Controlling a Wireless Sensor Network using a Android Smartphone

Umme Hafsa
M.Sc Students
Dept. of Information and Communication Technology
Mawlana Bhashani Science and Technology University

Mohammad Badrul Alam Miah
Associate Professor
Dept. of Information and Communication Technology
Mawlana Bhashani Science and Technology University

ABSTRACT
The use of smartphone devices over the past years seems to follow a growing trend. This great acceptance along with the endless possibilities that go hand to hand with having a mini computer at all times within reach, can explain this vast interest shown by solo developers and major companies in the mobile industry. As a result, many innovative applications roll out daily to the various online stores, making the lives of the smartphone users a lot better. This thesis describes the design and implementation of a mobile app, a Web Service and a TinyOS application, that bind together allowing the user to execute a variety of queries on a sensor network from any place in the world.

Keywords
Smartphone, Wireless Sensor Network, Sensor Network Control Smartphone, Android Phone Control of Sensor Network, iOS Phone Control of Sensor Network;

1. INTRODUCTION
The role of applications combined with the flexibility they offer, are the major factors behind the popularity of smartphone usage. The time spent on applications compared to the time spent on websites has grown from 73% (2011) to 81% (present). The number of new subscriptions to Android and iOS systems, which at the moment lead the smartphone market share, in the first half of this year had already crossed 84 million compared to the total number of 2011 year subscriptions which was 38 million. The average number of applications per smartphone has increased from 32% to 41%. Moreover, the percentage of app downloads in Android and iOS operating system phones has grown from 74% to 88%. All these statistics indicate that mobile industry is without question on a raise.

These figures also show that mobile development offers opportunities for profit not only to software companies but to solo developers as well. Innovative software can easily be built and help users in their personal and professional lives. In addition, due to the fact that more and more developers get involved into mobile development, it is not unusual for companies and simple users to hire experienced solo developers to build customized applications that satisfy their needs completely.

Android is, at the moment, one of the most popular mobile platforms, with hundreds of millions of mobile devices in more than 190 countries around the world, with daily activations surged from a million Android devices back in June of 2012 to today's number of 1.3 million.

Additionally, Android is a Linux-based operating system designed primarily for touchscreen mobile devices. It is developed by Google in conjunction with the Open Handset Alliance, which is a consortium of 86 hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices.

1.1 Sensor Network and TinyOS
A wireless sensor network (WSN) consists of spatially distributed autonomous sensors that monitor physical or environmental conditions, such as temperature, sound, pressure and so forth while they cooperatively pass their data wirelessly through the network to the base station. The development of wireless sensor networks was motivated by military applications such as battlefield surveillance. Today such networks are used in many industrial and consumer applications, such as in the industrial process of monitoring and control, machine health monitoring, and so forth.

The sensor nodes used in this application use TinyOS as their operating system. TinyOS is a free open source, BSD-licensed, component-based operating system, designed for low-power wireless devices. TinyOS is written in nesC(network embedded systems C) programming language. There is a worldwide community from academia and industry that uses, develops and supports TinyOS and its associated tools. This operating system is less common in the embedded world of sensing and control. In the area of the embedded systems, applications are usually bound to a specific hardware. This is preferred due to the very limited hardware resources and the degree of specialization of the applications.

1.2 Web Services
The Android device had to be able to connect to the sensor network, without any proximity limitations to the area where the sensor network was installed, a web service had to be used. The primary concept of web services is that a client sends a request over HTTP (or any similar protocol) to an address in which the web service is hosted and "listens to". When the web service receives it, it sends a response back to the client with the requested data. The developer of the client does not have to be aware of what is happening in the server side, he only needs to make sure that the messages that are sent and received follow the rules that the provider of the service has set. On the other hand, the business logic that runs on the server side is completely up to the developer of the Web Service as long as the data returned to the client, again, follow the rules he set.

1.3 Thesis Contribution
The main object of this thesis was to find a way to assemble these three parts, the sensor network, the android application and the web service, in such manner that will let users use a sensor network without any prior knowledge to any of these

fields. In order to achieve that goal, it was important to make sure that final user would not have to worry about technical issues since that user could be virtually anyone. Any user that just wants to keep track of the temperature, light or humidity levels of an area, which could be his house or his office, had to be able to do so without any technical concerns. Thus, the interface of the application had to successfully inform the user, providing him with the data he needs in a clean and minimalistic design, without demanding too much effort on his part. Of course since the target user of this system can be anyone with or without any prior technological knowledge we had to make sure that precautions had to be taken in order to avoid destructive use of the system.

It should be mentioned that this application works in the same way no matter what it measures. The fact that TinyOS is a component-based OS provides the option to use components that especially measure what the end user wants without any changes to the main code of the application with just a minor change on the wiring. That way, it is possible to satisfy the needs of a broader audience and use the system in a variety of situations since it is possible to receive any kind of measurement and not be limited to a specific one.

In order for someone to use this system a number of sensors running TinyOS have to be acquired. Right after the installation of the provided application on each and every one of them, they should be placed into the area on which they will operate. The base station of this network must be connected to a computer system with internet access. After acquiring a unique internet address and having initiated the service on the computer system, it is ready to accept incoming messages from the client. The client in this case, is the Android App. When the user creates an account with his personal details, through the application he can log in and start using the system.

2. DEVELOPING A TINYOS APPLICATION

TinyOS programmers everything about this system with a step-by-step procedure. Additionally, ”TinyOS Programming” [1] is a great book that will certainly be useful not only to a beginner in TinyOS but to more experienced programmers as well.

Figure 2.1: Iris mote.

What is more Yeti [2] is a plugin for eclipse IDE that can be very useful while development tinyOS applications. Although it is not currently under development, it is a tool that undoubtedly will save a lot of time to any programmer. Additionally, a syntax highlighter for nesc is available for the famous editors gEdit and Kate.

2.1 Simulating TinyOS Networks

TinyOS applications are expected to run on motes with very limited resources in extremely uncontrollable physical environments. If that was not enough the embedded nature of those sensors makes controllable experiments difficult, therefore reproducing a bug is virtually impossible. As a result debugging is a really difficult procedure.

What adds to that statement, is the fact that no mechanism can be used to control the execution of the program. No breakpoints can be used to check if the program runs correctly and printing out messages can be truly problematic as the buffer is very limited and messages are frequently lost. Not to mention that in order for these messages to be printed on the screen of a conventional computer, every mote has to be connected with it. However connecting the motes an entire sensor network to a PC can be extremely inconvenient.

TOSSIM is a very useful mote simulator that can be used to easily develop sensor network applications. This simulator scales to thousands of nodes and compiles directly from the source code. TOSSIM simulates the TinyOS network stack at the bit level, this means that the programmer can use this simulator to experiment not just with top level applications but with low level protocols as well.

2.2 Power Consumption

Since these motes are supposed to run independently in remote areas for a long time they should be energy efficient. Limiting power consumption has been the main purpose of many publications in the field of Wireless Sensor Networks this definitely shows the importance of energy efficiency on such networks. Certain mechanisms were used to limit the power consumption in this application as well.

Firstly, it should be mentioned that there are two ways to accomplish energy efficient algorithms. The first one is by limiting the information transmitted by individual nodes and the second one is by increasing the amount of time the nodes remain inactive.

Both of these ways were used to make our application more energy efficient. Two algorithms were used to accomplish this, TAG and TiNA.

2.3 TAG (Tiny AGgregation Service for Ad-Hoc Sensor Networks)

TAG [3] (Tiny AGgregation Service for Ad-Hoc Sensor Networks) among others, states that the processing and computation of aggregate queries can be performed within the network to limit the transmitted amount of data. In TAG the base-station generates an aggregate query that the user specifies and it is being transmitted towards the sensor motes within the network.

Messages are transmitted from the base station to the nearest nodes and these messages are forwarded to their neighborhood nodes, thus creating a tree. At the end of this distribution phase, a tree is being created and the base station is positioned at the root of this tree while each node belongs to a certain depth and has a unique ID. Additionally, each node is aware of the parent node to whom it will be periodically sending the outcome of the aggregate query computed by the measurement it receives along with the measurement received from the sub-tree formed under it.

2.4 TiNA (A Scheme for Temporal Coherency-Aware in- Network Aggregation)

TiNA [4] was also used to limit the power consumption. The main idea of this publication is that two sequential

measurements retrieved by a sensor usually are not that different from one another. As a result sending sequential values that are very similar does not significantly change the result of the aggregate query of the sensor network. However it does result in higher power consumption since identical values are sent periodically. Therefore, a tolerance clause can be used to avoid sending the same value over and over again. If the newer value is different enough from the last one it is being transmitted, otherwise it is not.

This mechanism lowers the number of messages that are being transmitted within a network since new measurements that do not provide any useful information are withheld. Since sensor network applications are intended to run on different environments, some measurements that are considered useful in one occasion may not be so useful in another. Therefore, the user is able to select which measurements are in fact useful and which are not by specifying the tolerance clause at the beginning of the application.

2.5 Description of Sensor Measurement TinyOS Application

This thesis implementation contains two TinyOS applications. The first one is used to collect the aggregate values of the sensor network. The main object of this application is to retrieve the measurements of the network and use the serial communication mentioned earlier to transmit these measurements to a server. That way the measurements can be stored into a database and can be later retrieved from the mobile application.

2.6 Routing Phase

At first this application begins with the routing phase. During this phase a message is broadcasted by the base-station and is being forwarded from node to node until an entire tree is formed. This message contains information about the aggregate query that is going to be executed and it includes data about the current depth and the parent node.

2.7 Synchronization Phase

As mentioned above each node turns on and off the radio on specific time slots to receive and send packets. In order for this to happen every node on this network has to be synchronized. A special component is used for this purpose that makes use of a global clock. After syncing every node can estimate the exact moment to turn the radio on and off.

2.8 Collection Phase

After the motes have been synced they are ready to collect measurements and calculate the result of the aggregate query. The aggregate query demands the calculation of the summary, average, maximum, minimum or count value of the tree. This process is performed periodically until the user selects to stop the measurement gathering. The result of the aggregate query is sent to the base station and is being forwarded through the serial port. On the receiving end of this serial port there is a web service listening for new packets. When a new packet is received it is un-serialized and stored into the database.

2.9 Ending Phase

If the user selects to stop the execution of a query the motes have to be informed. However since the motes turn on and off the radio if such decision is made by the user and a cancellation query is sent there is no guarantee that it will reach every node. In order to be certain that no problems will occur while stopping the query a special window is opened periodically. During this window every node on the tree turns on the radio. If the user desires to stop the execution, a special message will be transmitted through the nodes of the tree. After receiving such a packet every mote re-initializes its variables, the timers are stopped and the radio is turned on. That way when the user decides to execute another aggregate query every mote will be in his disposal again.

2.10 Outlier Detection

The second application is about outlier detection. This is the process of detecting abnormal node measurements within a sensor network. This information is useful since it may lead to interesting findings. Abnormal measurements may be due to malfunctioning motes or due to other physical phenomena, such as fires. This application was created by Mr. Antonis Igglezakis as a part of his thesis implementation and was included in this project. The implementation was based on a recent paper from Mr. Sabbas Burdakis about an algorithm for detecting outliers in sensor networks, based on a geometric approach [5][6].

3. IMPLEMENTATION

The main objective of this thesis implementation is to be able to control the described sensor network via a mobile device from any place in the world. In order to accomplish that an intermediate layer had to be used since the sensor network cannot directly connect with the mobile device. The reason why this step was necessary is because no web service is bundled with the TinyOS architecture, so we had to build our own.

This intermediate level has to accept connections from the mobile device through the internet and interact with the sensor network to manipulate it. Apart from that this level has to be able to receive the messages from the sensor network through the serial port, store them in a database and send those data to the client as well.

3.1 Right Architecture & Framework

SOAP (Simple Object Access Protocol) and REST (REpresentational State Transfer) are the two approaches available in this field. Each of these two architectures has a series of available frameworks and tools that can be used. Every single one of these architectures and respectively their frameworks and tools has their own advantages and disadvantages so the best option depends on the occasion in which they will be used.

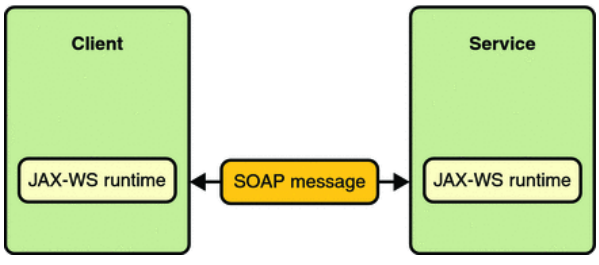


Figure 3.1: JAX-WS communication between the server & the client.

REST approach is praised for its ease of use and the fact that it is extremely lightweight. It has recently been used by many well-known companies and organizations. This approach is very easy to understand and it can be used by virtually any client and server that has HTTP/HTTPS support. SOAP on the other hand has been used extensively for the past few years and it will continue to do so. The majority of the initial issues have been fixed since it has been around for quite a while.

SOAP has some additional overhead compared to REST, but it also comes with some advantages over it. SOAP is more ”generic” than REST which means that while REST architecture can only be used on HTTP/HTTPS protocols, SOAP can additionally be used over SMTP (Simple Mail Transfer Protocol), JMS (Java Messaging Service) and more. On the other hand SOAP relies on XML to represent the transmitted information which is more verbose compared to REST. After weighting each option we decided to go with SOAP Architecture as we noted in the section: ”REST vs SOAP”

3.2 JAX-WS

JAX-WS stands for Java API for XML Web Services. JAX-WS is a framework used to build web services and clients that communicate using XML. Although SOAP message transaction can be very complex, this API hides this complexity from the developer.

3.3 Web Service - TinyOS Interaction

After describing the way the client-server interaction was implemented, it is time to describe how the sensor network interacts with the web service. As mentioned earlier TinyOS provides a tool that allows us to generate Java Classes and Python or C files describing the messages transmitted from the sensor motes to the computer and vise versa. This tool is named MIG (Message Interface Generator). Every message is in reality a sequence of bytes. What this tool does is parsing and un-parsing this sequence to a packet with usable fields.

Finally, after implementing the above-mentioned system the communication between the computer and the base-station was completed.

3.4 Database Design

In this section we will analyze the entities of the database system and the attributes each of these entities possesses.

User

The user is the actual operator of the system. No functionality can be performed if no user account has been created. The required attributes are the following:

- **Username.** It is used as the primary key of this entity
- **Password.** It is used along with the username to identify a user
- **Name**
- **Lastname**
- **Avatar**. It is an optional profile picture the user can enter
- **AccountCreatedOn.** It keeps the exact moment in which the user created the account
- **LastLoggedin.** It keeps the last time the user logged into our system
- **Mail.** It keeps the user’s e-mail address. In future work thiattribute can be used to send e-mails with additional features provided by the system, to restore the user’s password and so forth 6.2.2 Session The session is the central entity related to every action that takes place on the sensor network. No session can be created if the user has not logged in or signed up, since the username is used as a foreign key in this entity. This is a vital component while

3.5 Session

The session is the central entity related to every action that takes place on the sensor network.

- **sid**. It is used as the primary key of this entity
- **UserFK**. It is used as a foreign key pointing to the user entity. Every time, a new session is created the username of the user that chose to execute the query, is stored here. The reason why this is necessary, is because we want to be able to retrieve every previously executed session along with the measurements, edges and so forth if the user chooses to see the history log
- **hasFinished**. This is a boolean attribute used to identify if the session has been completed or not. This becomes true when the user presses the stop query
- **startedAt**. It keeps the precise time in which the session started
- **finishedAt**. It keeps the precise time in which the session finished
- **isitoutlier**. This attribute is used in order to specify which application runs on the sensor network. If false, the executed application is the sensor measurement application, if true the executed application is the outlier detection app.
- **QueryOrFunc**. This attribute is used to specify the exact query or function that is being executed. If during this session the sensor measurement application is executed this field can take one of the values: sum, average, maximum, minimum or count. On the other hand, if the outlier detection application was executed it can get one of the L1, L2 or L inf options.
- **thresholdOrTc**t. This is used to keep the threshold used by the outlier detection, or the TCT of the sensor measurement application. In case the sensor measurement application is executed, it gets the value ”-1” if no TCT has been selected.
- **period**. This value shows how often new measurements arrive from the sensor network.

isTemperature. It keeps information about the physical quantity that is being measured by the sensor network. The most convenient quantity is the light intensity because it allows us to easily change the retrieved measurements by just turning on and off the lights or by covering the sensor motes with another object. This quantity has been extensively used to test the application because it enables us to test the application without having to wait for a long time, as we would have, if we tested it using the room temperature, or any other physical quantity.

measurementReceivedOn. This keeps the exact time in which a measurement was retrieved.
sessionFKey. Foreign key pointing to the session entity. 6.2.4 Edges When the sensor measurement application is initialized, an actual representation of the tree created on the sensor network has to be visualized on the Android application. In order for this to happen the edges created by the sensor network have to be stored in the database.The attributes of this entity are the following:
id. It is used as the primary key of this entity
parentNode. This keeps the id of the node that is closer to the root.
childNode. This keeps the id of the node that is further away from the root.
depth. This keeps the exact depth of the child node, in order to be easily visualized on the Android application.
sessionFKey. Foreign key pointing to the session entity.

6.2.5 outliersEdges - outlierEdgesFinal
When the sensor outlier detection application is initialized, the sensor network has to be visualized. Apart from a simple

visualization, information about the similarity of the nodes also have to appear on the client's screen. In order to achieve these visual effects this entity has to be used. Specifically, the attributes of this entity are the following:

• **id**. It is used as the primary key of this entity
• **nodeOne**. This keeps the id of the first node.
nodeTwo. This keeps the id of the second node.
createdOn. This keeps the exact moment when this edge arrived at the web service from the base-station.
hasChanged. This keeps information about changes that occur on the edges.
similarity. This keeps the exact similarity these two nodes have with each other, according to the outlier detection application.

threshold. This keeps the exact threshold the user chose when he started the application. It is used along with the actual similarity the nodes have with each other by the client to draw the edge with the appropriate color. An edge has a green color if the two connecting motes are similar or red if they are not. Since this entity is being changed periodically, these data are stored on the memory. When the user decides to stop the execution of the query, the stored data are transferred to the outlierEdgesFinal table which includes the id of the current session as a foreign key for future reference. 6.2.6 Occupied The sensor network can only be operated by a single user. Therefore a mechanism had to be implemented to avoid the access of multiple users on the same network. The "Occupied" table is used as a part of this mechanism. When a user chooses to execute a query on the network the occ value turns from false to true. Now, if a new user chooses to run a new query while the system is busy, a message shows up, informing the user that this action is not allowed. Since responsible for the interconnection between the server and the client is actually the python script, as we mentioned on the previous chapter, there is an expected latency between the time in which the user presses the stop button and the actual execution of the stopping query on the sensor network. So this table changes back to false only when this procedure has been completed and the sensor network has been reinitialized and is therefore ready to accept new connection.

Finally, in this chapter we described the database design implemented for this application. This design operates as the backbone of our system. Since the web service is stateless, database tables are used to represent some sort of state when this is required. To sum up, in this section we described the analysis of the database design, by analyzing the enhanced entity relationship model (EER) and the relational schema used for the synthesis of the database tables.

4. IMPLEMENTATION WITH ANDROID

4.1 Application

In this chapter, we describe the implementation of the Android application. This app is what the end user utilizes to interact with our system. This application should provide easy-to-use functionality, as the end-user could be virtually anyone. Additionally, a certain level of abstraction had to be implemented, because users without any prior knowledge of sensor networks, databases or web services should be able to operate our system without any problems.

4.2 Mobile Limitations

Since the targeted medium in our occasion is a mobile device, we should take into account several limitations that are bound together with these devices. Firstly, their processing power is significantly lower when compared to conventional computers. Therefore, it is preferable to execute complex calculations on the server side, and just let the client consume these results.

Additionally, battery consumption is undoubtedly a major factor. Poorly written applications, often use resources that are not really necessary, thus resulting in higher power consumption. A higher battery consumption results in a lower battery life and in a disappointed user that will uninstall our application in no time.

Storage is another important issue on these devices. Especially older devices, have very limited space that can be used by third-party applications. Since we wanted to allow the vast majority of the Android users to be able to run our application, we had to make sure that the size of our executable file was kept to the minimum.

4.3 Abstraction

Certain elements/actions can require a specific knowledge of the terminology in order to be used correctly. However it is essential to be able to hide any unnecessary elements/actions that can confuse the user. Additionally, while it is important for the user to be informed about the current state of the application at all times, a certain level of abstraction had to be implemented in order to avoid the display of unnecessary data that may confuse him or her.

In order to achieve this, many operations are performed on the background, thus creating a more natural flow of the UI

4.4 Blocking - Non-blocking Operations

There are two kinds of actions that involve the execution of code on the background, blocking and non-blocking. The first one requires the completion of a specific piece of code, before continuing to any further actions.

The reason why this operation is performed on the background is because a loading screen has to pop up in the foreground. If these operations take place on the foreground, the application will freeze.

4.5 Android's AsyncTask

Thankfully Android OS provides a useful Class that enables the execution of background operations without having to manipulate threads and handlers. Thread is a concurrent unit of execution. Each thread has its own call stack for methods being invoked, their arguments and local variables.

While AsyncTask is a very useful tool that makes the execution of background operations a specifically easy task, it does not constitute a generic threading framework. It should be used for short operations with a small duration of a couple of seconds at most. An asynchronous task is defined by 3 generic types, called Params, Progress and Result, and 4 steps, called onPreExecute, doInBackground, onProgressUpdate and onPostExecute.

4.6 Android's Background Service

Let's say that an Android application requests some measurements from the server but the returning values have to be displayed on more than one screens. Since the data that are about to be displayed are the same, it is rather inefficient to use AsyncTasks on each of those screens to request them multiple times. On the other hand, we could request the data using a single background operation and store them locally on

the phone. That way when these data are needed again they can be accessed directly from the phone.

4.7 kSOAP2

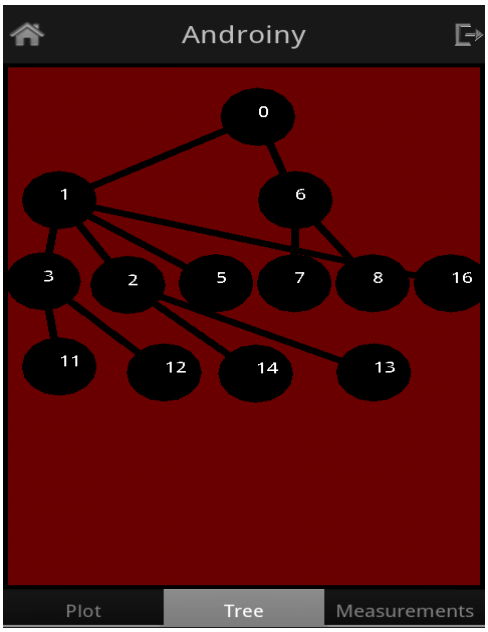
As we mentioned earlier we used a SOAP web service to feed the client with the needed information. Unfortunately, Android OS does not natively support the interaction with SOAP web services. Additionally, while JAX-WS framework (which was used for the development of the web service) could also be used to generate the client stubs, which would enable the client-server communication, the overall size of the application would have been extremely big for a mobile application.

KSOAP2-android provides a lightweight and efficient SOAP client library for the Android platform. It is a fork of the kSOAP2 library that is tested mostly on the Android platform, but should also work on other platforms that use Java libraries. KSOAP2 is also still using Java 1.3, so it should work fine on Java ME, Blackberry and so on. Furthermore, ksoap2-android is licensed under MIT and can therefore be included in any commercial application.

4.8 Canvas

As we have mentioned multiple times, the main purpose of this application is to help the user operate a sensor network. Two operations are supported. The first one is to detect outlier nodes within the sensor network and the second one is to execute an aggregation query and inform the user about the measurements received from the sensor network. In both these occasions, the user has to be aware of the formed network. During the outlier detection functionality, the user has to be able to get information such as the outlier nodes and precisely view how different these nodes are from the neighboring ones. On the sensor measurement functionality, information about the created tree by the sensor network, has to be displayed.

Figures 4.1 show some of the drawings on the canvas.



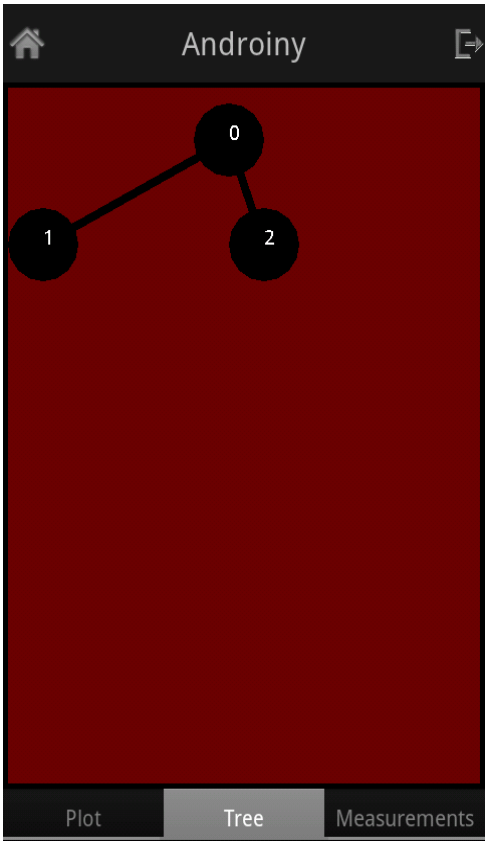
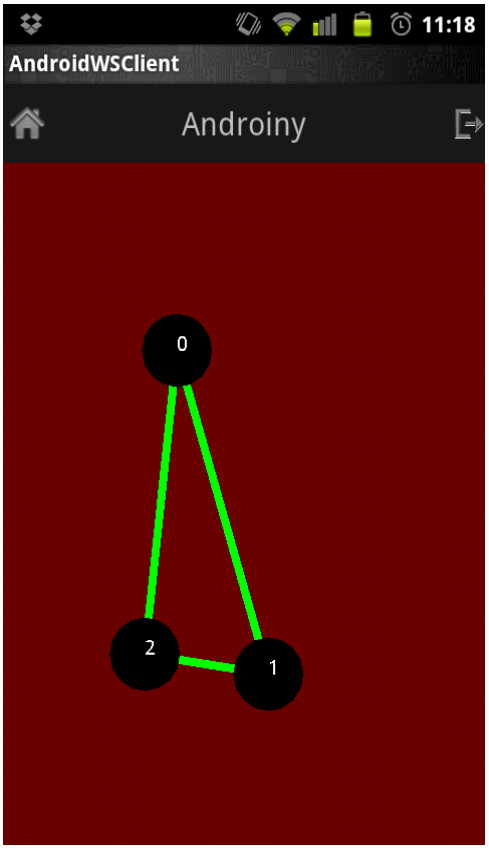


Figure 4.1: Canvas Drawings for Sensor Measurements



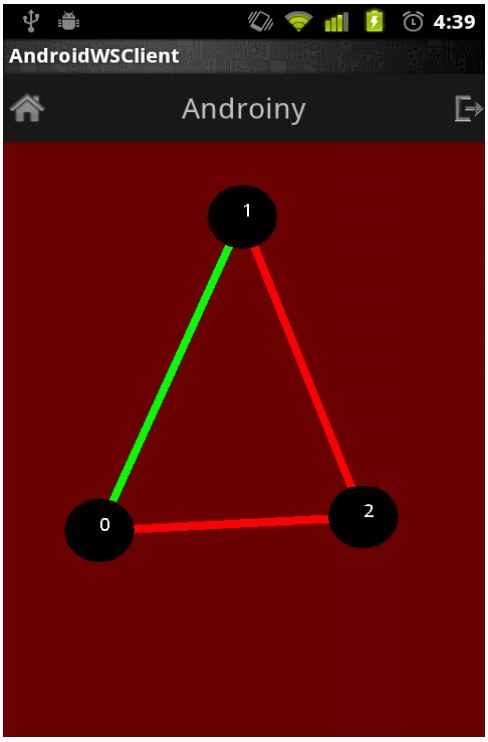


Figure 4.2: Canvas Drawings for Outlier Detection

AChartEngine - A Charting Library for Android Applications

AChartEngine is a charting library available for use in Android Applications. It supports all the Android SDK versions starting from 1.6. Since as we mentioned versions prior to Android 1.6 do not support pinch to zoom gestures, it displays zoom in and out buttons to cover this functionality. By the time these words were written the devices that used a version of Android from 1.6 and above were more than 99% in a global scale.

AChartEngine currently supports the following chart types:

- Line Chart
- Area Chart
- Scatter Chart
- Time Chart
- Bar Chart
- Pie Chart
- Bubble Chart
- Doughnut Chart
- Range (high-low) Bar Chart
- Dial Chart / Gauge
- Combined (any combination of Line, Cubic Line, Scatter, Bar, Range Bar, Bubble) Chart

Cubic Line Chart All the above supported chart types can contain multiple series, can be displayed with the X axis horizontally or vertically and support many other custom features. The model and the graphing code is well optimized such as it can handle and display huge number of values. In figures 4.3

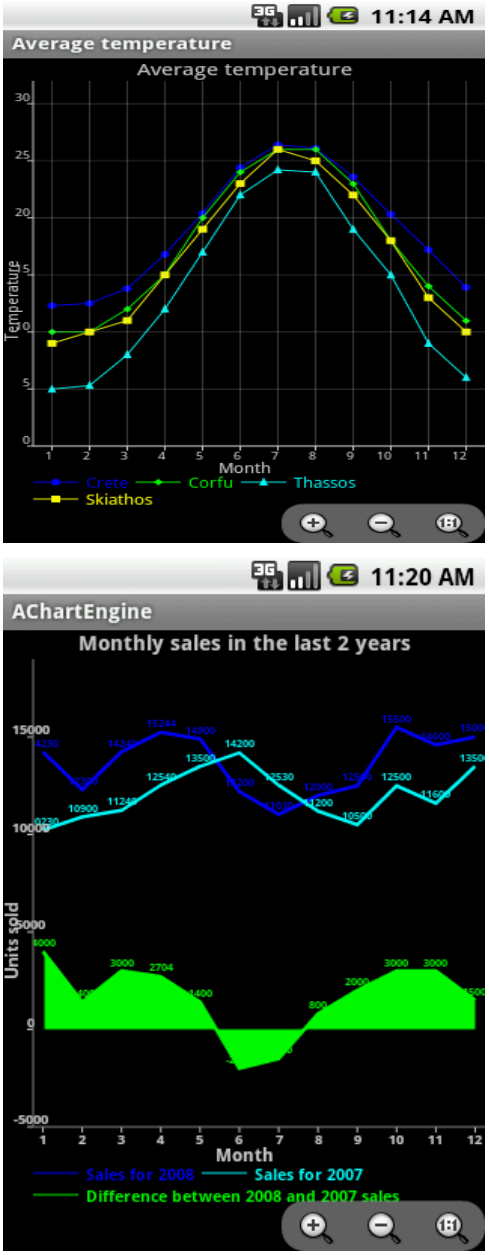


Figure 4.3: AChartEngine examples for mobile phone

Finally the client application is perhaps the most important component of our system. While the code itself is not as challenging as the TinyOS application (even if the overall number of lines was extremely high when compared to the web service or the TinyOS application), there were many aspects that had to be taken into consideration. Firstly, the look and feel of the application had to be appealing. It is important to make the user feel satisfied after using our application, so we had to make sure that warm, and cheerful colors were used instead of cold or dark colors which are known to generate negative emotions. Apart from good-looking, the application had to be useful. Users are expected to operate a sensor network while on the go, so it is very important to have an easy-to-use interface. In order to achieve this, we used graphic elements that are already well-known to Android users as they have been used extensively on other famous Android applications.

5. CONCLUSIONS

In this thesis implementation, we created a complete system that enables the manipulation of a sensor network using a mobile application. The user of this system is able to execute operations such as simple sensor measurement retrievals or outlier detection functionalities, without having to deal with bloated software that requires advanced programming skills to run. Additionally, while several applications have been developed in the past, enabling the direct manipulation of a sensor network, neither of them was free from certain limitations.

The vast majority of these applications is intended to run on older versions of the Windows operating system. Undoubtedly, there were some obvious restrictions that arose from this fact. The user had to specifically install outdated versions of an operating system he would not use for anything else. Dual/Triple-booting, installations of virtual machines and other complicated operations had to be performed in order for him to be able to operate a simple sensor network. Apart from that, he was forced to sit behind a screen in order to operate on the sensor network since Windows is a desktop operating system.

These limitations, certainly do not apply to our system. Firstly, the client program used by the end user to enable him interact with our system is an Android application. This application was developed while keeping in mind that it had to run flawlessly both on older and newer versions of Android. Additionally, since Android is a very popular operating system with millions of activations per day, it is rather self-explanatory that the target audience is particularly big in number, so there should not be any real limitations there. It is worth mentioning that, the fact that Android is such a successful.

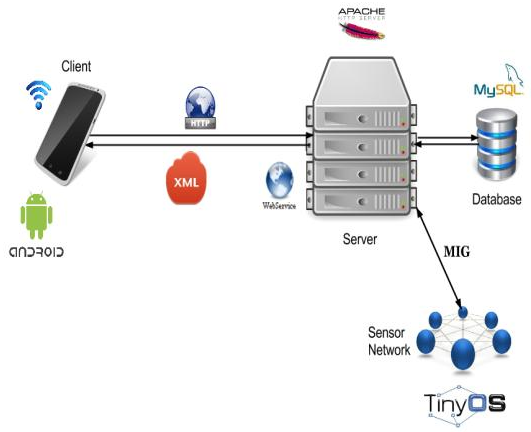


Figure 5.1: Architecture of the Implemented System.

Operating system, works in our favor. A new cheap Android device can be purchased in a price lower than that of a sensor mote. This is far cheaper than buying a brand new computer, so there is also a certain advantage on this part too.

Perhaps the most important advantage of this system is the mobility factor. The users can be virtually anywhere and still be able to use our system, provided that there is an internet connection. The fact that this system is using a web service, as an intermediate level between the Android application and the actual sensor network can verify the flexibility that comes with our system. The web service also guarantees the security of our system, since the client cannot not operate directly on the sensor network. That way several activities performed by potential hackers with intentions to cause problems to our system, are limited. Additionally, since this is a SOAP application, there is literally no limitations concerning the operating system the server uses. The web service is hosted on conventional HTTP servers, such as Glassfish or Tomcat which run both on Windows and Linux operating systems.

6. REFERENCES

[1] Philip Levis and David Gay. TinyOS Programming. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[2] Nicolas Burri, Roland Flury, Silvan Nellen, Benjamin Sigg, Philipp Sommer, and Roger Wattenhofer. Yeti: an eclipse plug-in for tinyos 2.1. In David E. Culler, Jie Liu, and Matt Welsh, editors, SenSys, pages 295–296. ACM, 2009.

[3] Mohamed A. Sharaf, Jonathan Beaver, Alexandros Labrinidis, and Panos K. Chrysanthis. Tina: a scheme for temporal coherency-aware in-network aggregation. pages 69–76, 2003.

[4] Samuel Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. Tag: a tiny aggregation service for ad-hoc sensor networks. SIGOPS Oper. Syst. Rev., 36(SI):131–146, December 2002.

[5] Sabbas Burdakis and Antonios Deligiannakis. Detecting outliers in sensor networks using the geometric approach. In Kementsietsidis and Salles [6], pages 1108–1119.

[6] Anastasios Kementsietsidis and Marcos Antonio Vaz Salles, editors. IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012. IEEE Computer Society, 2012.