Figure 11, the majority of the functions provided in the Smart Meeting Application suite are part of the Smart Meeting Organiser android application. A user can request through his mobile device the instantiation of various components that handle the:

- **Smart Meeting Creation**: the user can create a new meeting by completing a form with all the necessary details of a Smart Meeting, as well as providing the personal info of the visitors to be invited. When he selects to save/create a meeting, a connection with the Java server is made through MQTT.

[1]https://github.com/zxing/zxing/

- **Smart Meeting Break**: the user can select between two (2) modes to decide whether a break is needed, or not. The manual mode creates a poll to be answered from all the attendants without the need to disrupt the meeting flow. The automatic mode, gathers measurements from the visitors' devices in the background and through a complex algorithm is produces the possibility of having a break is necessary.
- **Smart Meeting Recording:** the meeting organiser can choose to start or stop the background recording of the ongoing meeting. All the devices present in the meeting venue are sending data that are being analysed by the CVO Engine in order to find the most appropriate device in terms of power and performance to handle the recording at any given time.
- **Smart Meeting Wrap-Up:** finally, the user with this mode can gather all the meeting-generated data and upload them to a cloud store or an ftp server.

It is important to mention that all the functionalities mentioned above are completed via the CVO Engine using the MQTT protocol.

4.3.5 Smart Meeting Visitor

The Smart Meeting application designated for use by meeting visitors is the last piece of our applications suite. On the front end it is an information guide for the attendants, while in the back-end is a feedback and measurement provided aiming the support of our Smart Meeting scenarios.

As depicted in Figure 12, the majority of the components are running in the background without endangering the user experience or interrupting his work. Same as Smart Meeting Guidance, Smart Meeting Organiser, and CVO Engine, the use of MQTT protocol is essential for all the communications with the external components of our system.

- **Organisation Info Display:** This component allows the user to receive information about the host organisation by fetching the official website of the university.
- **Map Display:** this component is a Google maps implementation inside the application. It has pinpointed the exact location of the meeting venue enabling the visitor to guide himself at the meeting building.

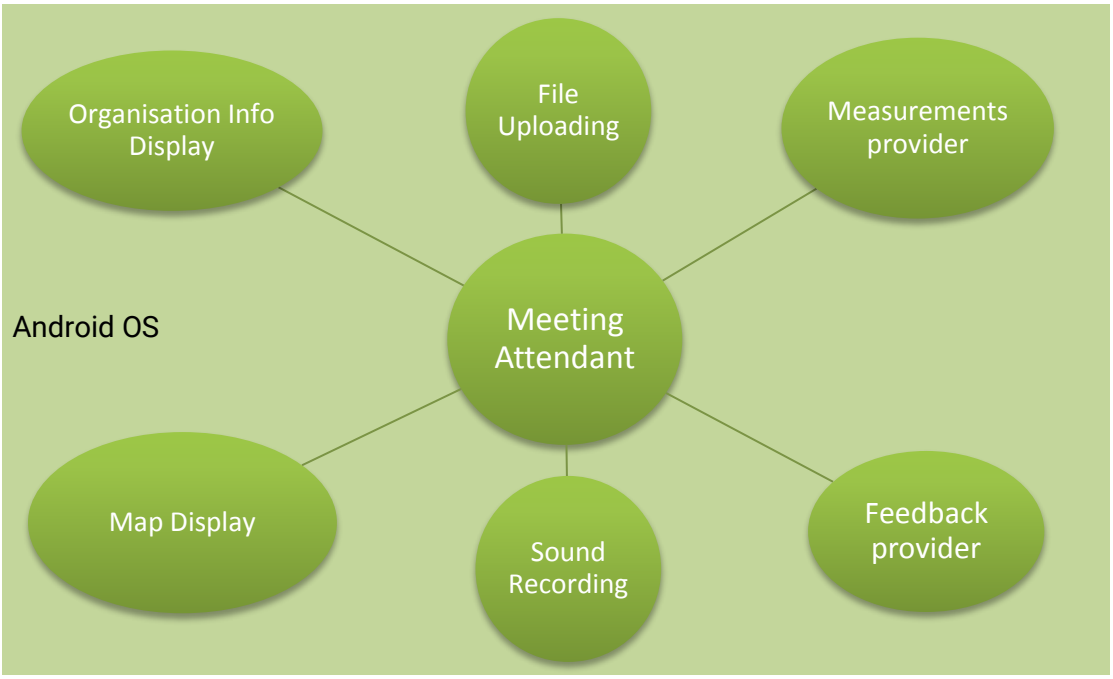


Figure 12: Smart Meeting Attendant Design.

- **Measurements provider:** when needed, this component accesses the hardware acceleration sensor inside the device as well as the microphone to get the level of the ambient sounds. Those measurements are needed for the SMB and SMR features.
- **Sound Recording:** this component accesses the microphone of a device and records the meeting for a specific time window. After the end of the recording, the sound fragment is stored locally on the device's memory card.
- **Feedback provider: i**n order to create the poll presented in the Smart Meeting Break scenario, we need to get some feedback from the users. This component creates a notification that enables the user to express his opinion on the subject. The response will be sent to the CVO Engine and then stored temporary to the MySQL database for later processing.
- **File uploading:** this component is used at the Smart Meeting Wrap-Up scenario. It is responsible to gather all the meeting-generated data and send them to the CVO Engine for further handling. In order to transmit the files, the system creates a serialisable object of a class that defines any file and then it is sent as the payload of an MQTT message. Finally, it deletes all the locally-stored data from the device.

4.4 Sequence Diagrams

This section of system design depicts the components interactions with each other based on sequence of events in the order at which they will take place. It displays the interactions between the Android applications, the MQTT broker and the Java server. User actions will be highlighted as side-notes when needed and the same rule applies to the database transactions.

4.4.1 Smart Meeting Creation Sequence Diagram

When the meeting organiser decides to create a new meeting a series of events that are depicted in the figure below takes place. We can see that the whole procedure is a combination of MQTT messages with database interactions and usage of external libraries that generate data.

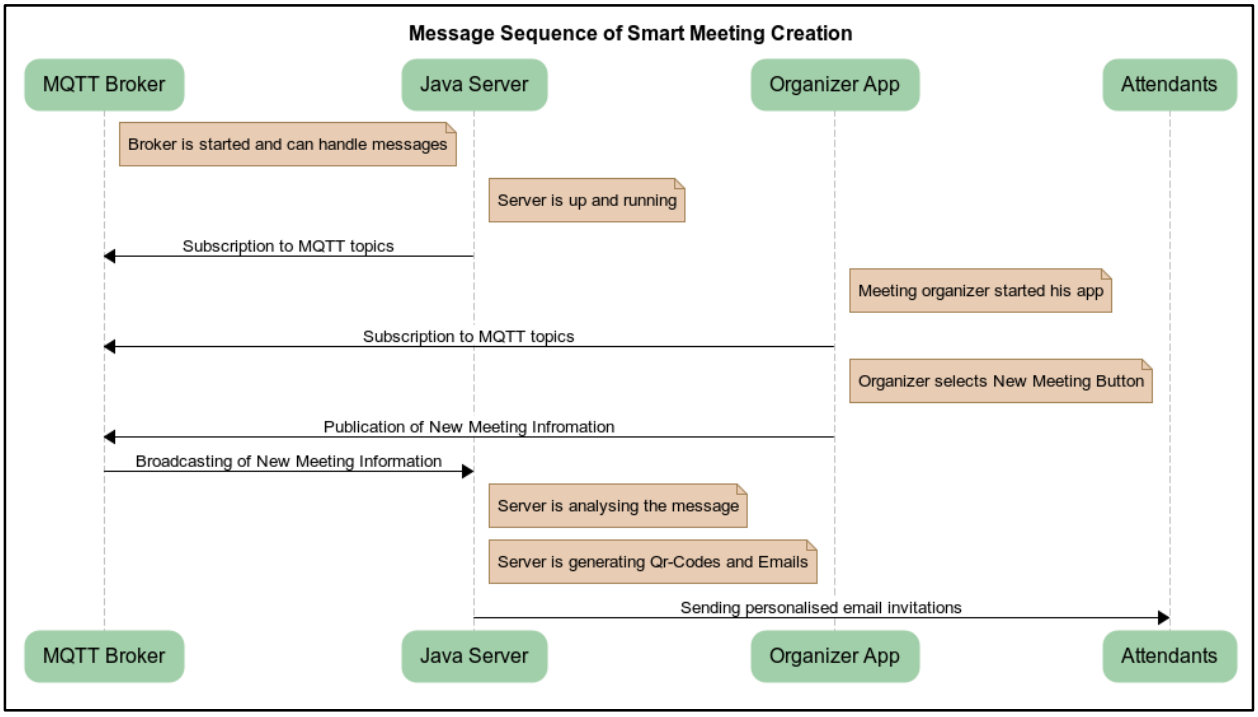


Figure 13: Smart Meeting Creation Message Sequence.

All the end user has to do is select the appropriate button inside the application and a fill the form. When he saves the meeting the details of the meeting are serialized and sent to the MQTT broker, which is responsible to publish them to the Java server. Then, after being de-serialized, the data are stored in the database and personalized QR codes and emails are generated and sent to all the invited attendants.

4.4.2 Smart Guidance Sequence Diagram

When a visitor arrives at the meeting building he can see the pre-installed Smart Meeting Guidance (SMG) system mounted on the wall. There, he scans the QR-Code attached to the received email and the system shows him the direction he should follow in order to get himself to the meeting venue.

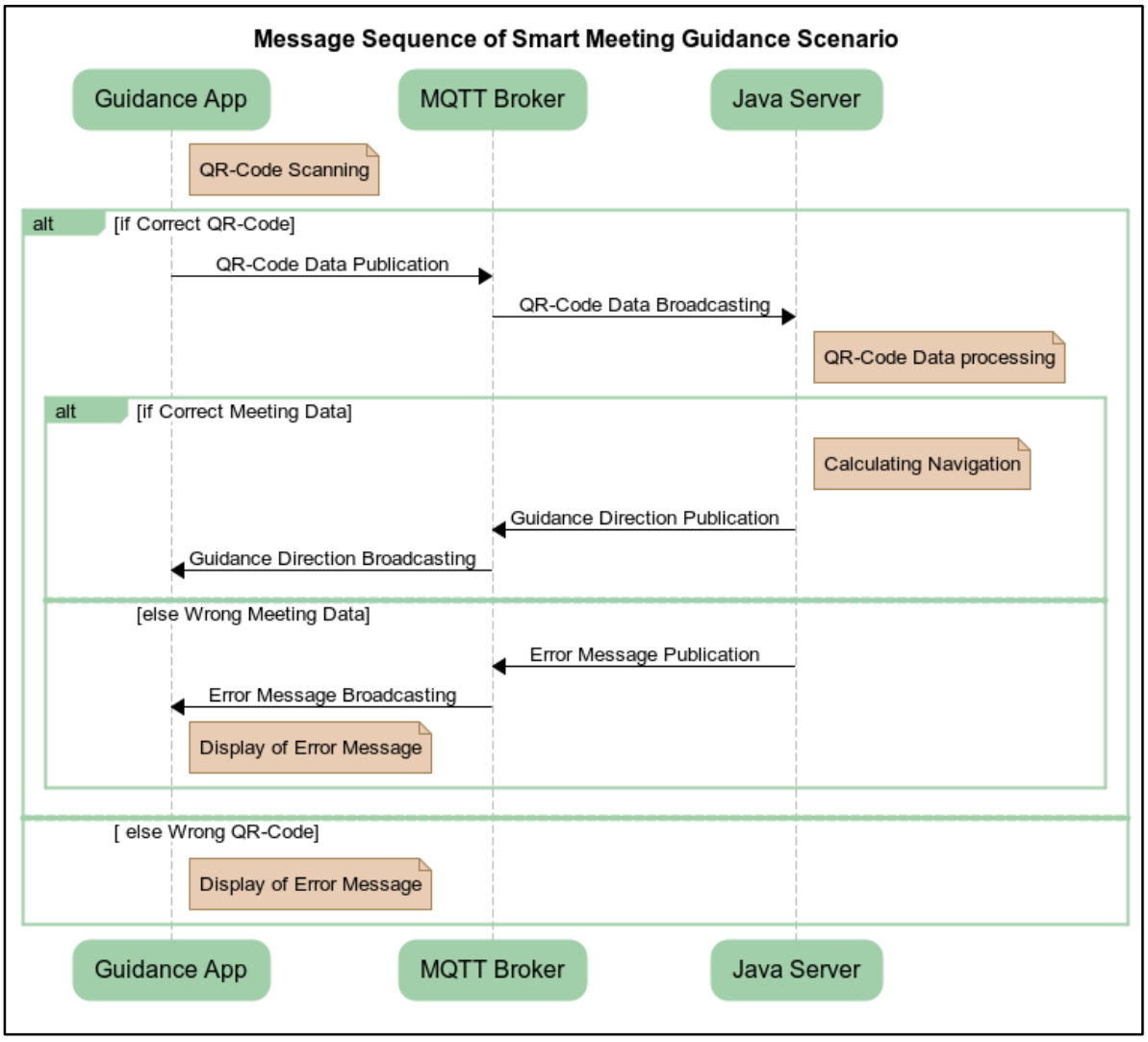


Figure 14: Smart Meeting Guidance Message Sequence.

On the back end of the system, the QR-Code is read and analysed by the mounted panel and an MQTT message is sent to the appropriate CVO through the MQTT broker. When this message is received by the CVO, it is processed and the all the meeting data are fetched from a database. Finally, the direction is being sent back to the SMG and the visitor is informed of where he should go.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

4.4.1 Smart Meeting Break Sequence Diagram

In this section we will talk about the two (2) different approaches we have concerning the Smart Meeting Break (SMB) scenario: the automated and the manual.

4.4.1.1 Automated

As depicted in Figure 15 the Automated Smart Meeting Break scenario is initialised by the meeting organiser. This is done via the appropriate button on his mobile device screen. The trigger is sent from the mobile device to the MQTT broker and then, it is broadcasted to the CVO Engine. After analysing the request, the CVO Engine is sending the trigger to all the available devices at the meeting venue. When the Smart Meeting Attendant application receives the request it uses the appropriate components to access the needed sensors of the device. Then, periodically sends data to the Esper Engine which is also up and running via MQTT messages. Those data for the sake of our demo is just noise level. The Esper Engine is not part of this study, but it is used in this scenario hence the reference. After a set time, the Esper engine calculates the outcome of the measurements and then publishes it to the MQTT broker. Finally, the MQTT broker is broadcasting the automated SMB result to the meeting organiser.

4.4.1.2 Manual

In this scenario, the meeting organiser wants to get direct feedback from the attendants and a sequence of events – that are depicted in the Figure 16 – take place. Similarly with the rest of the scenarios presented in this study, we take for granted that the CVO engine is up and running and the MQTT broker is set up to receive and broadcast messages. When the meeting organiser selects to use the Smart Meeting Break function, a trigger is sent to the broker and then broadcasted to the CVO Engine. Then, the java server analyses the request and informs all the available devices running the Smart Meeting Attendant application to start gathering data. From now on, until the meeting organiser explicitly requests it, all the devices inside the meeting venue will gather and transmit sensor data to the Esper Engine. All these measurements, are temporary stored and analysed after a given amount of time. The Esper Engine is able to predict the urgency of having a break expressed as a probability. Furthermore, another outcome of the back-end algorithm is the projection of the next break. All the information generated by the Esper engine is published to the MQTT broker end finally to the meeting organiser. Both the probability of having a break and the estimated remaining time are displayed in the applications' notification area.

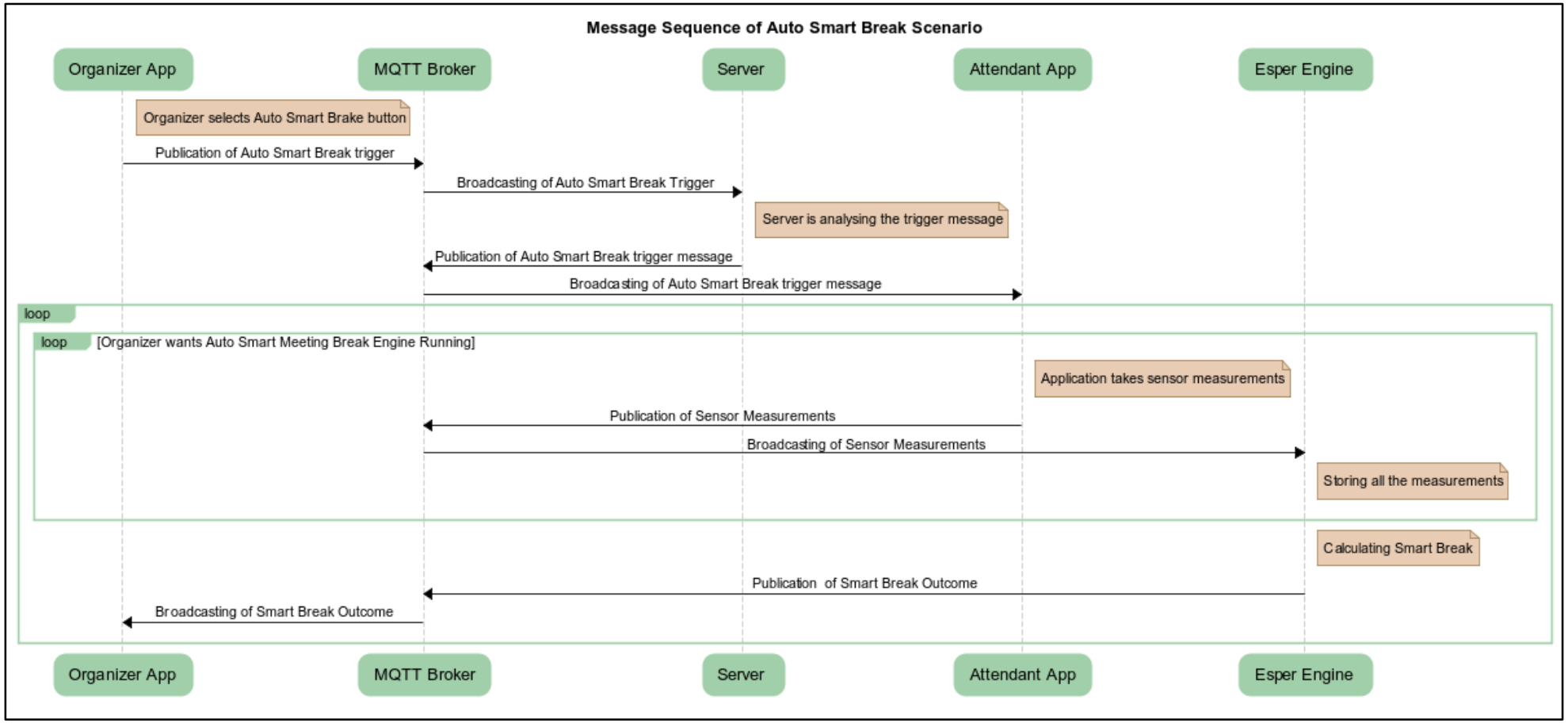


Figure 15: Auto Smart Meeting Break Message Sequence

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

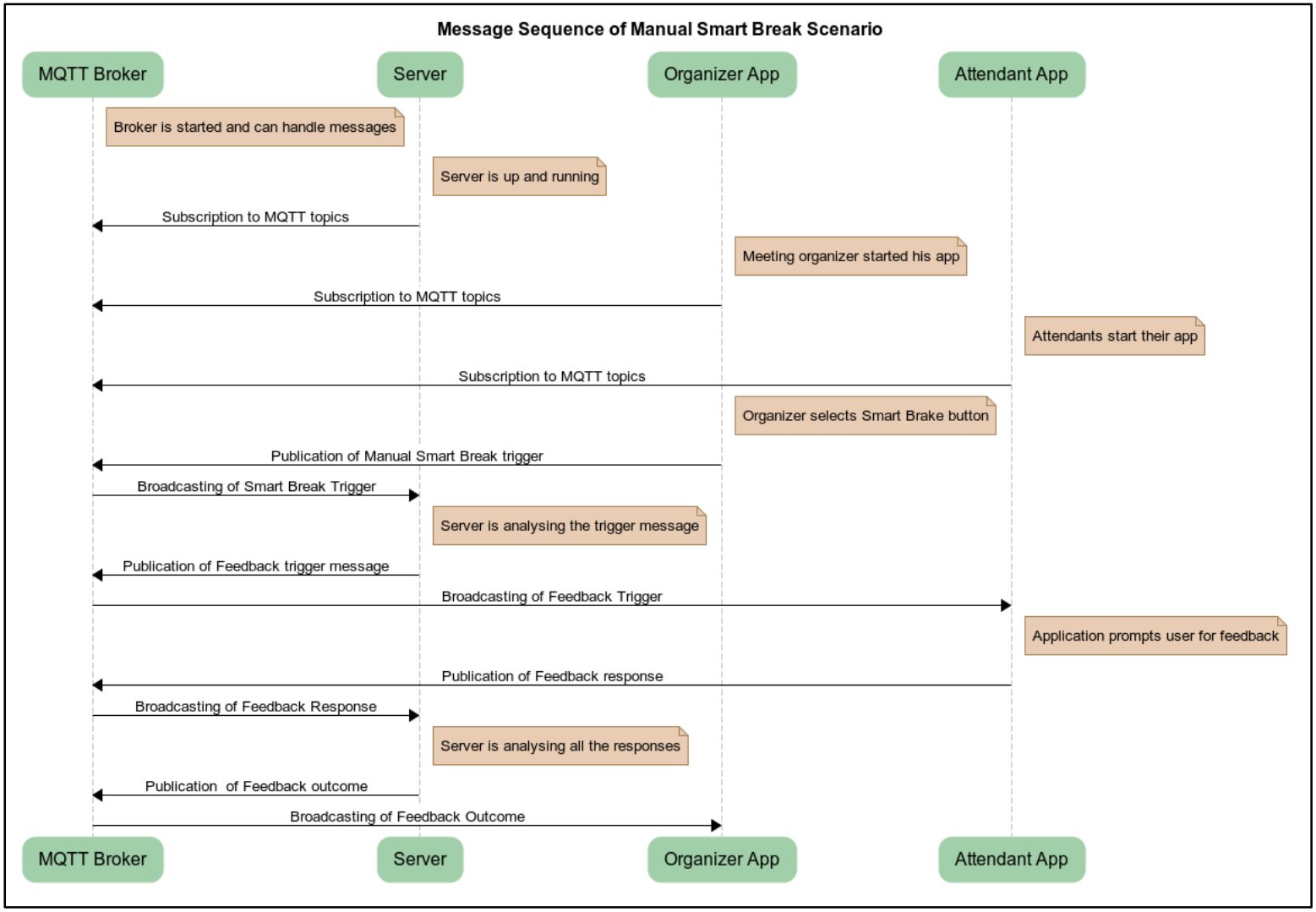


Figure 16: Manual Smart Meeting Break Message Sequence..

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

4.4.2 Smart Meeting Recording Message Diagram

During the Smart Meeting Recording scenario the tangible target is the recording of the on-going meeting from the best possible audio source. The steps required to succeed that are similar in some points to the previous scenarios.

First of all, we assume that all MQTT broker, Java Server, Organiser-Attendant app, are up and running and waiting for requests handling. When the meeting organiser selects the smart-recording button from his management screen, a trigger is sent via the MQTT Broker to the Java server. There, this message is analysed and the request of smart recording is forwarded to all the eligible devices inside the meeting venue. Then, until the organiser selects to stop the recording, all the devices are responsible to gather sensor data and publish them to the Java server. The data contain information about the remaining battery on each device, as well as, its motion status the last seconds. When the server receives data from an attendant's smart portable device, it stores the data for later processing. After a user-defined amount of time, all the measurements are processed by the Java server in order to find the best device to record the meeting according to the meeting-organiser preferences. Finally, the qualified portable device is informed to start the recording through the MQTT trigger message send by the Java Server.

4.4.3 Smart Meeting Wrap-Up Message Diagram

The Smart Meeting Wrap-Up is the last scenario presented by this study. As described before in this document, the purpose of this feature is to gather all meeting-generated data and upload them to an easily-accessible location.

When the meeting organiser selects the according button, the wrap-up trigger is send and analysed by the Java server. Later on, the Java server through the MQTT Broker broadcasts the wrap-up trigger to all the attendants' portable devices. Then, until all the files from each device are uploaded the following steps are taking place:

- Each smart portable device is running a search in order to find all the meeting-generated data.
- If found any, then, all the important information is serialised using the appropriate java class.
- Later on, the serialised object is published via MQTT as the payload of a message.
- When the Java server receives the serialised object, it stores it locally and then processes it.
- After the de-serialisation, the received file is stored to a widely-accessible location.

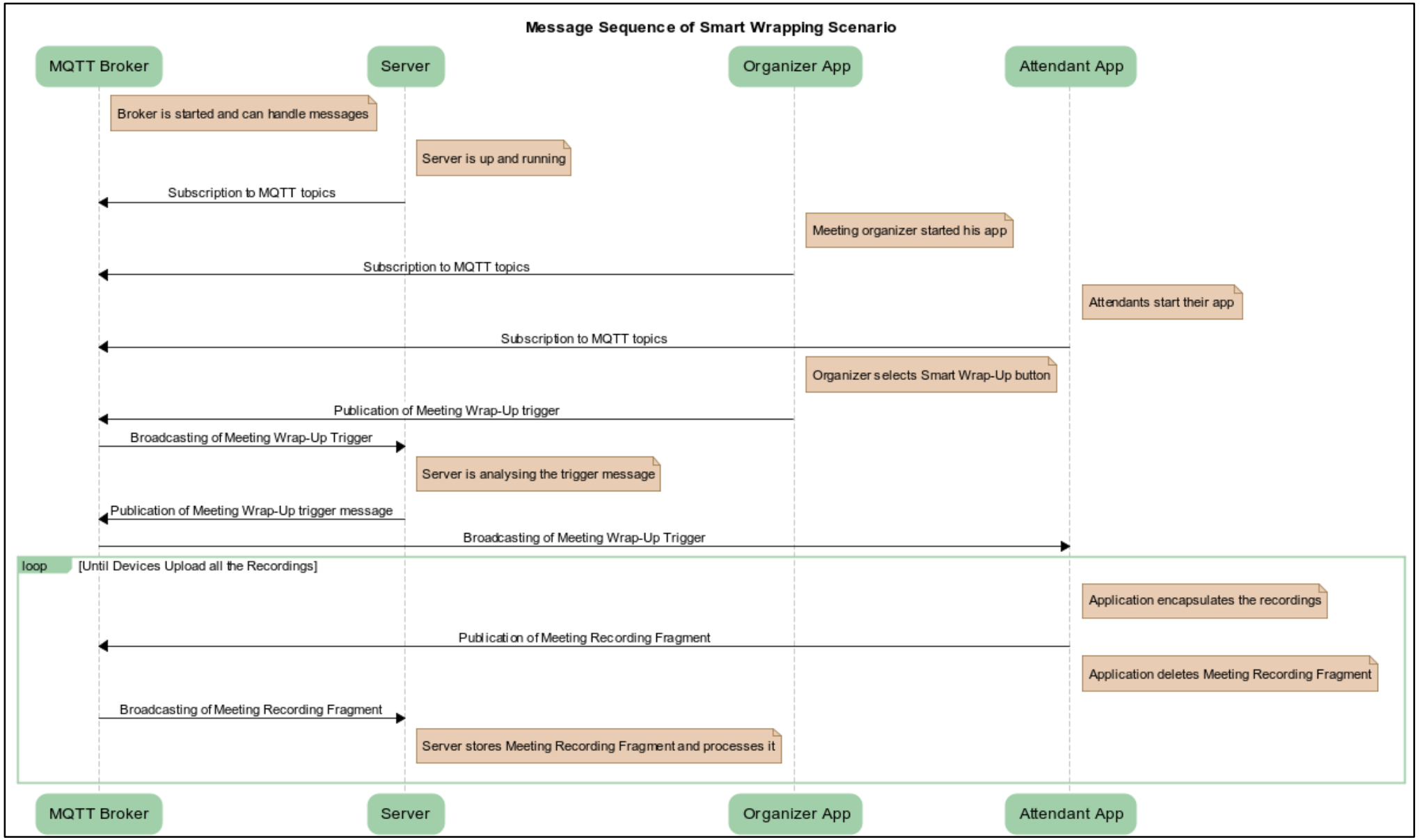


Figure 17: Smart Meeting Wrap-Up Message Sequence

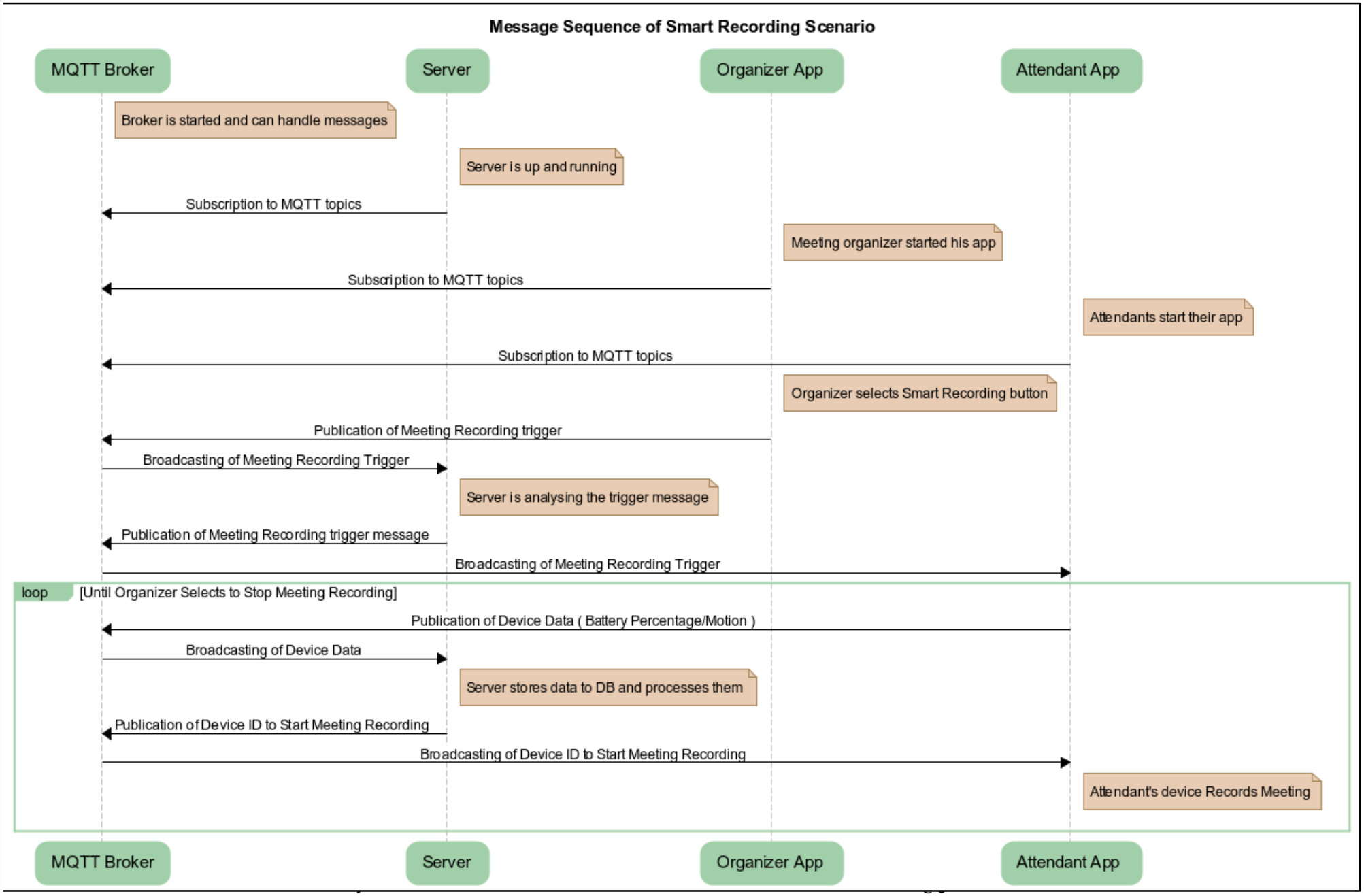


Figure 18: Smart Meeting Recording Message Sequence

5. System Implementation Techniques

5.1 Introduction

This chapter of the report details how the design of the system is implemented, as well as, the various techniques used in the implementation of the visitor guidance system, following the system requirements and design guidelines.

Firstly, we will talk about the general UI design and later on, we will display the various functionalities that serve the system requirements. For the sake of the presentation let us assume Vassilis. Vassilis is a member of the host organisation and the meeting organiser. Throughout the next sections, the study will be structured from Vassilis' point of view.

5.2 Introduction to Implemented Components

Before presenting the detailed analysis of the techniques, a quick introduction to the user interface will take place. This will enable the reader to familiarise with the context showcased in the rest of this chapter.

5.2.1 Organiser Mobile

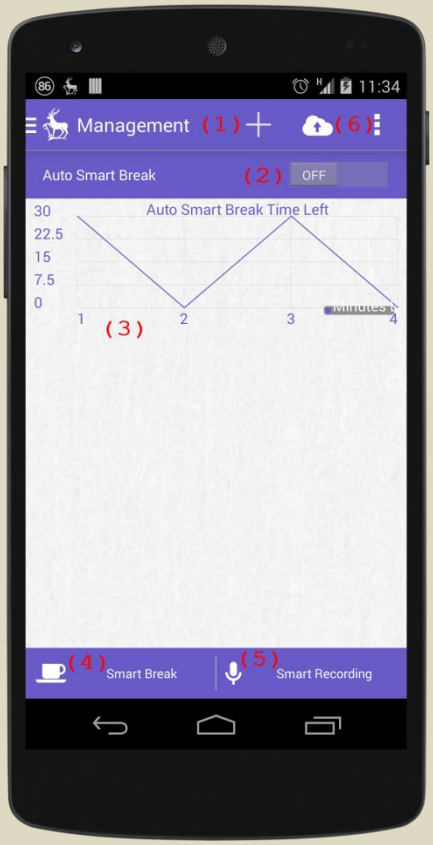
The organiser mobile application was designed including two (2) versions, suitable for either smart phones or tablets, enhancing the offered user experience for a wide variety of devices.

As depicted in the Figure 19, the meeting can be organised through this Android activity. All the available features are displayed in discreet, yet comprehensive, way.

On the top part of the screen, the end user can either create a new meeting, wrap-up an existing one, or display some additional features, such as the settings menu. On the bottom of the layout, he can initiate the manual smart break feature, or the smart recording functionality. Furthermore, right below the action bar (top banner of the screen), there is a switch which informs and provides the functionality of the automated smart break scenario. The outcome of this scenario is projected on the graph listed as No.3 in the Figure 19: Organiser application UI. More details about the real-life scenarios will be presented later on this study.

Inside the application, there is an activity [2] that handles the login of the user to the system. It is also responsible to store the credentials if needed. However, this is out of the scope of this study, but a mock-implementation took place for the sake of a more complete demo application.

[2]http://developer.android.com/guide/components/activities.html



Organiser Application UI includes:

(1) Meeting Creation
(2) Meeting Auto Break
(3) Meeting Auto Break Real Time Graph
(4) Meeting Manual Break
(5) Meeting Recording
(6) Meeting Wrap-Up

Figure 19: Organiser application UI.

5.2.2 Introduction to Java Server Engine

This section includes the first presentation of the java server implemented for the smart meeting scenario. Besides the provided functions, it also serves as a system message output. The end user can see any given time what is running in the background in the appropriate text view.

This server can be started by the button placed in the middle of the screen which becomes disabled once selected. Once up and running, the proper notification messages appear on the system output and the server can now handle smart office scenarios. Furthermore, a “clear log” button is activated and can clear the output screen at any given time.

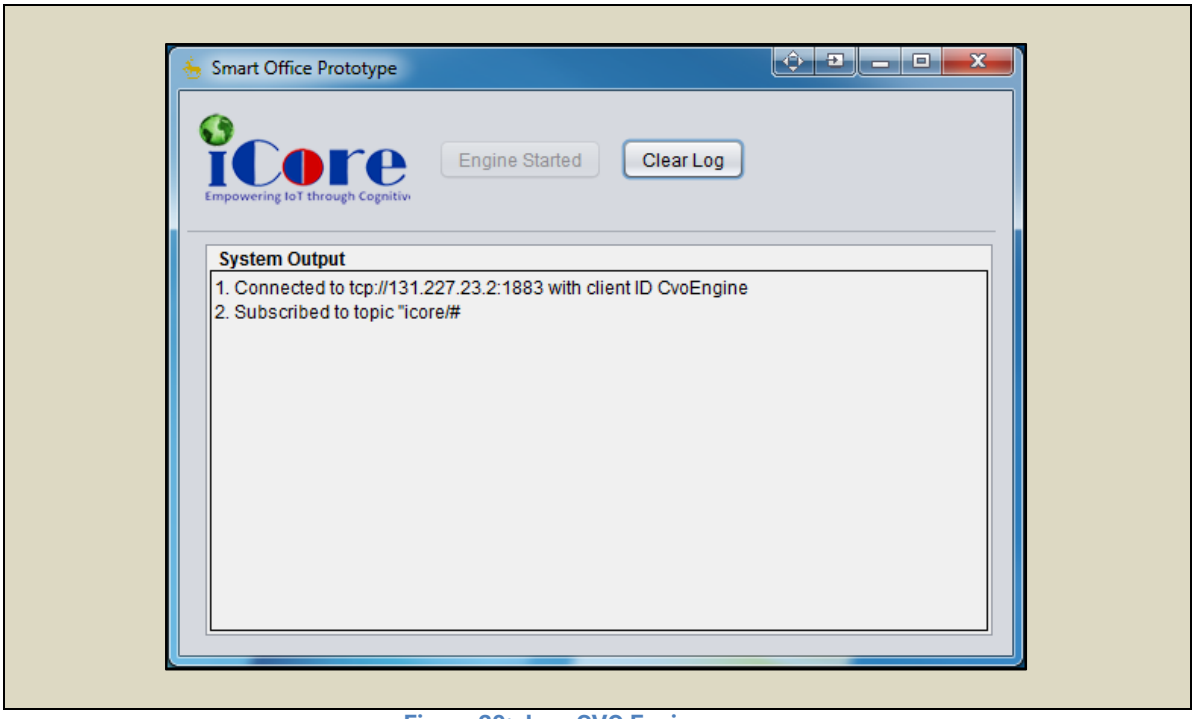


Figure 20: Java CVO Engine.

5.2.3 Introduction to the Guidance Mobile Application

On this part of the chapter, we will introduce the Guidance Mobile Application which is deployed to all the smart displays for our smart meeting scenario. Those displays are mounted across the building where the meeting venue is.

Two main functionalities are offered:

1. Visitor Check in / Guidance
2. Internet browsing

Apart from those two features, the rest of the device is “locked and not accessible” to a casual user. The underline reason for the locked state of the device is purely for security. Since this device is connected to the internal network of the host organisation and can handle sensitive data, only authorised users should access the rest of the functionalities.

Furthermore, in order to configure each device separately, a setting menu was implemented, but following the security aforementioned policies, this is hidden from the host organisation visitor. More details about that will be presented latter in this chapter.

Figure 21: Guidance application UI.

5.2.4 Attendant Mobile Application

Figure 22illustrates the mobile application that was designed and implemented for the meeting attendant. This application serves as an information displaying front-end, as well as a sensor measurement provider for our system back-end. Furthermore, via this application, the user can vote for the poll introduced at the Smart Meeting Break scenario.

- Some of the functionalities provided are:
- Map of the area
- Details of the meeting host organisation
- Web browsing

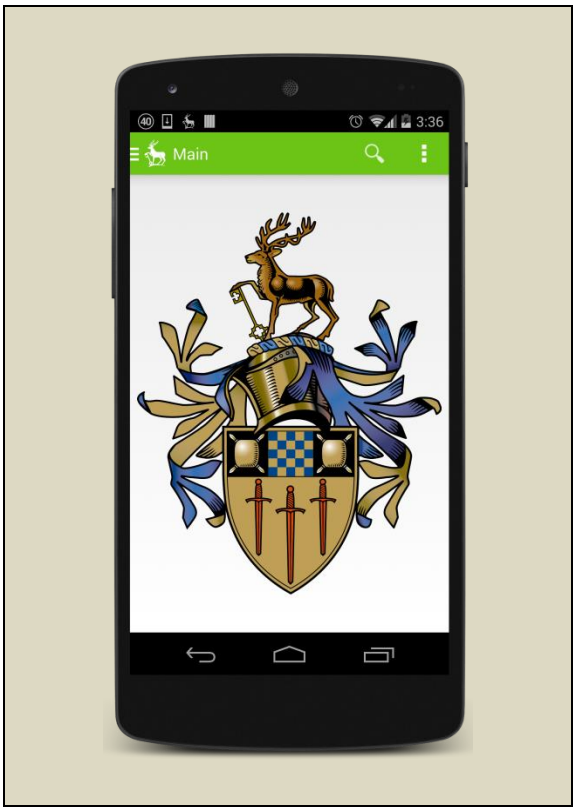


Figure 22: Meeting attendant application UI.

5.3 Implementation of Create Meeting System Requirements

In this section of our study a detailed presentation of the smart meeting creation system will be given.

The meeting organiser should press the button (1) illustrated in Figure 11in order to enter the appropriate UI that enables him/her to create a new meeting. This UI is a UI designed to help him set up all the details needed in order to create a new meeting. He can enter the name of the meeting and a short description that are going to be sent to all the participants. Furthermore, the UI offers the ability to choose easily the date, time and the meeting venue.

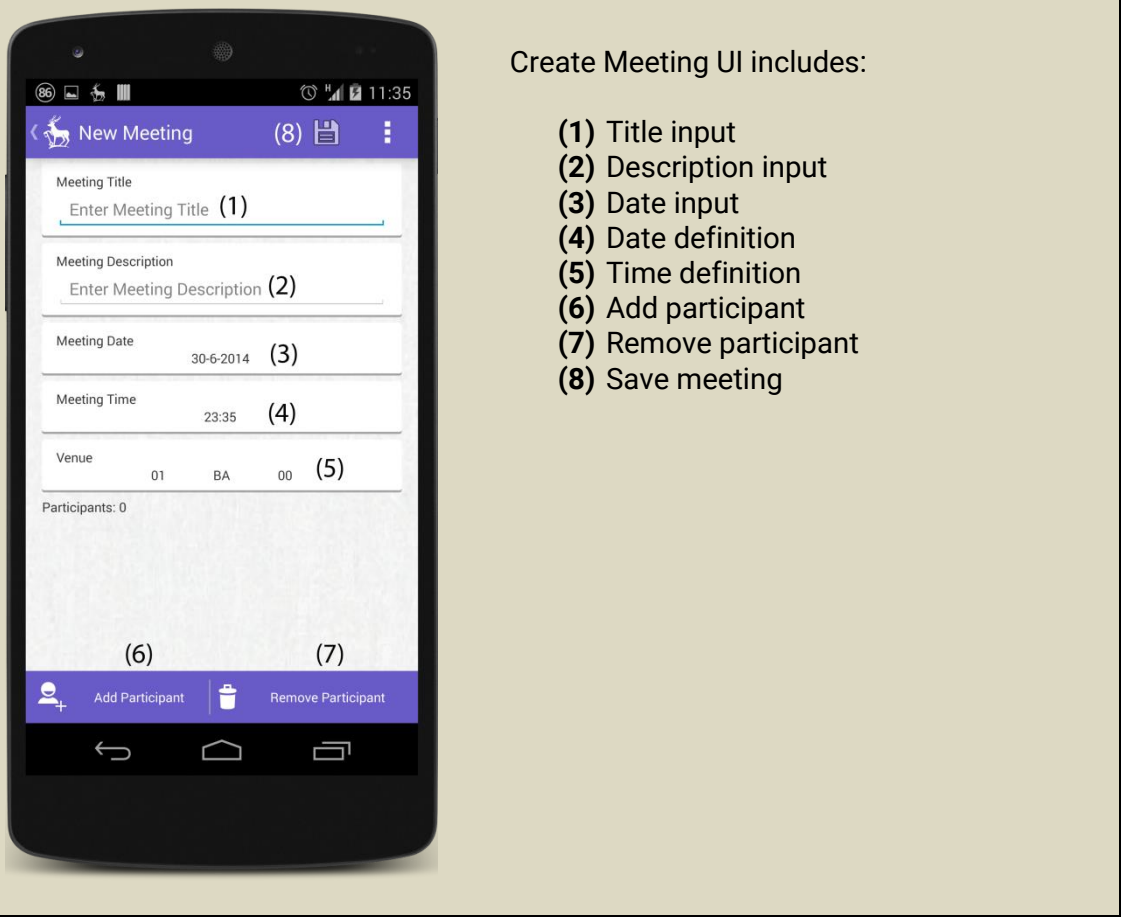


Figure 23: Create meeting UI.

5.3.1 New meeting form

The form is filled either by entering text directly from the keyboard of the device, or by selecting predefined values referring to the date, time, and the meeting venue. Also, as far as the meeting attendants are concerned, at the bottom of the layout, there are two specialised buttons to handle it. The first one called “add participant” is adding a new contact card in the screen. Vassilis can press it as much times as he needs to match his criteria.

- Name
- Surname
- E-mail

For each of the personal details, there is an on-click listener[3] that generates a popup dialog that enables the user to enter the data.

[3]http://developer.android.com/reference/android/view/View.OnClickListener.html

The second button is called “remove participant” and when pressed, it removes the last added participant. In case it is long-pressed[4] it clears the entire participants list.

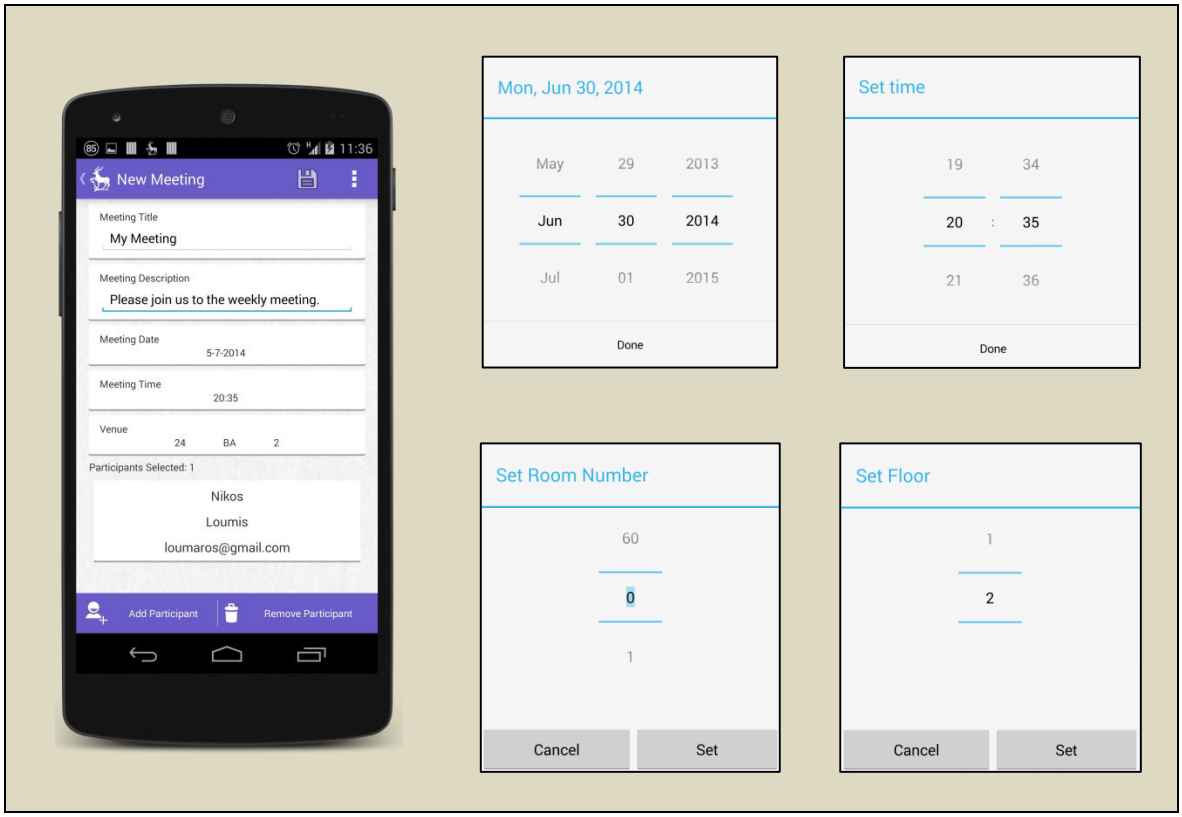


Figure 24: Completing new meeting form.

5.3.2 Save meeting

The Meeting Creation CVO is responsible to create a new meeting, store it, and inform the invited users using the auto-generated data. As input it receives a plethora of data such as:

- Meeting Title
- Meeting Description
- Meeting Venue
- Meeting Time
- Meeting Date
- List of Participants

All the above data are received as the payload of an MQTT message, in the form of a serialised java object. It is the CVO’s task to de-serialize and use them properly.

First, it does some background checks to make sure that the meeting venue proposed by the meeting organiser truly exists. If so, it new entry is created in the respective table of the connected database. Then, for each one of the meeting participants, a QR-

[4] http://developer.android.com/reference/android/view/View.OnLongClickListener.html

Code is generated containing information that will be used as input to the Smart Meeting Guidance scenario. Finally, a personalized email is also generated for each participant and is send on behalf of the meeting organiser to all the smart meeting invited users.

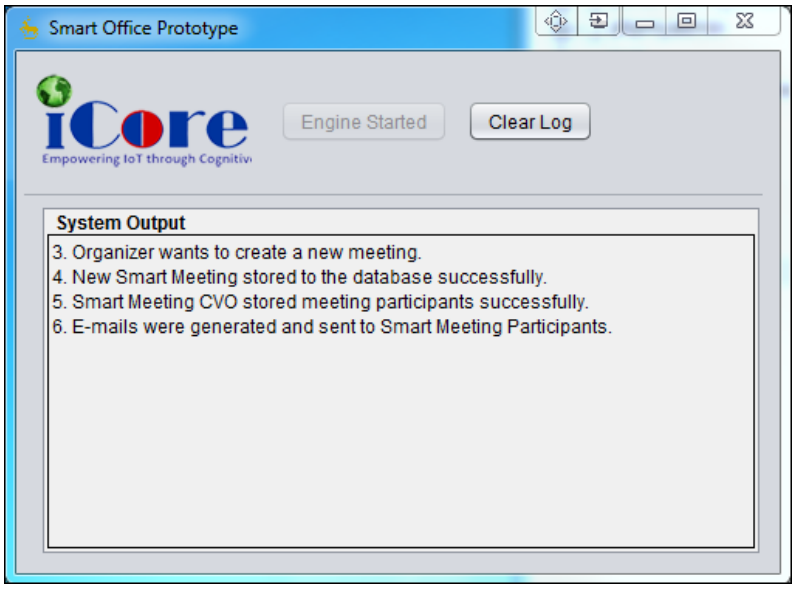


Figure 25: New meeting CVO notification.

The system generated emails are completely personalised and the QR code includes information about the meeting and each of the participants. Furthermore, details about the meeting organiser are attached to the mail body.

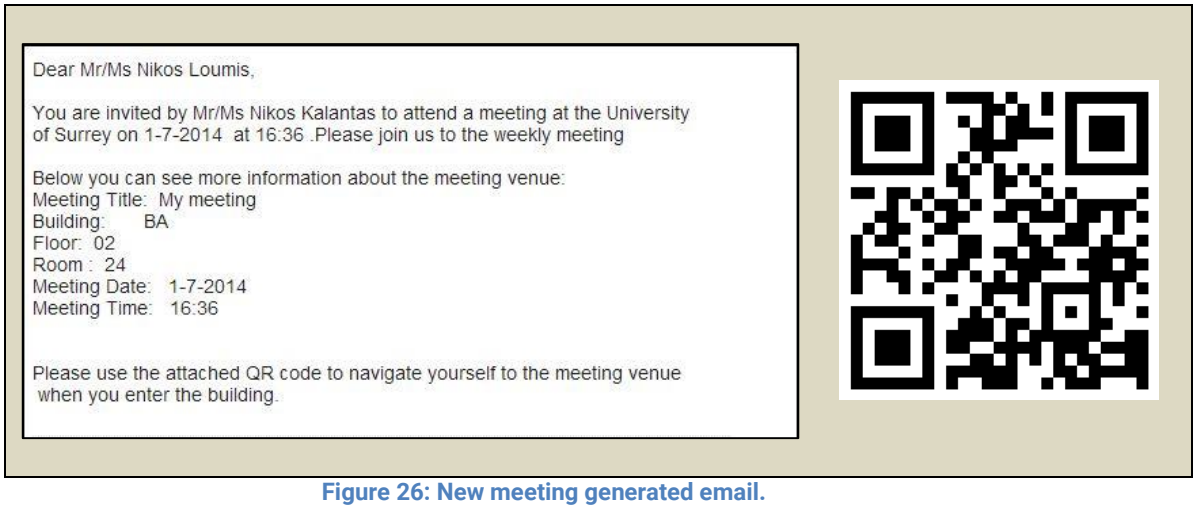


Figure 26: New meeting generated email.

As depicted in Figure 27, after the successful QR-code and email generating, the meeting is stored in the database. Foreign keys from the staff table and the room table are stored in each row.

id	name	description	time	date	staff_id	room_id
39	My meeting	Please join us to the weekly meeting	16:36	1-7-2014	4	54

Figure 27: New meeting database insert.

5.4 Implementation of Location Guidance System Requirements

The Smart Guidance is the second feature that is presented in this study. This component is supported by the Smart Guidance CVO and is responsible for matching incoming data with the database entries and providing real world.

When a user arrives at the BA building of the University of Surrey and enters from the main entrance, he can see the pre-installed Smart Meeting Guidance (SMG) system mounted on the wall. There, he scans his personalised QR-Code attached to the invitation email. The outcome of this action is the direction he should follow, in order to get himself to the meeting venue.

On the back end of the system, the QR-Code is read and analysed by the mounted panel and an MQTT message is sent to the appropriate CVO. When the MQTT message is received by the CVO, it is processed and the all the meeting data are fetched from a database. The output (direction) is sent back to the Smart Meeting Guidance system, through MQTT messages again.

Until the user arrives at his final destination, he will interact with more mounted panels and follow the same steps as described before. At the end point, an appropriate message will be shown.

The following figures show the screen presented to the meeting attendants prior to a QR code scanning event and after a QR code scanning event, when the Smart Meeting Guidance CVO has returned the direction to be displayed.

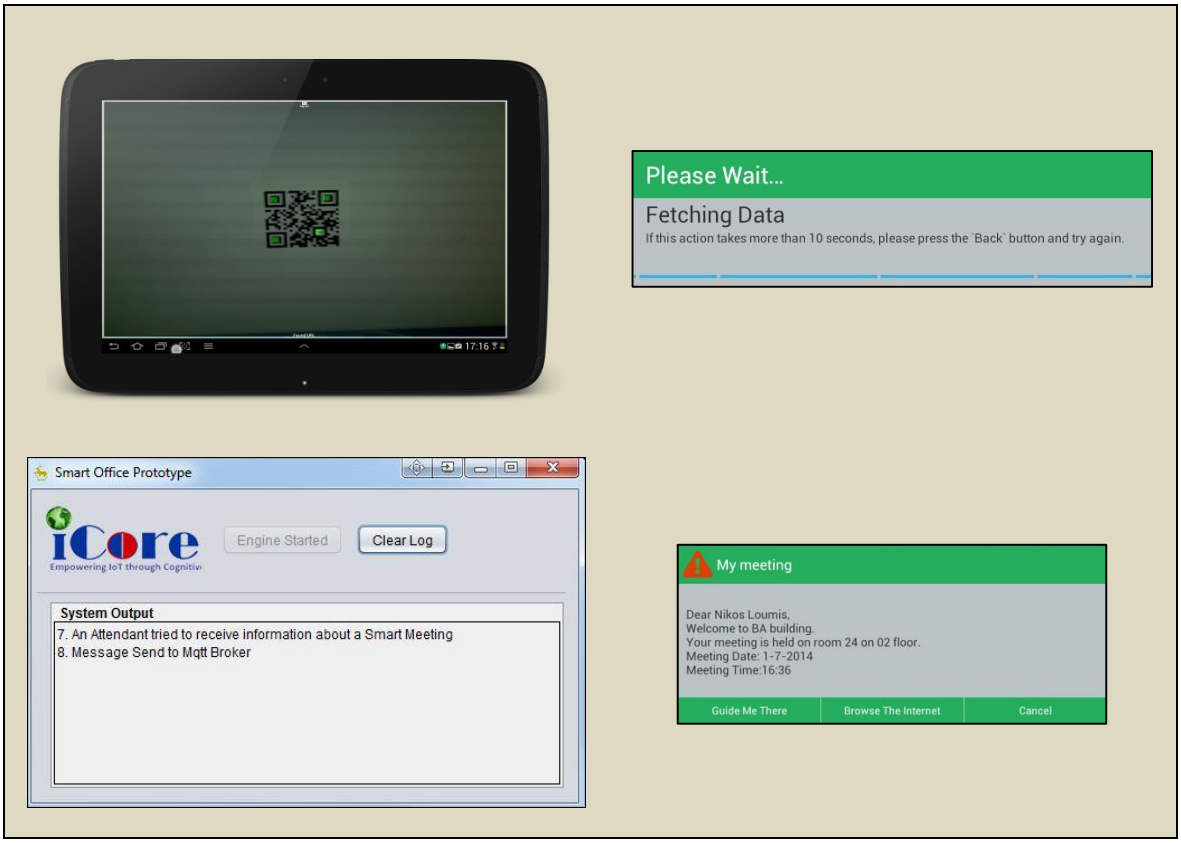


Figure 28: QR-Code scanning.

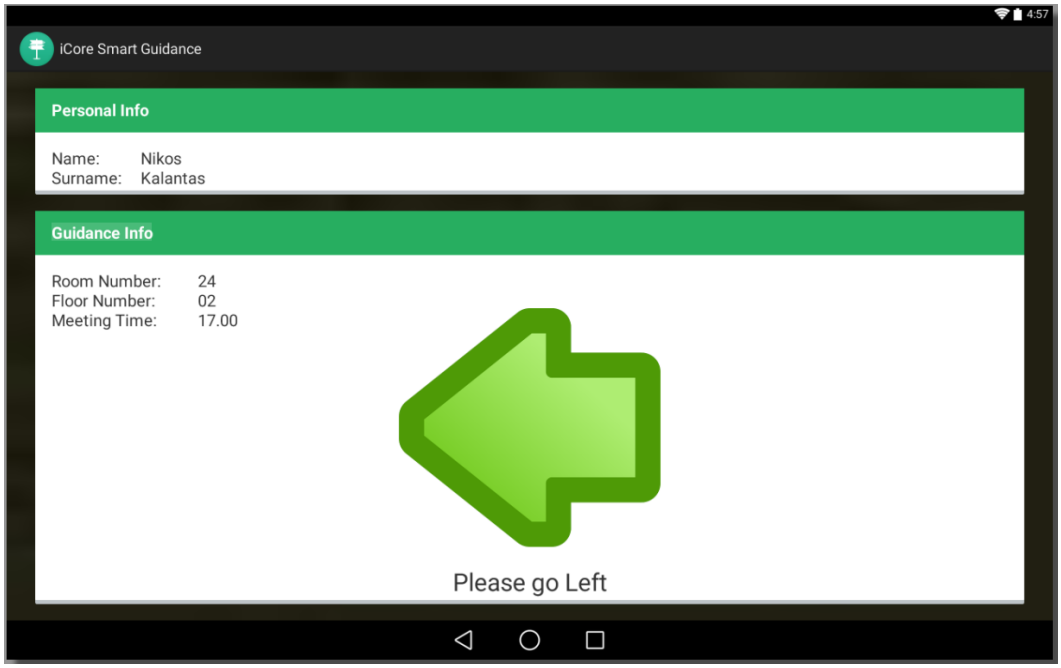


Figure 29: Direction received.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

5.5 Implementation of Meeting Recording System Requirements

The Smart Meeting Recording CVO calculates the best device to record a meeting session at any given time. The required inputs are:

- Mobile device's (VO) ID number
- Mobile device's (VO) motion status
- Mobile device's (VO) battery percentage value

Only the VOs that are assigned to the Smart Meeting Recording CVO are going to send their measurements back to it. In our SMR scenario, the meeting organiser wants to record the meeting session providing the optimization plan and the recording duration. According to the plan, the choice of the most appropriate device is altered as seen on Figure 30.

```
SELECT *
FROM recordmeeting
WHERE motion='false'
AND meeting_id='meeting'
ORDER BY battery ASC
Performance plan
```

```
SELECT *
FROM recordmeeting
WHERE  meeting_id='meeting'
ORDER BY battery ASC
Power efficiency plan
```

Figure 30: Smart Recording Optimization.

After the best match is calculated, a trigger message is sent by the CVO to the mobile device, in order to start the recording. The whole procedure is running in the background for as long as the meeting organiser wants to keep it alive.

The communication of the components is ensured by using MQTT messages, which are light-weight and adequately reliable.

5.6 Implementation of Meeting Break System Requirements

The Smart Meeting Break CVO decides on behalf of the meeting organiser when it is appropriate to have a break. As described previously in this report, there are two different approaches.

Automated Smart Break

The automated mechanism, that runs using the Esper Engine, takes as input the following data:

- Mobile device's (VO) ID number
- Mobile device's (VO) sound volume

Only the VOs that are assigned to the Smart Meeting Break CVO are going to send their measurements to the Esper Engine, also known as Smart Break Recommender CVO.

The Smart Break Recommender CVO is composed of six other CVOs such as the Noise Level Aggregation CVO, the Situation Detection CVO, the Situation Classification CVO, and the Situation Projection CVO. Furthermore two buffers are required to assist the Situation Detection CVO as well as the Situation Classification CVO with historical data.

All of the mentioned CVOs run in a Java environment. The Aggregation CVO subscribes to Noise Level events gathered from meeting attendants' phones. The meeting organiser gets notified about the recommendation the Situation Observer suggests every 30seconds. The notification value is changeable, but needs to be set before instantiating the Smart Break Recommender CVO. Depicted in Figure 31**Error! Reference source not found.** are the user interfaces of the three CVOs enabling Situation Awareness. The behaviour of the Noise Level Aggregation CVO is demonstrated by its console output. Figure 31**Error! Reference source not found.** it is hown that six noise level measurements from the microphone of an attendant's phone are aggregated over 30 seconds and the average, the standard deviation and the average deviation are calculated based on the values collected in that time frame. The specific value can be set on a CVO instantiation basis based on the specific scenario. Figure 31 shows that the Aggregation CVO produced an average value of 3.033 for the noise levels recorded over the last 30 seconds. It is way below the maximum value of 12.135, but the standard deviation of 4.773 and the average deviation of 3.598 indicate a high dynamicity in the meeting. Based on these statistical values the Situation Detection CVO discriminates the two hypothesises "NO Break" and "YES Break". If the Yes Break is less likely than NO Break the Smart Break Recommender does not suggest having a break soon. As shown in Figure 32 the Situation Classification CVO classifies the current situation (hypothesis NO Break with a confidence of 0.81, and the hypothesis YES Break with a confidence of 0.42) as not urgent (Urgency = 0.0). Thus the Situation Projection CVO calculates the time left until the next break to the maximum of 30 minutes from now

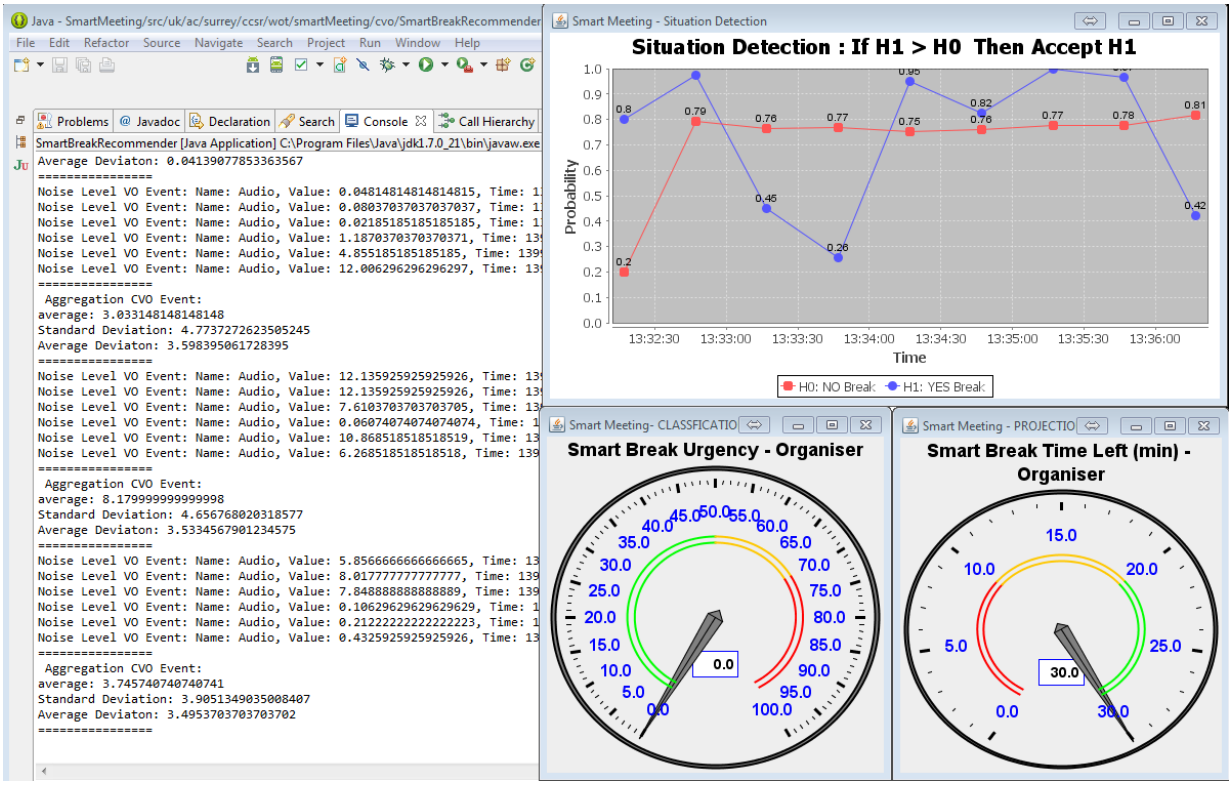


Figure 31: Situation Observer not recommending a break.

In Figure 32 the meeting situation looks different, a standard deviation of 0.27 as well an average deviation of 0.20 are hints to a situation of low activity in the meeting. The Situation Detection CVO reflects this by calculating a higher confidence for “YES Break” (0.85) than for “NO Break”. Thus the situation is now classified as urgent, which is reflected on the dial meter presented to the meeting organiser. The indicator is now in the red-marked part of the scale pointing at 73.8%. Also the Situation Projection CVO recommends having a meeting break soon (in 7.9 minutes).

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

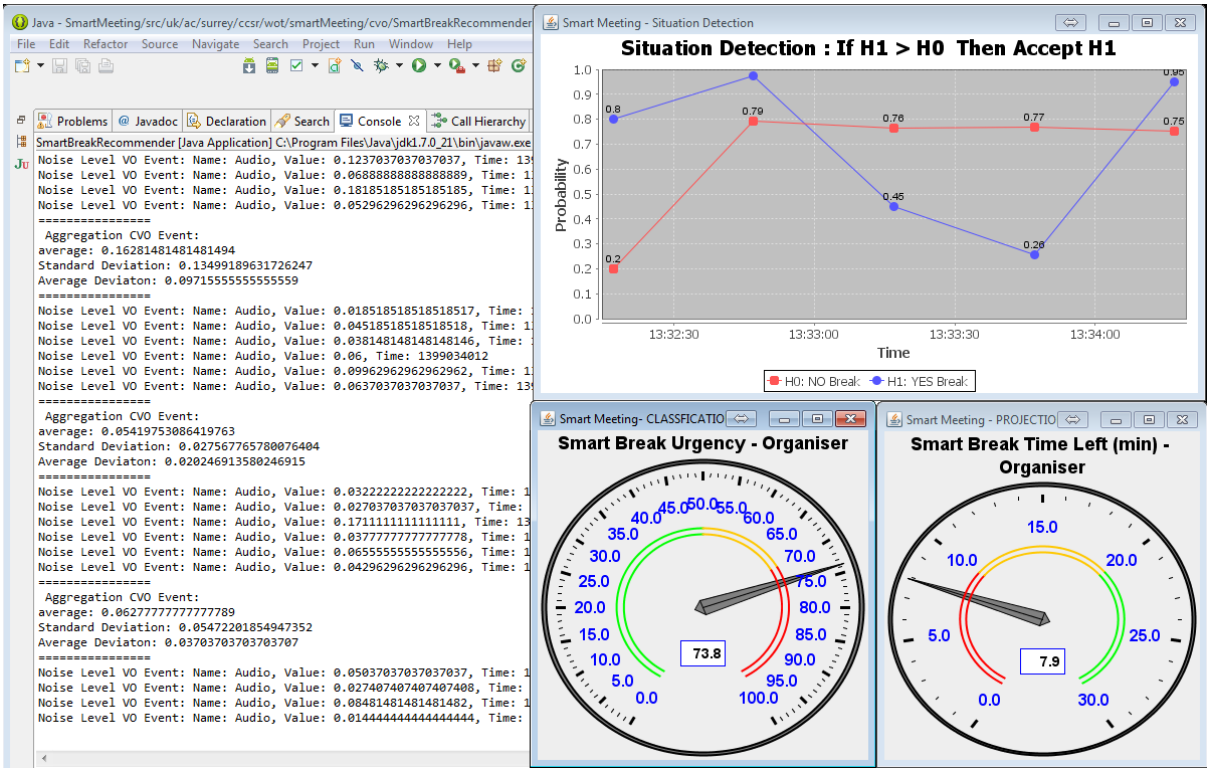


Figure 32: Situation Observer recommending a break

Since the Smart Break Recommender CVO is a Situation Observer operating on the speed layer of the RWK loop, the recommendation will adapt quite rapidly and frequently. The maximum value for “Time Left until Break” can be set by the meeting organiser during meeting organisation. The optimal value for this time is RWK that can be learnt after several iterations of the Smart Meeting [6].

Manual Smart Break

In this scenario, the meeting organizer wants to know if the attendants need to have a break, without interrupting the flow of the meeting. Via his mobile device, he starts the “Smart Break” procedure that is responsible to acquire feedback from the meeting participants and return the outcome to him. In order to trigger the Smart Break procedure, all he has to do is click the appropriate button on his mobile device screen. When the request is issued, policies and preferences are passed over to other components until finally the appropriate CVO is executed. In the Figure 16, we can clearly see all the steps followed in the Smart Meeting Break scenario.

In this section we will demonstrate all the steps mentioned above that take place when the Smart Meeting Break is called. First, when the appropriate button is pressed by the meeting organiser, it changes and informs the meeting organizer that the system is gathering feedback from the other attendants.

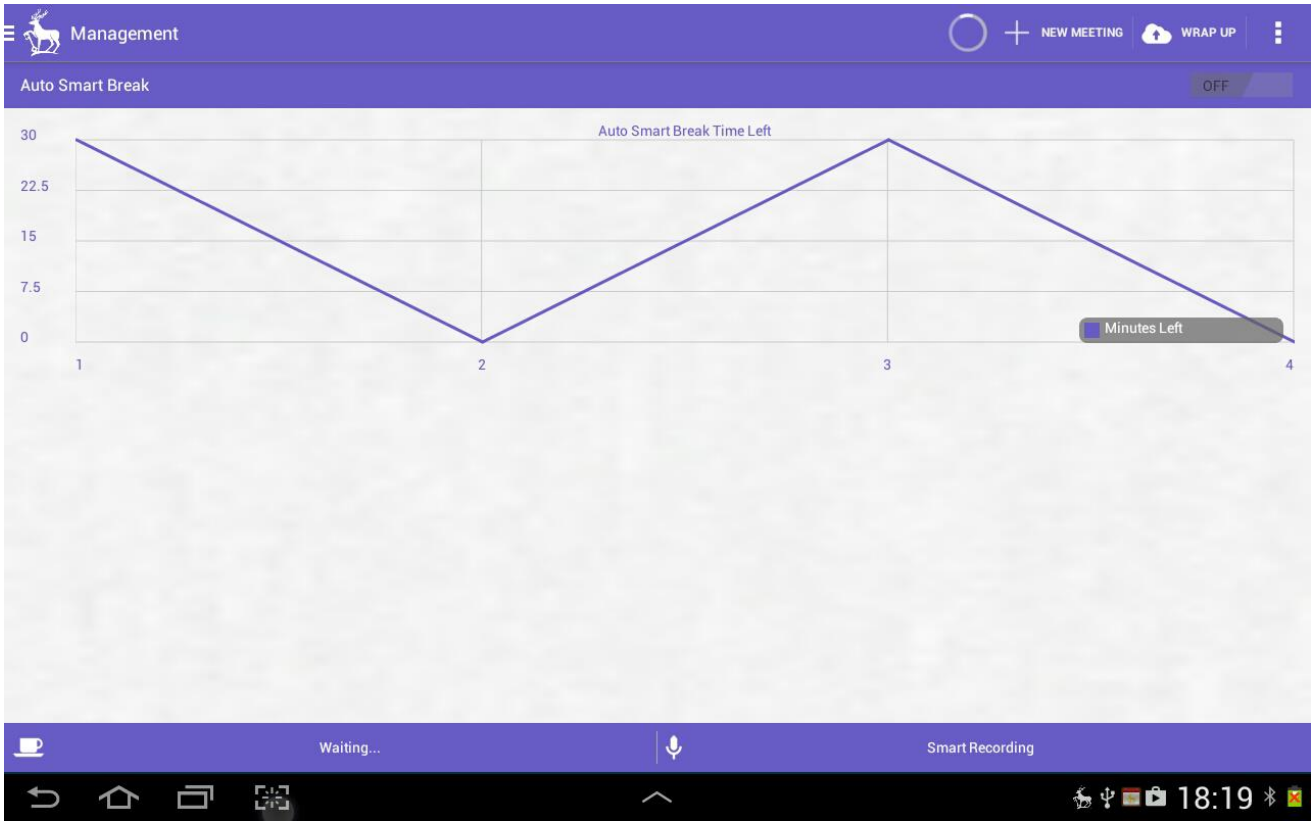


Figure 33: Smart Meeting Break button triggered.

At the same time, the MQTT trigger message is received and analysed by the Smart Meeting Break CVO as shown in the figure below:

Figure 34: CVO Front-End when Smart Meeting Break is triggered.

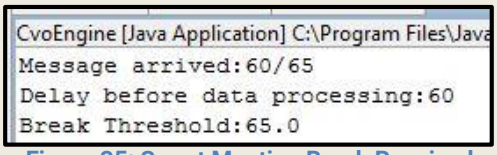


Figure 35: Smart Meeting Break Received Preferences Analysed.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

When the attendants VOs receive the trigger, a notification is shown to them, prompting to submit the feedback.

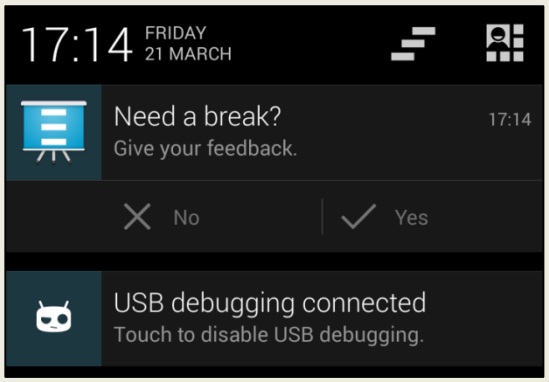


Figure 36: Providing Feedback via Notification Bar.

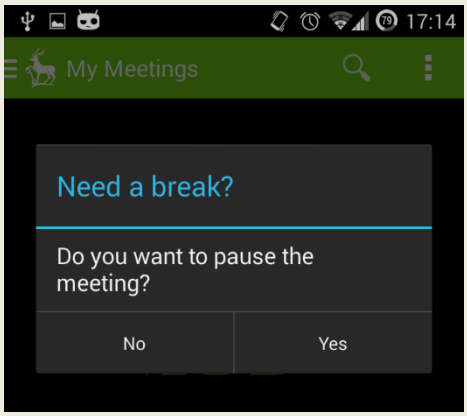


Figure 37: In-app Feedback Provide.

The Smart Meeting Break CVO is responsible to handle all the incoming feedback and store them in order to analyse them when it is necessary. Finally when the time arrives, it provides the meeting organiser the outcome;

```
message: f7727f44225e6be5/1
StoreFeedbackToDatabase: successfull insert
```

Figure 38: Feedback Received and Stored by the Smart Meeting Break CVO.

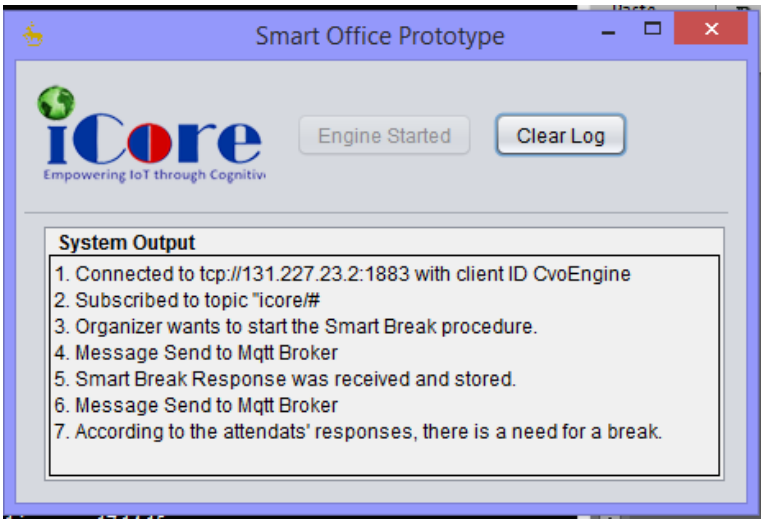


Figure 39: Smart Meeting Break Outcome Shown by the CVO.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

5.7 Implementation of Wrap-Up System Requirements

The Smart Meeting Wrap-Up CVO is responsible to gather all the data generated during a Smart Meeting and upload them to a publicly accessible location (i.e. Dropbox folder, ftp server).

When the meeting organiser wants to end the meeting, the Smart Meeting Wrap-Up CVO sends a trigger message to all the registered VOs. Then, all the remote devices, which are registered to this meeting, are looking inside their storage directory for Smart Meeting generated files. Then, each device creates an object of a class that encapsulates all the files with their metadata and sends them as an MQTT message. After the upload, the locally stored files are deleted, in order to ensure the meeting privacy.

In parallel, the Smart Meeting Wrap Up CVO, receives all the serialized objects and has to de-serialize each one of them. Finally, by using the provided metadata, it stores the files in the appropriate location and sends a completion message to the meeting organiser.

5.8 Implementation of Non – Functional System Requirements

In this section the implementation of non –functional system requirements such as the system settings are going to be discussed.

The user organiser can define all the variables concerning the meeting management by entering the appropriate menu page. As depicted in Figure 40: Organiser application settings UI. Figure 40, the following settings are provided to the user:

- **Break Delay**: the amount of time from the moment that users receive the break notification and the time that the system will calculate the potential meeting break outcome.
- **Break Threshold:** the minimum percentage of positive responses required in order to label a break poll as “break needed”.
- **Recording Optimisation**: selection between the two different schemes.
  - Battery saver, where the selection of the recording device is depended on the battery level on each device.
  - High Quality, where the recording device has to be stable to eliminate recording noise
- **Recording fragment duration**: the amount of time a device will record the meeting continuously.

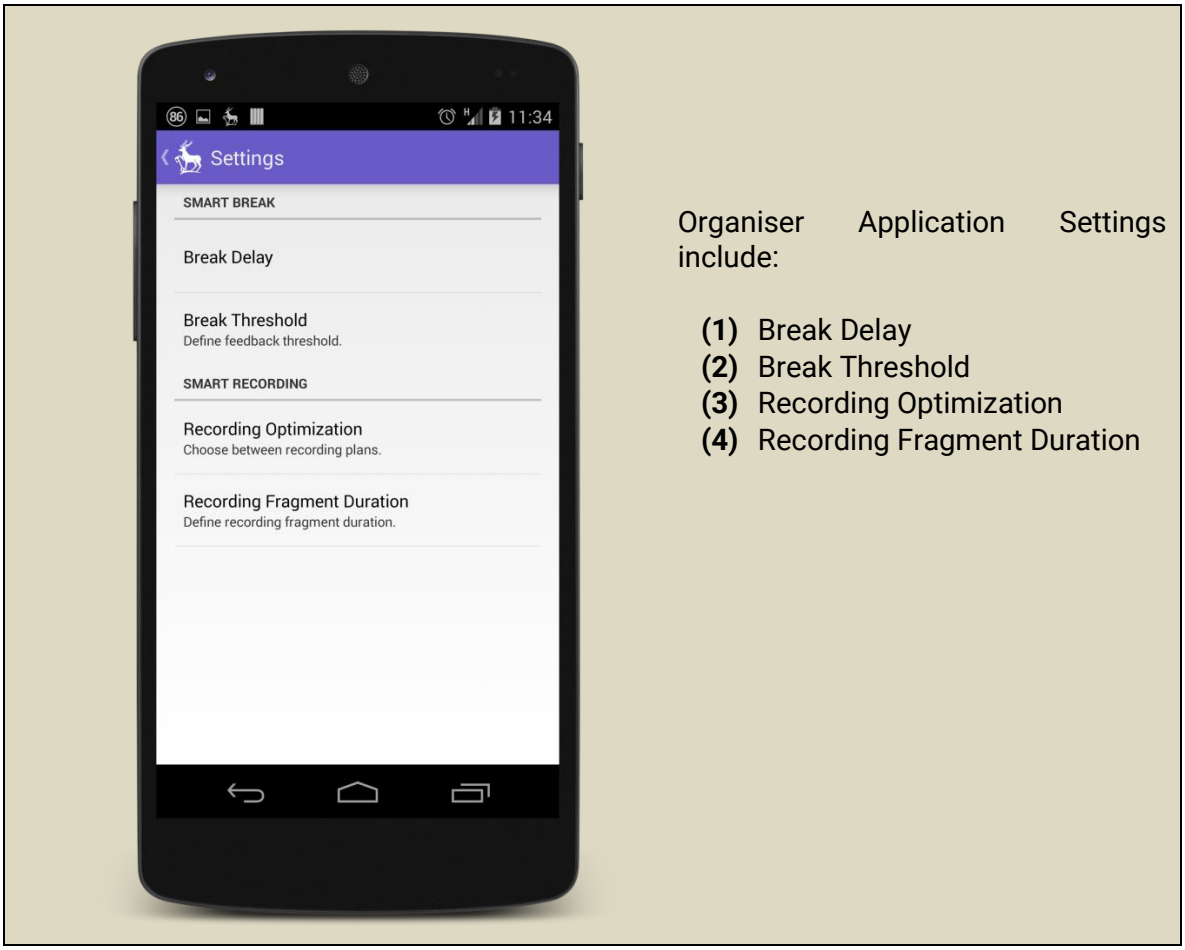
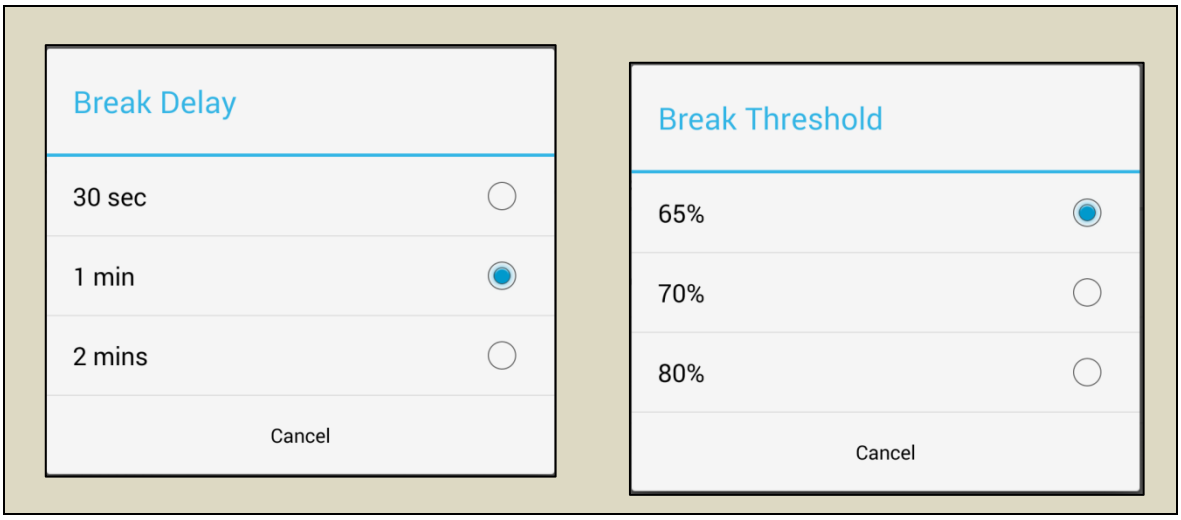


Figure 40: Organiser application settings UI.



Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

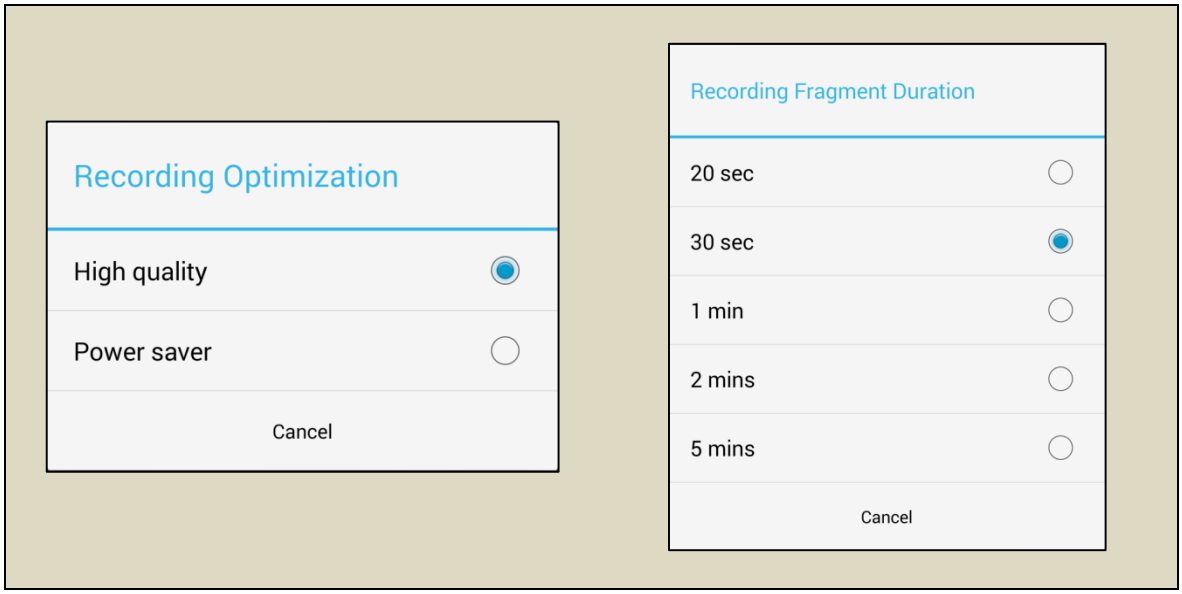


Figure 41: Organiser settings dialogs.

Navigation panel

The system's administrator is the only one with access to the navigation panel's settings. As shown in **Error! Reference source not found.**, the user can configure the ollowing:

- **Tablet position:** the predefined position of the tablet inside the building. Changing this affects the way the system calculates the next hop during the navigation process.
- **Navigation timeout:** The amount of time the system displays the navigation instructions before it resets itself.

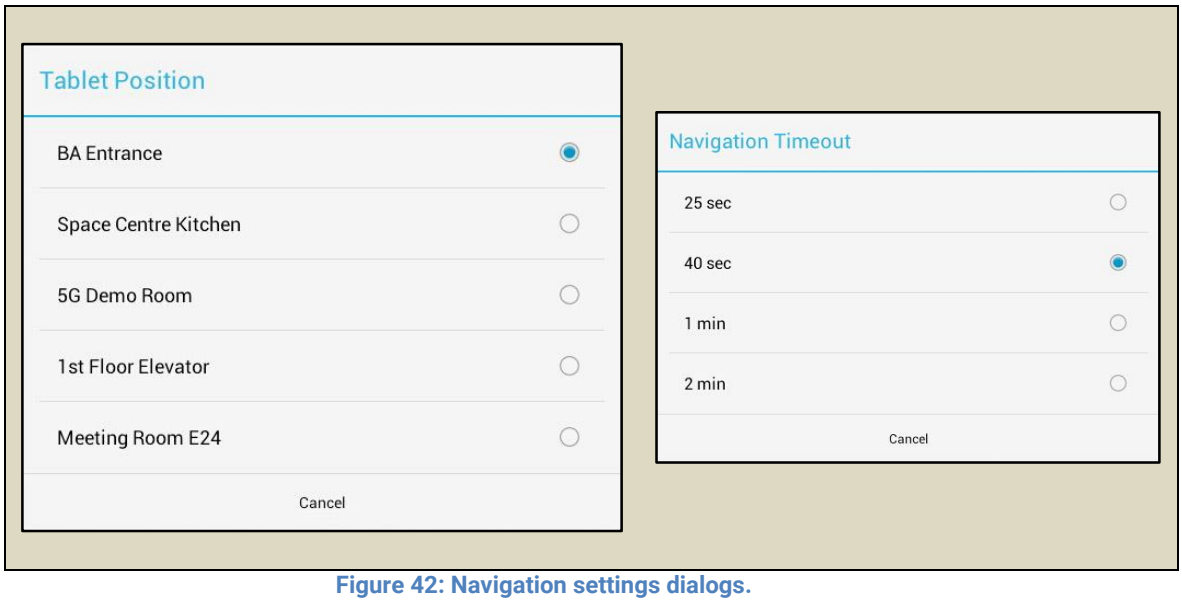


Figure 42: Navigation settings dialogs.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

5.9 Specification of interfaces

Interfaces used by the implemented components of the Smart Office PoC are as follows.

5.9.1 Interface between CVOs and VOs/Service Front-End

❖ IMqttCallback

Through this interface the involved components can handle the incoming MQTT messages and the possibility of a lost connection to the MQTT broker. It also enables an application/component to be notified when asynchronous events related to the client occur.

- **Method**
  - void messageArrived(IMqttTopic topic, IMqttMessage message)
- **Request**
  - **IMqttTopic topic**: complex structure that defines the MQTT topic at which each message was published to.
  - **IMqttMessage message**: complex structure that defines the received MQTT message. An MQTT message holds the application payload and options specifying how the message is to be delivered The message includes a "payload" (the body of the message) represented as a byte[].
- **Datatype Structure**
  - **IMqttTopic:** [String, int]
- **Included Parameters**
  - ***(a) String getName*** : it's a function that returns a variable that represents the name of the MQTT topic, where the incoming message was published
  - ***(b) int getQoS***: it's a functions that returns an integer that represents the Quality-of-Service (QoS) that we have defined for our MQTT messages.
- **Datatype Structure**
  - **IMqttMessage:** [int, byte[ ], boolean, boolean]
- **Included Parameters**
  - ***(a) int getQoS:*** it's a function that returns an integer that represents the Quality-of-Service (QoS) we have defined for our MQTT messages.
  - ***(b) byte[ ] getPayload:*** it's a function that returns an array of bytes that includes the actual information published through the MQTT protocol.
  - ***(c) boolean isRetained*:** this is a variable that is used to determine if each publication was recently retained to the MQTT broker, before it was sent.
  - ***(d) boolean isDuplicate*:** this is a variable that corresponds to the additional attribute of the MQTT messages about whether it is a duplicate. The duplicate flag is only set if the quality of service is "at least once" or "exactly once". If the message was sent previously,

and not acknowledged quickly enough by the MQTT client, the message is sent again, with the duplicate attribute set to true.

- **Method**
  - void connectionLost(Throwable throwable). This method is called when the connection to the server is lost.
- **Request**
  - **Throwable throwable**: it's a variable that includes information about the error occurred when the connection to the server was lost. It is a response to any server-side problems encountered after receiving a positive connection acknowledgment.

❖ IMqttClient

This interface enables an application/component to communicate with an MQTT server using blocking methods. This interface allows applications to utilize all features of the MQTT version 3.1 specifications.

- **Method**
  - void setCallback (IMqttCallback callback) . Sets the callback listener to use for events that happen asynchronously.
- **Request**
  - **IMqttCallback callback.** It is the class to callback for events related to the client.

- **Method**
  - void publish (IMqttTopic topic, IMqttMessage message). Publishes a message to a topic on the server and return once it is delivered
- **Request**
  - **IMqttTopic topic**. complex structure that defines the MQTT topic at which each message was published to.
  - **IMqttMessage message**. complex structure that defines the MQTT message to publish. An MQTT message holds the application payload and options specifying how the message is to be delivered The message includes a "payload" (the body of the message) represented as a byte[].

- **Method**
  - void subscribe (IMqttTopic topic)
- **Request**
  - **ImqttTopic topic.** Complex structure that defines the MQTT topics that we want to subscribe to.

- **Method**
  - boolean isConnected( ). Determines if this client is currently connected to the server.
- **Response**
  - **boolean connectionStatus:** [true (if connected), false (otherwise)] .

- **Method**
  - void connect (IMqttConnectOptions options). Connects to an MQTT server using the specified options.
- **Request**
  - **IMqttConnectOptions options.** Interface that describes a set of connection parameters that override the defaults.
- **Datatype Structure**
  - **IMqttConnectOptions:** [setCleanSession, setKeepAliveInterval, setUserName, setPassword, getCleanSession, getKeepAliveInterval, getUserName, getPassword]
- **Included Methods**
  - ***(a) void setCleanSession.*** Sets whether the server should remember state for the client across reconnects.
  - ***(b) void setKeepAliveInterval:*** Sets the "keep alive" interval.
  - ***(c) void setUserName:*** Sets the user name to use for the connection.
  - ***(d) void setPassword:*** Sets the password to use for the connection.
  - ***(e) boolean getCleanSession:*** Returns whether the server should remember state for the client across reconnects.
  - ***(f) int getKeepAliveInterval:*** Returns the "keep alive" interval.
  - ***(g) String getUserName:*** Returns the user name to use for the connection.
  - ***(h) char[ ] getPassword:*** Returns the password to use for the connection.

- **Method**
  - void disconnect( ). Disconnects from the server.
- **Method**
  - void ping ( ). Sends a PING message to the broker to examine if the Broker is up and running.

5.9.2 Interface IMqttClientFactory (Local)

Through this interface, a new MQTT client can be created.

- **Method**
  - IMqttClient create(String host, int port, String clientId, IMqttPersistence persistence).
- **Request**
  - **String host**. A string describing the URL address of our MQTT broker
  - **Int port:** It's a variable that represents the port that is open for MQTT connections at the broker.

- **String clientId:** A character identfier which must be unique across all MQTT connections to the broker. Max length is 23 characters.
- **IMqttPersistence:** A MqttPersistence implementation for persisting MQTT state across connections.

- **Response**
  - **IMqttClient.**

- **Datatype Structure**
  - **IMqttPersistence:** This Interface is null for the time being, as it does not serve any usefull function for the Smart Office PoC.

5.9.3 MQTT Topics (Communication Parameters)

In this section we will enlist all the MQTT topics that are used throughout our proof of concept prototype.

Table 1: MQTT Topics used for communication purposes

No.	Topic Name	Description
1	icore/cvo/[id]/feedbackRequest	Used in order to create the feedback notification at the Attendants VO.
2	icore/cvo/[id]/recordMeeting	Topic that contains information about the status of the SMR service.
3	icore/cvo/[id]/recordingDevice	Topic used to inform the available devices to start recording at the SMR use case scenario.
4	icore/cvo/[id]/wrapUpMeeting	When called from the CVO Engine, it triggers the SMW scenario to the VOs.
5	icore/cvo/[id]/autoSmartBreakControl	In this topic, we can start and stop the automated SMB scenario.
6	icore/cvo/[id]/feedbackOutcome	Used in order to inform the meeting organizer about the manual SMB outcome.
7	icore/cvo/[id]/autoSmartBreakUrgency	Used from the SO in order to provide the urgency for a Smart Meeting Break.
8	icore/cvo/[id]/autoSmartBreakTimeLeft	Used from the SO in order to provide calculated time left until the next Smart Meeting Break.
9	icore/attendant/[id]/feedbackResponse	Used by the attendants VOs to send back the feedback acquired.
10	icore/attendant/[id]/ recordMeeting	This topic encapsulates data gathered by the attendants' VOs about the SMR scenario.
11	icore/attendant/[id]/ uploadRecording	This topic is used in order to upload the meeting recording fragments in the SMW use case scenario.
12	icore/organizer/[id]/ recordMeeting	When called it triggers the SMR service request.
13	icore/organizer/[id]/ wrapUpMeeting	When called it triggers the SMW service request.
14	icore/organizer/[id]/manualSmartBreak	When called it triggers the SMB service request.
15	/ccsr/meetings/add	This topic is published when the meeting participant scans his QR code at the panel, containing information on the panel at which the scan has taken place and meeting details in the payload. The subscribers include the SmartGuidance CVO.

6. Conclusion

This project aimed to specify and develop a Smart Meeting Application suite which helps meeting organisers and participants throughout the life cycle of a meeting. Other meeting applications were not covering the full spectrum of needs and where targeting a simple task such as recording, or sending emails.

The approach of this project was to combine the advantages and features presented in similar tools and eliminate, where possible, the side-effects. The production cost was minimised by using open source tools and operating systems. Furthermore, the system introduces an eco-friendly paper-less approach utilising digital QR codes as invitations.

The system objectives derived by the use cases and tried to cover the whole range of needs under all the perspectives during a meeting. Creating, guiding, recording, breaking, and wrapping-up a meeting were out tangible goals.

The meeting creating system requirements defined the behaviour of the system in allowing a meeting organiser to create a meeting using his mobile device, in an easy and robust way. In achieving this requirement, an Android application was developed which communicated with a Java server, an SQL database and various open source java libraries.

The visitor guidance system requirements specified how a meeting participant will find himself to the meeting venue, inside the host building. In order to support this functionality, several Android tablets were deployed around the host building. A native android application was deployed to those devices in order to scan QR code invitations, calculate the shortest path, and display navigation directions to the end users.

The meeting break system requirements defined the system behaviour on order to pause an on-going meeting. To achieve this requirement, an Android application was developed, capable of opportunistic and participatory sensing. Two backend Java servers were implemented to aggregate the aforementioned measurements and to provide an outcome.

The meeting recording system requirements specified how the system should behave when it is recording an on-going meeting. To serve this purpose, the aforementioned android application was enhanced with additional sensor gathering capabilities. The back end java server was also enriched with an additional functionality that would select the most appropriate device to start the recording.

Meeting wrap-up was the last group of system requirements that was defined. It was indicating a way of wrapping up the meeting by fetching all the generated data to a remote and accessible location. All the aforementioned systems adapted with new functionalities.

All the communications were handled by the MQTT protocol. The de facto IoT communication protocol proved to be robust and easy to integrate to all the systems implemented.

After the end of the implementation and testing stage, we realised that a Smart Meeting application suite is something really useful for organisations of all sizes. For future work, an Outlook plugin could be introduced, which will well meeting organisers even more with the meeting management.

Nikolaos – Stylianos Loumis ME12051 loumis.nikos@gmail.com

7. References

[1] Shari Lawrence Pfleeger, *Τεχνολογία Λογισμικού*.: Κλειδάριθμος, 2003.

[2] Nilesh Parekh. Intelligent Life on the Web. [Online]. http://www.buzzle.com/editorials/1-5-2005-63768.asp

[3] Madison Smith. PubNub. [Online]. http://www.pubnub.com/blog/what-is-mqtt-use-cases/

[4] Roger Light. Mosquitto Broker. [Online]. http://mosquitto.org/man/mqtt-7.html

[5] HiveMQ Team. MQTT Essentials Part 6: Quality of Service 0, 1 & 2. [Online]. http://www.hivemq.com/mqtt-essentials-part-6-mqtt-quality-of-service-levels/

[6] Nikos Loumis et al. (2014) [Online]. http://www.iot-icore.eu/attachments/article/89/20140331%20d6.3_smartoffice_final.pdf

[7] Herman Kopetz, *Real-Time Systems*.: Springer, 2011.