End User Programming and Perceiving the Environment
Acknowledgement

This thesis is submitted for the degree of “Master of Science specialized in Software Technology” at the Linnaeus University, Växjö. We would like to express our sincere thanks to our supervisor Professor “Jesper Andersson” for his enthusiastic and expert guidance. We would like to show our gratitude to our parents for their love, encouragement, support and inspiration. Finally we would like to thank all my colleagues and friends who contributed to this project.
Abstract

The current mobile technology is growing very fast, and it has already included many excellent features on mobile phones. Current mobile technology research mainly aims at fast execution and enriching mobile phone with more and more portable features.

This thesis aims at a thorough investigation of Android enabled mobile phone’s capabilities. Mobile devices today may utilize several types of sensors. These sensors may be used to sense the environment, in which, the device is situated, directly or indirectly. The main idea is to connect sensing with end-user programming (EUP). We enumerate some challenges that will be addressed in this thesis. This thesis is exploratory, which implies that it contains a survey of available techniques, tools, and approaches particularly in the mobile device domain. In addition the thesis will also explore and identify the limits with focus on the Android platform.

Thesis Implementation languages are mainly Core Java, Android 2.1 programming language and XML. We have developed Sensors framework using Android API, which gives latest value of all possible sensors used in mobile phones, and notify end user programming about sensor value change. We have also developed ECA (Event Condition Action) framework, for end user programming to handle end user configuration changes.

The thesis research and implementation results helped us to find answers for various challenges on mobile phone domain. Few of them to mention are, gathering information about different kinds of sensors, how they can be used for sensing real time environment, how we can combine different sensing results to identify particular action, identifying framework and domain language for end user programming.

Keywords: Sensors, End user programming, ECA framework, Android, XML, Core Java
# Table of Contents

1. **Introduction** ................................................................. 1  
   1.1 Objectives ........................................................................ 1  
   1.2 Motivation ........................................................................ 1  
   1.3 Goal ................................................................................ 2  
   1.4 Thesis research questions .................................................. 2  
      1.4.1 Sensors ...................................................................... 2  
      1.4.2 End user programming (EUP) ......................................... 2  
      1.4.3 Adaptation ................................................................. 2  
   1.5 Solution tools .................................................................... 2  
   1.6 Structure of the thesis ....................................................... 2  

2. **Backgrounds and Review of the State of the Art** .................. 4  
   2.1 Background and existing surveys on context aware mobile applications ........ 4  
   2.2 Background on sensors and its integration with mobile phones .................... 5  
   2.3 Background on android .................................................................. 6  
   2.4 Background on end user programming ............................................ 7  

3. **Proposed Solution to Research Questions** ........................... 8  
   3.1 Sensors ........................................................................... 8  
   3.2 End user programming .......................................................... 10  
   3.3 Adaptations ....................................................................... 11  

4. **Design and Implementation** ................................................. 12  
   4.1 Software development model ................................................... 12  
   4.2 Requirement analysis ............................................................. 12  
      4.2.1 Functional requirement .................................................. 12  
   4.3 System design .................................................................... 13  
      4.3.1 Low level design for sensor framework ............................... 14  
      4.3.2 Low level design for EUP framework ................................. 14  
   4.4 Implementation .................................................................. 16  
      4.4.1 Implementation details on sensor framework ...................... 17  
      4.4.2 Implementation details on EUP framework ....................... 19  
   4.5 Integration ....................................................................... 22  
   4.6 Testing .............................................................................. 22  

5. **Evaluation** ....................................................................... 23  
   5.1 Evaluating current implementation with respect to earlier surveys ................ 23  
   5.2 Goal criteria testing ............................................................. 23  
   5.3 Actors .............................................................................. 23  
   5.4 Use case scenarios .............................................................. 23
6. Conclusion and Future Work ................................................................. 31
   6.1 Conclusion .......................................................................................... 31
   6.2 Future Work ....................................................................................... 31
7. References ............................................................................................. 32
List of Figures

Figure 2:1 Android Architecture.................................................................6
Figure 3:1 Sensor Decision Tree.................................................................8
Figure 3:2 Interaction between Sensors and EUP........................................11
Figure 4:1 High Level Architecture of Thesis Project.................................13
Figure 4:2 Class Diagram of Sensor Framework........................................14
Figure 4:3 User Context Collaborated with ECA Framework .....................15
Figure 4:4 Class Diagram for EUP Framework........................................16
Figure 4:5 GUI for User Defined Rule.......................................................22
List of Abbreviation:

1. API - Application Programming Interface
2. CCD - Charge Coupled Device
3. EUP - End User Programming
4. ECA - Event Condition Action
5. GUI - Graphical User Interface
6. OS - Operating System
7. TEA - Technology Enabling Awareness
8. VM - Virtual Machine
9. XML - Extensible Markup Language
1. Introduction
Our thesis main focus is to explore context aware mobile computation, its current challenges, limitations, disadvantages etc. “Context Awareness” signifies connecting environment change with any possible device. Here context signifies any environment change, like time change, any action change, or any location change.

Present world represents new era of fancy, stylish and automated mobile technologies. Many sensors are being used by mobile phones to achieve automated and excellent solutions. For example, at present mobile phone use a sensor called “Accelerometer” to sense motion, and hence by sensing motion, many mobile phones perform different actions like changing MP3 song automatically etc. Here motion is a type of context change, which is sensed by sensor Accelerometer.

There are many sensors present, which are still undiscovered in mobile domain. This thesis main aim is to investigate about different sensors which can be used in “Android” mobile phones to detect context change, and categorize them according to their usage. Moreover, we need to investigate the possibilities to utilize context as a driver and configuration parameter for device adaptations. Connected to this challenge is how the adaptive behavior should be controlled and configurable by device users. EUP plays an important role here.

The objective, motivation, goal, thesis research questions and structure of our thesis report are described in detail under below sub sections:

1.1 Objectives
The main objective of this thesis is to present implementation result on Android based mobile phone, i.e. if real world environment changes then phone configuration should also change automatically. The main objective here is to control phone configuration change by the environment change. Sensor framework should handle sensing environment, collecting latest sensor data and sending notification to end user programming about sensor value change. EUP part should combine different sensor value results and change the phone configuration automatically, without user’s intervention.

1.2 Motivation
Mobile phones have become mandatory part of every individual’s life in present world. Every individual wants to keep important files, access important data, store entertainment sources etc in their pocket size mobile phones. That’s the reason mobile technology has been revolutionized to the greatest extent in past few years. Our thesis research will add more interesting and much awaited features in mobile phones. Our thesis mainly concentrated on phone automatic configuration change, if real time environment changes. This feature will add more convenience to end user, because then he needs not to perform some phone setting changes explicitly.

For example: User wants his mobile phone to be set to silent mode automatically, if he is in University and current time is between 1 pm to 4 pm. So, here our thesis can provide a good solution. Sensors framework will sense user’s current location, and clock timing. As soon as user’s current location is university and time is between 1pm to 4pm, sensors framework will send notification to EUP and EUP will check if both condition satisfies. If yes, then it will automatically change the phone mode to silent, without user’s intervention. Otherwise, there will be no change done in the phone settings.

Our thesis research can add more features like connecting phone with surrounding non mobile devices, accessing internal phone database etc. Increased demand of more mobile features and increased dependency of user on their mobile phones has motivated us to investigate more on this domain. Android platform especially the new version, i.e. 2.1
provides sufficient set of in built API to implement mobile functionalities. Development on Android is pretty simple and fast.

1.3 Goal
The main goal of this thesis is survey of available techniques, tools, and approaches particularly in the context aware mobile computation domain. The goal primarily includes research on implementation language supported on Android platform, implementing sensors framework to sense real world environment and implementing End user programming framework. We need to check in our implementation result that, if real world environment changes, then how EUP changes the phone configurations automatically.

1.4 Thesis research questions
There are three possible sub parts of this thesis work. Challenges and questions to be addressed for each sub part are explained below:

1.4.1 Sensors
- What sensors will be used to sense the environment?
- How to sense real world environment change?
- How do we model the environment?
- How to combine multiple sensor results?

1.4.2 End user programming (EUP)
- What are the possible EUP types and their classification?
- What are the possible domain languages, available for EUP implementation?
- How EUP will interact with sensors?

1.4.3 Adaptation
- How may End user programming influence device adaptation?

1.5 Solution tools
We have used Core Java and Android 2.1 API to develop our thesis implementation. Android 2.1 has very enriched library and supports huge set of API to access hardware sensors like Accelerometer, as well as software sensors like WIFI, Bluetooth etc. Android also supports API to build nice graphical user interface (GUI) and to control phone settings like, changing volume etc. As compared to previous mobile platforms, development on Android platform is very easy; because of it’s in built APIs. Android also supports fantastic GUI features, which makes it first choice for end users.

We have chosen Core java, because Android platform supports only core java as implementation language [1].

1.6 Structure of the thesis
The overall structure of thesis is organized in following way. Chapter 2 contains information about background on thesis’s main idea. Under this chapter, we discuss foundation of our work, history on mobile technology evolution, existing work done by other researchers on same context etc. Under chapter 3, we are presenting our proposed solution to all research questions, mentioned under section 1.4. Under chapter 4, we are presenting details about designing and implementing our proposed solution to thesis research problems. The discussion in this chapter also includes, requirement analysis, choosing software development model, design details of each module, their integration and testing. Under chapter 5, we are providing different use cases and evaluation of our implementation with respect to these use
cases, and existing work on this area. Finally, chapter 6 describes conclusion and future work on this thesis context.
2. Backgrounds and Review of the State of the Art

As stated in chapter 1, our research is mainly towards investigating the possibilities to utilize context as a driver and configuration parameter for device adaptations. Moreover the challenge also includes how the adaptive behaviour should be controlled and configurable by device users. There has been some research and work already done in this area. We are inspired by previous work done by many researchers in sensor networks and Android framework. Under this chapter, we will describe about background on context aware computation, sensors, EUP and Android.

2.1 Background and existing surveys on context aware mobile applications

- **Background**
  
  To start with general understanding on context and context awareness, here we will first describe what exactly context means.

  “Context is the set of environmental states and settings that either determines an application’s behavior or in which an application event occurs and is interesting to the user.”

  In simple words, Context signifies any environment, which could be any action, location, services etc. Many researchers have defined and categorized it in different ways, based upon their nature. For example, “Schilit” has categorized context into three categories: user context (For example: user’s location, user’s profile etc), physical context (For example: lightening, noise etc) and computing context (For example: network connectivity).

  Once we have context data available, next aim is to use this data for effective purpose. **Context aware computing** is a technique which process context data in mobile phones or computers for automatic device adaptations. There are two techniques for context aware computing: Active context awareness and Passive context awareness. **Active context awareness computing** takes discovered context data, and changes the device configuration automatically. Whereas, **Passive context awareness computing** preserves the discovered context data for future use, or it gives updated context values to interested users only. It does not take action immediately.

- **Challenges**
  
  Context awareness has been introduced in mobile computing very recently, but it has been introduced earlier in fieldwork and tourism, to provide latest location details. Although, there has been rigorous research in this domain, but still it has lots of challenges existing in developing stable context aware applications. Using insufficient techniques in context recognition leads to lots of ambiguities and erroneous conclusions of current context. Even identifying context attributes and properties is a difficult task. Many other concerns exist as well, first issue is, by enabling context aware features in a phone, it will take away the user control from end user, and it’s not clear that how many users will actually like it. Secondly, there is inadequate information about how many end users actually knows about end-user programming etc and will be able to configure it. That’s the reason; it’s a very challenging domain to research.
Earlier survey and work on context aware computation

Lots of papers have been published, and surveys have been done on context aware applications. One of the papers published by NOKIA on similar domain is [10]. In this paper they have mentioned how they model GUI for context aware mobile application, where user can configure user defined configurations, and settings of phone will change automatically, whenever there is change in the context. In this paper, they have combined sensors value to do configurations changes in mobile. Another important paper published in same domain is [11]. In this paper, the authors have done detailed research on sensors, how to combine different sensor values in multi sensor mechanism, and integration of sensing part with mobile applications. They have developed TEA module, which is a layered architecture for sensor-based computation of context, with separate layers for raw sensor data, for features extracted from individual sensors (cues), and for context derived from cues [11].

There is lots of context aware applications have been made in different domains. We would like to mention few of them: “Call Forwarding” by Olivetti Research Ltd. (ORL), “Mobisaic Web Browser” by Voelker and Bershad at University of Washington”, and “Shopping Assistant” by AT&T Bell Laboratories” [6]. These all mentioned applications uses active and passive context aware computing. Some similar context aware applications have been developed on Android platform earlier. One of these applications is “Locale” [12]. This Android based application also changes device configuration automatically, based upon series of conditions. Currently, it does not use any hardware sensors; it works mainly on internal sensors like Clock, Calendar etc.

2.2 Background on sensors and its integration with mobile phones

To start with the historical background on sensors and sensing environment, silicon sensors has been existed from decades now. Their main usage was monitoring pressure, temperature, sound, vibration, motion etc. The development and research on sensors and related sensor network was totally inspired by military applications, area monitoring, industrial monitoring, environment monitoring earlier, but later this technology was also introduced in telecommunication [2].

Martin Cooper, a Motorola researcher in 1973, invented first successful commercial mobile phone. The first few versions of mobile phone did not use any hardware or software sensors significantly. But later, in 1985, Motorola again launched first hardware sensor integrated with mobile phone i.e. “Microphone” to detect noise. The model name was “MicroTAC”. Then later in 1990’s J-phone of Japan introduced first camera phones in the market, using CCD sensors. In late 1999, 2nd generation (2G) mobile services were introduced and introduction of the 3rd generation (3G) high-speed data in 2001 became another turning point in the history of cell phones. Lots of network provided features were used as sensors, or to sense the environment and end user programming has been introduced on mobile phones to access network provider as sensors. In early phase of 21st century, many other sensors were also integrated with smart phones, like light sensor, temperature sensor, accelerometer etc [3] [4].

As we can see above, the mobile has been integrated with software and hardware sensors gradually over period of years, to enhance and automate functionalities.
2.3 Background on android

Our thesis research also includes investigation on Android platform, so we will also add some important information about background history and foundation of Android.

“Android is a software stack for mobile devices that includes an operating system, middleware and key applications”. [1]

This OS uses modified Linux kernel internally, and its internal implementation language is Java. To give a brief about background history on Android, a small company named “Android Inc” in California USA developed it in 2005. Later, “Google” has purchased this small company and started more research on Android to add more features and libraries. First Android version launched in market for use in 2008, it was very basic and not very enriched. But the latest stable version i.e. v2.2 has been launched on 20th May 2010, which supports almost all possible functionalities. The main founders for Android are Andy Rubin, Rich Miner and Nick Sears [5].

![Android Architecture](image)

**Figure 2:1 Android Architecture [1]**

Figure 2:1 shows Android’s internal 5 major internal components. The upper most component i.e. “Application” contains default and inbuilt applications, which comes along with Android platform. It includes browser, contacts window, calendar etc [1].

The second component i.e. “Application Framework” represents the open source android development platform. This component provides rich set of libraries and in built API for developers to develop applications. It also allows developers to access device hardware, location information and other functionalities. As shown in figure 2:1, this component gives public access on all managers to perform all types of activities, like sending notification, controlling activity, GUI etc [1].
The third component i.e. “Libraries” contains rich set of libraries developed in C/C++, and it is internally being used by other components of android. These libraries provide 2D and 3D graphics features, relational database engine, media features etc [1].

The forth component i.e. “Android Runtime” consists of two sub parts: Dalvik virtual machine (Dalvik VM) and Core libraries. Each android application, with unique instance runs individually on Dalvik virtual machine. Device supports multiple virtual machine execution at one time efficiently. The Dalvik VM converts the files into “.dex” format and then executes it. It uses Linux Kernel internally, for threading operations and memory management. Core libraries in Android Runtime, integrates Android platform with many important libraries of Java programming language and make it available for developers to use [1].

The fifth and final component i.e. “Linux Kernel” acts as separation layer between software and hardware layer in Android. Android uses 2.6 version of Linux, and it is totally dependent on Linux for core functionalities like memory management, process management, network stack etc [1].

2.4 Background on end user programming

End user programming can be defined as:

“End user programming provides the tools, techniques for those people who exploit computer automation for their purposes without becoming professional programmers.” [13]

This technique is used by those people, who are not expert in programming, but they still need to use this to serve their needs. The programming languages included under it are mainly scripting languages or visual languages like Perl, Java Script, Informatica etc. These kinds of languages are mostly very easy to operate and learn. Details on its categorization are explained under section 3.2.
3. Proposed Solution to Research Questions
In this chapter, we are going to discuss our proposed solution to the research questions, mentioned in section 1.4 above. As described in section 1.4, there are three possible sub parts of our thesis research, the brief solution to each sub parts are presented below:

3.1 Sensors

➢ What sensors will use to sense environment?

**Proposed Solution:** To answer the above question, we have investigated on all possible sensors, which can be used along with mobile phones and categorized them in specific category, based upon their nature. The diagram below represents the sensor decision tree:

![Sensor Decision Tree](image)

**Figure 3:1 Sensor Decision Tree**

Figure 3:1 shows sensors categorization based upon their nature. We have first categorized sensors into internal and external category. Internal sensors signify the sensors, which are present internally within the phone body, whereas External sensors signify the sensors, which are situated outside the phone body. Further, we have broken down each category (internal and external) into sub category of Hardware and Soft sensors. Hardware sensors represents hardware sensor units present within phone, like microphone, camera etc. Soft sensor represents those services or features of the phone, which we can use to sense environment, for example GPS, web services, network provider etc.
In the tree, we have again broken down Hardware and Soft sensor category into further categorization, i.e. Trusted and Non Trusted sensors. Trusted sensors are those sensors, which can sense environment all the time and give values, irrespective of change in environment, for example clock, camera etc. Non-Trust sensors are those sensors, which can be inactive during some conditions or in some environment and cant sense all the time. For example Network provider, this sensor will not sense environment if there is no network coverage in particular area.

The sensor decision tree above provides appropriate and sufficient sensors to sense environment/context; hence it provides solution to above research question. We have chosen this approach to sense context change using sensors, because they are less prone to produce wrong results. Even in earlier work done on same domain; researchers have used different type of sensors to sense context state. For example GPS services to sense location context, clock to sense time, etc.

- **How to sense real world environment change?**

  **Proposed Solution:** To sense real world environment change, we need to implement listeners for each sensor. The listeners will keep on checking sensor value change. Whenever there is some change in the sensor value, it will send notification and store latest sensor value, in case of environment change.

- **How do we model the environment?**

  **Proposed Solution:** To answer this question, first we need to think on what environment change we want to sense and which available sensors can be used to sense that particular environment. There could be possibility that there is no available sensor to sense a particular action, and we need to combine multiple sensor results to sense that action.

  For example: to sense motion, we have a sensor called accelerometer, but to sense relative distance of user with a particular monument, there is no available sensor to sense it. In this case, we need to combine few sensors results like GPS results, range sensor results etc.

  So, to model the environment, we have chosen to sense environment like time, motion, temperature and location only. And then used respective available sensors mentioned in figure 3:1, to sense these environments.

- **How to combine multiple sensor results?**

  **Proposed Solution:** We need to develop a sensor framework, which will store all sensors’ value in a map with key as sensor name and value as sensor value. So, whenever we need to combine different sensor’s result, we just need to get their latest values from map mentioned above, and combine them to identify any particular action. For example: If we need to identify what is user’s latest location, then we can take latest value for “GPS” sensor and also take latest location value from network provider sensor, then combine these two results to identify user’s current position.
3.2 End user programming

➢ What are the possible EUP types and their classification?

**Proposed Solution:** There are four possible end user programming categories; they are explained in detail below [13][14]:

a> Application Specific Programming: This category of end user programming is more application specific. The programmers who use this EUP technique, they are not expert programmers, but they do some little programming and use basic coding of this kind of EUP programming languages, to achieve their goal. Example: scripting languages like Perl. The main users of this category of EUP are network administrators and web page authors.

b> Visual Programming Language: A Visual programming language (VPL) is programming language which allows users to write his own programs by manipulating programming language elements graphically rather than by writing textual code for creating them explicitly. For Example: Informatica.

c> Natural Programming: This kind of EUP programming languages includes general keywords, simple methods, and simple programming technique. Hence very less amount of learning and effort is actually needed to write programs, even for non-professional programmers. For example: programming microworlds.

d> Programming by Example (PBE): This kind of EUP programming languages includes a technique for teaching the computer new behavior by demonstrating actions on concrete examples. The system records user actions and generalizes a program that can be used in new examples. Example: keyboard macros in an editor.

➢ What are the possible domain languages are present for EUP implementation?

**Proposed Solution:** There are many possible domain languages for EUP implementation. Some of them we have already mentioned in above section under EUP categories. For example, Java script, Perl etc. Since we are going to implement our thesis proposed solution on Android enables mobile phones, so we have to develop EUP implementation using Android SDK 2.1, and currently Android SDK supports core java as a domain language.

➢ How EUP will interact with sensors?

**Proposed Solution:** As mentioned above, we have to develop a sensor framework, which will collect data from sensors and maintain them in a map. This framework will also be responsible to collect latest sensor value, if any environment change occurs. If there is any environment change, then the sensor framework will send notification or event to EUP about this change. EUP will have an event receiver to receive such notifications. As soon as EUP will receive any notification, it will trigger EUP engine and perform the appropriate actions. The same approach is presented diagrammatically in figure 3:2 below:
So, by sending and receiving notifications/events, both EUP and sensor layer can interact with each other. More details on its implementation are explained under section 4.3 and 4.4.

3.3 Adaptations

➢ How may End user programming influence device adaptation?

**Proposed Solution:** To influence device adaptations, we need to develop an End user programming framework, which will be installed in the device. This framework will work on event handling mechanism, so that whenever sensor framework sends any notification, it should be able to receive it and do required processing. Our proposed solution on end user programming framework is to develop an ECA (Event Condition and Action) rule engine, which will receive notifications, check if any condition is matching with the received event on device, if yes then this rule engine will perform specified action on device.

So, in above manner, EUP can influence device adaptations on real world environment. We are discussing the design and its implementation more in detail in chapter 4.
4. Design and Implementation
In this chapter, we are going to discuss details on designing and implementing our proposed solution mentioned in chapter 3.

To solve any complex problem even in real life, there is a standard technique, which every one follows, i.e. Divide and Conquer. We need to divide the problem into smaller sub parts and then analyze each of them separately. Once we have analyzed the problem in each sub part and found solution for each sub part, then we have to find common glue or interface, which will combine all solutions on same context to form overall solution to the problem. We have also followed same steps to solve our thesis research problem, Segregation, Analysis and Combination. We have segregated our problem in three parts: sensors, adaptation, and End user programming, and then investigated about each of them.

4.1 Software development model
There are many possible software development models, which can be used to develop software projects. The most popular models are: Waterfall model, V model, Prototyping model, Transformational model, and Spiral model etc [15]. There are some standard steps, which every software developer needs to follow while developing software projects. Those major steps of SDLC (Software Development Life Cycle) are:

- Requirement Analysis
- Design and Modeling
- Implementation
- Components Integration
- Testing

We are using the above steps to implement our proposed solution, and we are using Waterfall Model in developing our thesis project.

4.2 Requirement analysis
As mentioned above, our thesis is mainly focused on showing how device gets adapted to change in real world environment. So, the main functionality we need to implement is, if any sensor value changes, device should be able to adapt to that change. Given the short time frame, we cannot implement all the sensors, but few essential requirements are explained below:

4.2.1 Functional requirement
- If any sensor is changing by itself, then device should be able to adapt this change automatically, without user’s intervention. For example: Clock is a sensor, and it changes every second by itself irrespective of environment change. So, in this case device should be able to adapt to this change in clock sensor value.
- If end user explicitly modifies any sensor value, then device should be able to adapt to this change. For example: If end user changes the date and time in date settings of the phone. So, in this case device should be able to adapt to this change in clock sensor value.
- As soon as device is started, default configurations should be loaded and sensors to sense real time environment should be active, without user’s interaction.
- User should be allowed to make his own phone adaptation configuration, according to his need. Then users need not to explicitly make phone configuration change when that particular configuration condition satisfies. For example: If user wants to beep an alarm, when he is in university and current time is 2 pm. So, we need to provide a
GUI, where user can enter his adaptation configuration (beep alarm), along with satisfying conditions and actions to be taken.

4.3 System design
To design the whole thesis implementation, we have developed two different frameworks “Sensor framework” and “End user programming framework” using Android API v2.1. Here is the overall high-level architecture of thesis implementation:

![High Level Architecture of Thesis](image)

Figure 4:1 High Level Architecture of Thesis

Figure 4:1 shows 3-tier structure of our implementation. Top layer represents “Sensors framework” which will sense environment changes and sends notifications or events if there is any change encountered. Middle layer i.e. “BroadcastReceiver” acts as interface or connector between EUP framework and Sensor framework. Its main job is to receive notifications from sensor framework and trigger EUP rule engine, so that EUP will start changing device configurations accordingly. The lowest layer in our architecture is “EUP framework”, which is a kind of rule engine, it maintain default rules as well as user defined configuration rules. This layer is responsible to take latest sensor values from sensor framework, and iterate through whole list of rules to check, if current sensor values are satisfying any condition. If yes, it will perform registered action against that satisfied condition.
4.3.1 Low level design for sensor framework

Figure 4:2 shows class diagram of sensors framework. “CollectSensorData.java” is the main class of Sensor framework, which internally uses “Sensor.java” interface to collect all sensors data. It even registers and deregisters listeners for all sensors. The class “CollectSensorData.java” maintains a static map to store all sensors data as key value pair. More implementation details are explained in section 4.4.

4.3.2 Low level design for EUP framework

We will discuss the ECA framework that we have used for developing the EUP in detail under this section.

Event Condition Action (ECA) framework is an event handling framework which works on three fundamental concepts i.e. event, condition and action. The basic functionality of ECA languages follows below logic,

“On Event
If Condition
Then Action”
This means whenever any Event occurs, then Condition will be verified, and then it executes the Action only when condition returns true. ECA systems get input in the form of events from the external environment and execute actions that change the internal stored information or change the environment itself [16] [17]. The below diagram depicts a scenario of how an end user context is collaborated with our ECA rule framework:

Figure 4:3 User Context Collaborated with ECA Framework

Figure 4:3 shows, an end user creates user-defined configurations using GUI. GUI has to make sure that user enters all required data needed to form a new ECA rule. The event here will be any context change, which could be change in time or location. In this scenario, Condition to be satisfied will be “Time value should be 10 AM and location should be University”. And Action to be taken will be “putting mobile phone on silent mode”. So, whenever there will be change in external environment (time/location), an event will be send to ECA framework, then ECA checks if any condition is getting satisfied. If yes, then it will execute the specific action. Low-level class diagram for EUP framework is explained below:
Figure 4:4 shows class diagram for EUP framework. “GUIEUP.java” is the main class in EUP framework, which does most of the processing. The major functionalities it supports are, collecting data from sensor framework, receiving notifications from sensor framework, loading user defined rules GUI, checking conditions, changing android enabled device configurations etc. More details on their implementation are included under section 4.4.

4.4 Implementation
In this section, we will discuss class level implementation for overall thesis implementation. As mentioned earlier, thesis implementation has two frameworks: Sensor framework and EUP framework, and then there is common interface “BroadcastReceiver” to connect their processing. We follow the below basic algorithm, for almost all scenarios and use cases:

Step 1: Sensor framework accesses all sensors and makes them active at device load time.

Step 2: Sensor framework registers and implement listener classes for each sensor, so that if any change occurs in their values, it should send notification to “BroadcastReceiver” interface.

Step 3: Sensor framework also stores latest sensor values in a static map, so that all sensor listener will have access to this map, and they can update their latest values, whenever there is some change.

Step 4: As soon as “BroadcastReceiver” interface receives notification messages, it invokes the EUP framework.
Step 5: Sensor framework maintains a list of sensor names, whose values has been changed, due to environment change. So, EUP framework will check this list, and then takes latest value for only those sensors which are in the list.

Step 6: EUP framework is a rule based engine, which has configuration rules maintained in a XML file. EUP framework iterates thorough all rules to check, if any rule becomes active after latest sensor value change. If yes, then it performs the relative action assigned on that rule. Otherwise, it does nothing.

To implement the thesis solution, first work item is to install Android emulator as a plug in with eclipse. For developing solution on Android enables platform, we first installed Android v2.1 SDK and added it as plug in with eclipse to develop project.

Next step is to implement Sensors and EUP framework. Here are their implementation details:

4.4.1 Implementation details on sensor framework
Here we will explain only main classes of sensor framework. “CollectSensorsData.java” is the main class of Sensor framework, which connects other classes in this framework. If any module needs to have sensors data, then they need to make object of this class and call its functions. Here is the basic structure of class:

```java
public class CollectSensorsData {
    public static Map<Object,Map<Object, Object>> outputMap = new HashMap<Object,Map<Object, Object>>();
    public static List<Object> changeSensorsList = new ArrayList<Object>();
    public void collectSensorsData(SensorManager manager, Activity activity) {
        Sensors timeSensors = new TimeSensors();
        timeSensors.getSensorsData(manager, activity);
        System.out.println("All time sensors are registered and available.");

        Sensors tempSensors = new Temperature();
        tempSensors.getSensorsData(manager, activity);
        System.out.println("All temperature sensors are registered and available.");

    }
    public void registerAllListeners(SensorManager manager, Activity activity) {
        manager.registerListener(new LightSensor().lightSensorListener,
        manager.getDefaultSensor(Sensor.TYPE_LIGHT),
        SENSOR_DELAY_GAME);
    }
}
```
As shown in above code, this class has three main functions: collectSensorsData () and registerAllListeners () and unregisterAllListeners (). collectSensorsData () is the function, whose main work is to access all sensors and make them active. This function calls all sensors interface and access all hard and soft sensors and make them active at device load time. registerAllListeners () registers listener classes for each sensor to sense value changes, and send notification. unregisterAllListeners () deregisters listener classes for all sensors, and make the inactive. The class also contains static map and static list, static map contains latest and updated sensor values with key as sensor name, and value as sensor value. The list contains names of those sensors, whose values have been changed due to change in environment. Both these global variables are static, because every sensor can have access to these variables and they can update sensor data into these global variables.

The structure of classes of all implemented sensors is pretty much same, except for few exceptions like “Clock”. For all other sensors, listener sends notification, whenever there is change in sensor value, but since clock changes every second, so we have handled it in such a way that, only if hour value changes then only it will send notification, otherwise it does nothing. Here is one sample class which represents basic structure of sensors implementation:

```java
public class Accelerometer implements HardwareSensors{

    public void getHardwareSensorsData(SensorManager manager, Activity activity) {
        Sensor sensor = manager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

        if(sensor != null) {
            float[] sensorVal = new float[1];
            sensorVal[0] = sensor.getMaximumRange();
            updateSensorsValues(sensorVal);
        }
    }
}
```
private void updateSensorsValues(float[] sensorValues) {
    Map<Object, Object> sensorDataMap = CollectSensorsData.outputMap.get(SensorsEnum.MotionSensor);

    if(sensorDataMap != null) {
        if(sensorDataMap.containsKey(SensorsEnum.Accelerometer)) {
            sensorDataMap.remove(SensorsEnum.Accelerometer);
            sensorDataMap.put(SensorsEnum.Accelerometer, sensorValues);
        } else {
            sensorDataMap.put(SensorsEnum.Accelerometer, sensorValues);
        }
    }
}

public final SensorEventListener accelerometerListener = new SensorEventListener() {

    public void onSensorChanged(SensorEvent event) {
        CollectSensorsData.changeSensorsList.add(SensorsEnum.Accelerometer);
        float[] values = event.values;
        updateSensorsValues(values);
        sendNotification();
    }

    public void onAccuracyChanged(Sensor sensor, int accuracy) {};
};

As shown in above code, to access and activate the sensor, we have used “SensorManager” class of Android API. With the help of this class, we can access sensor data from device and update default sensor value when device gets loaded. We have also made a listener object for this sensor using “SensorEventListener” class of Android API. This class helps in tracking changes in sensor values, so inside the function “onSensorChanged ()”, we update static map with latest value of sensor and send notification.

4.4.2 Implementation details on EUP framework
As we mentioned earlier, our implemented EUP framework is based upon ECA framework. It’s a rule-based engine, which is totally driven by stored rules in XML file.

- Possible ECA Framework Available
  Identifying a domain (Android platform) specific framework for the ECA plays a vital role in this End User Programming development. We have done a research on identifying frameworks to implement the ECA and came up with various techniques that are available in the present market. For example: DROOLS, JESS and
JRuleEngine. They vitally support the development of ECA framework, in which DROOLS and JRuleEngine are free and legally registered to use than JESS. They all are Java based ECA framework languages [18].

Even though there are existing frameworks available, but we could not use them, because it’s impossible to implement these available frameworks in our Android platform structure, due to various technical reasons. One of the major reasons we found that, class file generation in Android is different from class file generation in DROOLS and JRuleEngine.

So we resolved this problem by implementing our own rule based engine framework based on the reference and ideas of the JRuleEngine framework with respect to Android mobile platform.

➢ Rule Engine
“A rule engine may be viewed as a sophisticated if/then statement interpreter”. [19]
The if/then statements are called rules, which act as input to the rule engine for its execution. The “if” signifies conditions such as “at 10 A.M. in Lecture hall”. The then signifies set of actions to be taken such as Silent, Vibrate, Connect Bluetooth etc. Rule engine takes rules as input and process if/then statements accordingly. In most of the cases, rules are maintained in XML format. The outputs from a rule engine are governed totally by the inputs [19].

➢ Rule Definition
“The if/then statements that are interpreted are called rules.” [19] The rules can be loaded from a XML file that is stored in the Android data card. A sample template of rule is defined as:

A name
A description
A list of Condition objects, having the format: "condition1, condition2 .....”
All condition are handled by an AND operator.
A list of Action objects, having the format: "Action1, Action2 .....”

➢ DTD (Document Type Definition) for a Rule Execution Set XML file
Rules can be defined in an XML file. This file must respect the following DTD:

```xml
<!ELEMENT rule-execution-set (name, description, synonymn*, rule*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT synonymn>
<!ATTLIST synonymn name CDATA #REQUIRED>
<!ATTLIST synonymn class CDATA #REQUIRED>
<!ELEMENT rule (condition*, action*)>
<!ELEMENT condition >
<!ELEMENT condition1 >
<!VALUE condition1 CDATA #IMPLIED>
<!ELEMENT condition1 >
<!VALUE condition1 CDATA #IMPLIED>
...
<!ELEMENT conditionN CDATA #IMPLIED>
<!ELEMENT action >
```
Categories of Rules in EUP framework
There are two categories of rules, we have specified for the EUP framework:

- **Default Rules**
  Default rules are those set of rules, which are already present in the XML and will be invoked every time the mobile phone is started. When the mobile phone is started, EUP module invokes the rule engine to check for the valid conditions, using Boolean AND operator. If the conditions in the XML rule are satisfied then the corresponding action will be executed to update or change the phone settings.

- **User Defined Rules**
  End user can also define his/her own rules using GUI, according to his own needs. These rules will behave in same way, as default rules. It will be also stored in the same rule XML file. The GUI for user defined rules is shown below:
Class level details of EUP framework

“GUIEUP.java” is the main class of EUP framework, which has functions to parse the rule XML file and checking the conditions for each rule. This class also has functions to perform actions, which changes phone configurations and settings. Moreover, this class has GUI implementation code separated by various functions.

4.5 Integration
In this work, there were two major parts: developing Sensors framework and developing EUP framework. To integrate these two parts, we have developed “BroadCastReceiver” class, which integrates the mentioned two parts of our thesis project. This class receives signals send by Sensors framework and triggers EUP framework.

4.6 Testing
At present Android emulator, does not provide all real time sensors data, so we have used “Open Intent Sensor Simulator” to simulate sensor data [20]. The limitation with sensor simulator is that, currently it works only in integration with Android v1.5 emulator. So, we used v1.5 to test few test cases, and v2.1 for overall development, with slight changes. To test our implementation, we have used clock sensor and accelerometer variant sensor values to see, if emulator settings are getting changed with change in sensor values.
5. Evaluation
In this chapter, we will evaluate our thesis implementation according to use cases scenarios and also compare our implementation results with existing surveys and work done in this domain mentioned in chapter 2.

5.1 Evaluating current implementation with respect to earlier surveys
As mentioned in chapter 2, some earlier implementations have been done in same domain of context aware mobile applications. In paper [10] [11], researchers have used same pattern of overall structure as our implementation. They have combine sensors results to sense the context change and implemented end user programming to change mobile configurations.

But there are few differences on low level architecture and platform used in their implementation. Researchers have used Nokia specific Operating system, whereas we have used Android operating system. The implementation domain language to implement sensor and EUP frameworks is J2ME, whereas we have used Android API and core java. Moreover, our EUP implementation is also different from earlier implementations. We are maintaining rules in XML format and store that XML file in phone SD card, whereas in paper, they have used “Context Exchange Protocol (CEP)” technology whose flow of events is different from this thesis EUP implementation. Even GUI for earlier implementations is more complex to use, as compared to our thesis end user programming GUI.

Android context aware application “Locale”, currently it works on Battery, Calendar, Contact, Location, and Time only [12]. Our implementation uses more hardware and soft sensors like light sensor, temperature sensor, GPS, Bluetooth etc. The range of functionality provided by our implementation is much wider in scope than Locale.

5.2 Goal criteria testing
The main objective of this thesis is to present implementation result on Android based mobile phone, i.e. if real world environment changes then phone configuration can also change automatically. So, to test the implementation, we will evaluate it on below use cases. The main goal criterion is to meet requirements of below mentioned use cases.

5.3 Actors
➢ End User
➢ Mobile Phone Device
➢ Sensors

5.4 Use case scenarios
#Use Case 1 (Mobile Phone loaded first time)

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Mobile Phone loaded first time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief Description</td>
<td>According to this use case, whenever phone is getting loaded first time, all sensors should be activated to sense environment change and default values for each sensor should be loaded, in the device.</td>
</tr>
</tbody>
</table>
| Flow Of Events | Steps:  
1> Mobile phone is switched on by end user.  
2> All implemented sensors and their listeners, should get active, and loaded in mobile phone.  
3> Default values for each active |
sensor should also be loaded in mobile phone.
4> EUP framework should take default values of all active sensors and iterate through default rules stored in mobile phone device.
5> EUP will check if any condition is getting satisfied with sensor values.
6> If not, then there will no change in phone settings.
7> If yes, then registered action against satisfied condition will be performed by EUP framework on mobile phone, and phone settings will be changed accordingly.

<table>
<thead>
<tr>
<th>Actors</th>
<th>End User and Mobile Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre Conditions</td>
<td>Mobile phone must be switched on by end user.</td>
</tr>
<tr>
<td>Post Conditions</td>
<td>None</td>
</tr>
<tr>
<td>Relationships</td>
<td>N/A</td>
</tr>
<tr>
<td>Extension Point</td>
<td>None</td>
</tr>
</tbody>
</table>

Our implementation runs perfect for above use case, here is the snap shot from android emulator, which shows all implemented sensors are available and active:
# Use Case 2 (Mobile Phone is already loaded)

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Mobile Phone is already loaded</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brief Description</strong></td>
<td>According to this use case, mobile phone is already loaded and active. If any active sensor value has changed by user’s intervention, or if there is automatic change in environment, then device should adopt to these changes.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flow Of Events</th>
<th>Steps:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1&gt; Sensor listeners will keep checking if value of sensor changes.</td>
</tr>
<tr>
<td></td>
<td>2&gt; If there is any change in sensor value, it will send notification to EUP framework about this change.</td>
</tr>
<tr>
<td></td>
<td>3&gt; If EUP receives any notification from sensors, it will trigger rule engine to take latest value of sensors from sensor framework.</td>
</tr>
<tr>
<td></td>
<td>4&gt; EUP framework will iterate through all stored rules to check, if current sensor values satisfies any condition.</td>
</tr>
<tr>
<td></td>
<td>5&gt; If not, then there will no change in phone settings.</td>
</tr>
<tr>
<td></td>
<td>6&gt; If yes, then registered action against satisfied condition will be performed by EUP framework on mobile phone, and phone settings will be changed accordingly.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Actors</th>
<th>End User, Sensors, Mobile Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pre Conditions</strong></td>
<td>Mobile phone must be switched on and active.</td>
</tr>
<tr>
<td><strong>Post Conditions</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Relationships</strong></td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Extension Point</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

Here, we are using clock and accelerometer sensor data, to show if clock or accelerometer values are changing, and any rule becomes active, then it will change the mobile phone configuration automatically. In our rule file, there is a default rule that, if accelerometer value is between 0 -5, and clock value is 12 to 17, and then change the volume of phone to silent. Here are the snap shots:
Initial state of android emulator, when loaded first time, it is on general mode:

Now we will try to change the time in clock setting to 12.
Now, we checked the sound settings again in emulator, it is on silent automatically.
# Use Case 3 (End user creates new user defined device adaption configuration)

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>End user creates new user defined device adaption configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief Description</td>
<td>According to this use case, end user should create his own user defined device adaptation configuration on mobile phone, for his personal needs.</td>
</tr>
</tbody>
</table>
| Flow Of Events | Steps:  
1> End user clicks on “Menu” button on mobile phone.  
2> Welcome screen of our thesis implementation will appear on mobile phone screen. It will ask user if he wants to create user defined configuration.  
3> End user clicks on “yes” button.  
4> A new configuration form will appear on mobile screen.  
5> End user enters rule name, sensors data and actions, and then clicks on “Save” button.  
6> This new configuration rule will be saved in mobile phone and it will trigger EUP rule engine.  
7> EUP framework will iterate through all stored rules to check, if current sensor values satisfies any condition.  
8> If not, then there will no change in phone settings.  
9> If yes, then registered action against satisfied condition will be performed by EUP framework on mobile phone, and phone settings will be changed accordingly. |
| Actors | End User, Sensors, Mobile Phone |
| Pre Conditions | Device phone must be switched on and active. |
| Post Conditions | None |
| Relationships | N/A |
| Extension Point | None |

For evaluating our implementation against this use case, we have created a new user defined rule using GUI. Here are the snapshots:
On click of “save” button, it saves this rule in XML rule file. Since this is saved successfully on SD card of emulator, it will active as soon as both above condition will be
satisfied. We repeated the same steps as use case #2, to see if phone configuration changes. Here is the snap shot of rule XML file, where above user defined rule got saved.

```xml
<rule name="Volume low" description="" >
    <condition>
        <condition1>
            <sensorname>Clock</sensorname>
            <sensorvalue>10-12</sensorvalue>
        </condition1>
        <condition2>
            <sensorname>Accelerometer</sensorname>
            <sensorvalue>20-25</sensorvalue>
        </condition2>
    </condition>
    <action>
        <action1>Silent</action1>
    </action>
</rule>
```
6. Conclusion and Future Work
In this chapter we will discuss, what we have implemented at present, and what could be future extension to this thesis project.

6.1 Conclusion
The results presented under chapter 5, showed us possible use of context as driver and whenever environment changes, device can adapt to that change. As of now, we have presented one successful solution for research problems. We can achieve that by many other possible solutions as well. We have faced lots of challenges while modeling and implementing this thesis, especially on Android platform. Since this is very new platform to work with, android emulator does not support sensors live data and even its class generation technique is also different. So, we had to put lots of efforts in simulating sensor change and doing other implementations. Overall, this thesis was success, and we are able to present expected results.

6.2 Future Work
There can be many possible extension works to this thesis project, few of them are mentioned below:

- At present, we have not implemented many sensors like microphones to detect noise, camera to detect landmarks etc. These can be implemented to enhance the functionality of the mobile phone.
- On End user programming part, we have not handled the rule conflicts scenario. For example, if two rules becomes active at the same time and their actions are conflicting then how EUP should behave in this scenario. This could be very interesting extension work to the thesis.
- We can have more generic, robust and better extensive framework for EUP.
- Currently, in EUP framework, while evaluating rules, we have used only AND Boolean operator to evaluate conditions. Another possible extension to this thesis work could be to implement rule engine, which will evaluate even OR Boolean operator. For Example: If User current location is home OR there is no noise detected, then activate the phone volume into meeting mode.
7. References


