Dashboards of Service Fulfilment and Assurance for Mobile: Multi-plataform versus Native

António Paulo Cardoso Jr.

Thesis to obtain the Master of Science Degree in Telecommunications and Informatics Engineering

Supervisors: Prof. Rui António dos Santos Cruz  
Eng. Sónia Moura

Examination Committee

Chairperson: Prof. Paulo Jorge Pires Ferreira  
Supervisor: Prof. Rui António dos Santos Cruz  
Member of the Committee: Prof. Nuno Miguel Carvalho dos Santos

May 2015
Acknowledgments

I would like to thank my parents for their friendship, encouragement and caring over all these years, for always being there for me through thick and thin and without whom this project would not be possible. I would also like to thank my grandparents, aunts, uncles and cousins for their understanding and support throughout all these years.

I would also like to acknowledge my dissertation supervisors Prof. Dr. Rui Cruz and Eng. Sónia Moura, support and sharing of knowledge that has made this Thesis possible.

Last but not least, to all my friends and colleagues that helped me grow as a person and were always there for me during the good and bad times in my life. Thank you.
Abstract

The Service Fulfilment and Assurance (SFA) products division at Nokia is developing mobile applications to produce dashboards with the key information required by Chief Technical Officers (CTOs) in the telecommunications industry. But the diversity of mobile operating systems and their fragmentation in terms of different versions/platforms turns the development of mobile applications a very difficult task, requiring an exhaustive testing suite for each target mobile platform. The objectives of the work is the development of a mobile application capable of displaying Operations Support System (OSS) and Service Fulfilment and Assurance (SFA) data in real-time, and to evaluate whether the performance of the application developed in a multi-platform environment has impact on the usability. The prototype solution developed in a multi-platform environment proved to satisfy the requirements with its performance meeting Nokia’s standard values.

Keywords

Mobile Application; Android; PhoneGap; iOS; OSS; Nokia HERE
Resumo

O departamento de *Service Fulfilment and Assurance (SFA)* da Nokia tem como objetivo o desenvolvimento de aplicações móveis para produzir ecrãs com as informações-chave requeridas por Diretores Técnicos no setor de telecomunicações. A diversidade de sistemas operacionais móveis e a sua fragmentação em termos de diferentes versões transforma o desenvolvimento de aplicações móveis uma tarefa muito difícil, exigindo uma suite de testes exaustivos para cada plataforma móvel de destino. O objetivo deste trabalho incide no desenvolvimento de uma aplicação móvel capaz de exibir dados dos *Operations Support System (OSS)* e *Service Fulfilment and Assurance (SFA)* em tempo real, e avaliar se o desempenho da aplicação desenvolvida em ambiente multi-plataforma tem impacto sobre a sua usabilidade. O protótipo desenvolvido em ambiente multi-plataforma comprovou satisfazer os requisitos exibindo um desempenho em conformidade com os valores de referência da Nokia.

Palavras Chave

Aplicações Móveis; Android; PhoneGap; iOS; OSS; Nokia HERE
## Contents

1 Introduction
   1.1 Objectives and Contributions ........................................... 4
   1.2 Document structure ..................................................... 6

2 Background
   2.1 Nokia Networks Reference Architecture ............................. 9
   2.2 Nokia Networks ......................................................... 10
   2.3 Development for Mobile Application Platforms ....................... 11
      2.3.1 Native Application Development ................................. 12
      2.3.2 Web Application Development .................................. 13
      2.3.3 Hybrid Application Development ............................... 13
   2.4 Development Frameworks and Tools .................................. 14
      2.4.1 Phonegap Framework ............................................ 14
      2.4.2 Appcelerator Titanium Framework ............................... 15
      2.4.3 JavaScript framework ........................................... 16
      2.4.4 Charting Tools .................................................. 16
      2.4.5 AMCharts Library ............................................... 16
      2.4.6 jqChart Library ................................................ 17
      2.4.7 CanvasJS Library ............................................... 17
      2.4.8 Highcharts Library .............................................. 17
      2.4.9 ZingChart Library .............................................. 18
   2.5 Geo-Location ............................................................ 19
      2.5.1 GeoJSON .......................................................... 19
      2.5.2 Google Maps ..................................................... 20
      2.5.3 HERE Maps ....................................................... 20
   2.6 Platform Components .................................................. 20
      2.6.1 Liferay Portal framework ...................................... 20
      2.6.2 Communication Modules ......................................... 21
3 System Design and Development
  3.1 Platform Design ......................................................... 25
    3.1.1 Communication Module ......................................... 26
    3.1.2 SQM Mobile App ................................................ 27
  3.2 Development Environment ........................................... 28
  3.3 Development Process ................................................ 29

4 Evaluation .................................................................. 37
  4.1 Test Environment ....................................................... 39
  4.2 Analysis and evaluation of Test Results ........................... 40
    4.2.1 Stability ............................................................ 41
    4.2.2 Traffic and Response Times .................................... 41
    4.2.3 Usability ........................................................... 45
  4.3 Final Remarks .......................................................... 45

5 Conclusion and Future Work .......................................... 47
List of Figures

1.1 SFA Dashboard Mockup .............................................................. 5
1.2 Mockup of the Mobile Application Main Menu ............................ 5

2.1 eTOM Level-0 ............................................................................ 10
2.2 Customer Experience Management and Operations Support System ......................................................... 11
2.3 Reporting Workflow .................................................................. 12
2.4 Phonegap build ........................................................................ 14
2.5 Phonegap Features .................................................................... 15

3.1 SQM Mobile Platform ................................................................. 25
3.2 Communication Module .............................................................. 26
3.3 Mockup of the SQM Mobile App Main Menu ............................ 27
3.4 Configuration of VPN ................................................................. 29
3.5 Login Screen ............................................................................. 31
3.6 Home Menu .............................................................................. 31
3.7 Selection Menu ........................................................................ 32
3.8 Burger Menu ............................................................................ 33
3.9 Screen Service Dependencies .................................................... 33
3.10 Localization Service Screen ...................................................... 34
3.11 Problems of Service ................................................................. 35
3.12 Attributes of Service Screen .................................................... 35

4.1 Login Time ................................................................................ 41
4.2 Loading time of the Home screen ............................................ 42
4.3 Loading time of the Service Problem screen ............................ 42
4.4 Loading time of the Dependencies screen ............................... 43
4.5 Loading time of the Service Map screen .................................. 44
4.6 Loading time of the Attributes screen (with a Chart) ............... 44
List of Tables

1.1 Result of Surveys from Nokia Customers ................................. 4
2.1 Heuristics compliance of Charting Tools [1] ................................. 17
2.2 Type of Charts supported by Charting Libraries ............................... 18
4.1 Evaluation Test Suite ................................................................ 39
4.2 Test Scenario A ................................................................ 39
4.3 Test Scenario B ................................................................ 40
# Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Program Interface</td>
</tr>
<tr>
<td>AS</td>
<td>Application Server</td>
</tr>
<tr>
<td>ADT</td>
<td>Android Development Tools</td>
</tr>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>CTO</td>
<td>Chief Technical Officer</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheets</td>
</tr>
<tr>
<td>CEMOSS</td>
<td>Customer Experience Management and Operations Support System</td>
</tr>
<tr>
<td>DS</td>
<td>Database Server</td>
</tr>
<tr>
<td>DMZ</td>
<td>Demilitarized Zone</td>
</tr>
<tr>
<td>eTOM</td>
<td>Enhanced Telecom Operations Map</td>
</tr>
<tr>
<td>AS</td>
<td>Application Server</td>
</tr>
<tr>
<td>LB</td>
<td>Load Balancer</td>
</tr>
<tr>
<td>EMS</td>
<td>Element and Manager System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation</td>
</tr>
<tr>
<td>Acronym</td>
<td>Full Form</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
</tr>
<tr>
<td>JSONP</td>
<td>JSON with padding</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicator</td>
</tr>
<tr>
<td>KQI</td>
<td>Key Quality Indicator</td>
</tr>
<tr>
<td>MBB</td>
<td>Mobile BroadBand</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>OSS</td>
<td>Operations Support System</td>
</tr>
<tr>
<td>REST</td>
<td>Representational state transfer</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SID</td>
<td>Shared Information/Data Model</td>
</tr>
<tr>
<td>SQM</td>
<td>Service Quality Manager</td>
</tr>
<tr>
<td>SVG</td>
<td>Scalable Vector Graphics</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SFA</td>
<td>Service Fulfilment and Assurance</td>
</tr>
<tr>
<td>SLA</td>
<td>Service Level Agreement</td>
</tr>
<tr>
<td>TAM</td>
<td>Telecom Application Map</td>
</tr>
<tr>
<td>TMF</td>
<td>TeleManagement Forum</td>
</tr>
<tr>
<td>TO</td>
<td>Technical Officer</td>
</tr>
<tr>
<td>UI</td>
<td>User Interface</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VPN</td>
<td>Virtual Private Network</td>
</tr>
<tr>
<td>WTP</td>
<td>Eclipse Web Tools Platform</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
1 Introduction

Contents

1.1 Objectives and Contributions ............................................. 4
1.2 Document structure ....................................................... 6
The telecommunications industry considers Operations Support System (OSS) not just an essential mechanism to automate and control businesses operations but also the must-have tool for driving improvements in internal operations performance. In other words, the OSSs must automate the value chain in order to achieve operational excellence through flexible and compelling service offerings, efficient service acquisition and superior service delivery.

The business competition in this area is increasingly fierce with the technologies that support daily business decisions evolving at a galloping pace. For timely decisions and efficient and effective management, these types of companies need access to updated information in the shortest time possible.

It is the process of collecting large amounts of data, their analysis and presentation in high-level reports integrating the essence of such data, that allows management decisions to be taken daily, for the most basic of business actions.

With the increasing number of smartphones in the market it starts to become interesting to use these equipments to increase the efficiency not just in personal matters but also in professional matters.

Additionally, the overwhelming number of applications for daily use in mobile devices, from tablets to smartphones, is allowing businesses to access the information they need, typically received in the form of graphs and dashboards. However, this wide range of applications must define features and functionalities, including detailed evaluation matrices, in order to differentiate and thus facilitate the evaluation of those applications, and provide the best user experience matching the user requirements.

This new market naturally compels carriers and service providers to provide a large spectrum of services, including the newly developed, quickly, in a cost-effective manner, and with a high-degree of customization, such as high flexibility and attractive interfaces for customers, other operators or carriers and their suppliers [2].

One of the key areas in OSS addressing the aforementioned goals is the Service Fulfilment and Assurance (SFA). With the purpose of increasing efficiency, the SFA products division at Nokia is developing mobile applications able to produce dashboards with the key information eagerly required by Chief Technical Officers (CTOs) in the telecommunications industry.

But the diversity of mobile operating systems in the market, or more specifically, their fragmentation in terms of different versions/platforms with different software and hardware capabilities, turned the development of mobile applications a very difficult task, as the same application is required to be developed and exhaustively tested for each target mobile platform. The typical process of developing native applications for mobile platforms should be the appropriate way, however, it currently bears a huge disadvantage as it is hard to reuse the source code developed for some target mobile platform in a different platform/mobile Operating System (OS), or even a different version of the same hardware platform. In other words, the same application must be typically built (almost) from scratch for each mobile OS. Native applications (Apps) are typically developed using an Integrated Development Environment (IDE).
that provides the necessary tools for building and debugging them. The development of native Apps requires a much higher-level of experience and technological know-how than the development of other types of applications [3].

However, under certain scopes, a multiplatform application development technology may solve the aforementioned difficulties, by enabling the reuse of the same source code to produce the App targeted to various mobile OSs/platforms.

In order to address those issues, one of the objectives of the work presented in this thesis is related to the development of one such mobile application capable of displaying OSS SFA data from the network elements of a provider, allowing end-users (operations or management personnel) to access that information in real-time. Another objective of the work is the evaluation of the performance of such an App, developed in a multiplatform environment, in terms of the impact in its usability. The reasons for the evaluation are mainly due to the higher costs incurred when developing native Apps for each of the existing mobile OSs/platforms.

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Who uses mobile equipment?</td>
<td>CTO</td>
</tr>
<tr>
<td></td>
<td>Information needs are different depending on the user</td>
</tr>
<tr>
<td>What kind of information must be displayed?</td>
<td>OSS/TPIM/SQM – all the way up to CTO</td>
</tr>
<tr>
<td></td>
<td>Network director</td>
</tr>
<tr>
<td></td>
<td>Manager</td>
</tr>
<tr>
<td>Periodicity of information?</td>
<td>Real time graphics on network/service status</td>
</tr>
</tbody>
</table>

Table 1.1: Result of Surveys from Nokia Customers

To determine the way forward in the optimization of functions already provided by Nokia solutions for the development of the App, some surveys were performed involving current customers of the company. The data collected is summarised in Table 1.1.

From the survey results, the scope of this project has been tuned for the development of a mobile App targeted to be used by CTOs of the telecommunications industry, based in dashboards containing only the necessary information about SFA updated in real-time.

1.1 Objectives and Contributions

The objective of the work described in this thesis, is the analysis performance a multiplatform development framework. This mobile App will be based in dashboards. A mockup of the dashboards is illustrated in Figures 1.1 and 3.3.

The objective of the App is to display performance information of network (Elements and/or Plat-
forms) through graphics and lists, related with the main flaws and problems detected in the network.

The Nokia Service Quality Manager (SQM) is based on Services. Services are defined as the business products that are sold to the operator’s customers, such as voice, data, messaging, music or video streaming, etc. SQM models these Services internally, and monitors the impact of network state (existing problems, faults, performance degradation, etc.) on those Services, allowing an operator to support the Service Assurance processes in order to understand if the provided Services meet the agreed Service Levels.

The data processed by the App is constantly changing, as it is dependent on events (alarms) that are generated in real-time in the network. In each “view” provided by the App, the details of a specific service data must be loaded in order to ensure that the data presented are the most recent available in the system. This requires the management of a fairly huge amount of data (typically many thousands of
records). This is one of the main challenges for the App, as it must be able to process large amounts of data in very short intervals of time (i.e., whenever the end-user “selects” a certain “view” of a network Element or Platform). This is the key motive for the performance analysis between an App developed using an hybrid mode, i.e., using frameworks that enable the abstraction of specifications of each OS.

Another key aspects of the App is to be able to create graphics on the fly from the received data. In order to achieve the desired solution, some frameworks specialized in the generation of graphs were also studied in order to select the most suitable, being one of the key factors the reception of data in JavaScript Object Notation (JSON) format.

The App must also have the capability of displaying maps with geo-referenced location of the various components of the network, created from data received in GeoJSON format.

1.2 Document structure

This document is organized as follows. Chapter 2 will describe the necessary background information, Chapter 3 will present the system design and development, followed by the methods used to test and evaluate the prototype in Chapter 4. Chapter 5 draws conclusions on the work performed and presents perspectives for future work.
2

Background

Contents

2.1 Nokia Networks Reference Architecture ........................................... 9
2.2 Nokia Networks ............................................................................. 10
2.3 Development for Mobile Application Platforms ............................... 11
2.4 Development Frameworks and Tools .............................................. 14
2.5 Geo-Location ............................................................................... 19
2.6 Platform Components ................................................................. 20
This chapter addresses all the conducted studies for the development of the project. It starts by introducing Nokia as a company and also the OSS network architecture. Then, the research related with Development for Mobile Application Platforms that enable the design of mobile applications. It also addresses the development frameworks and tools required for the project.

Given that the application developed makes a great use of the capabilities to analyze large amounts of data to produce charts, and taking into account the cross-platform development, the research also focused on the analysis of suitable charting libraries.

Another relevant aspect that is required is the support for geo-referencing with graphical maps displaying the location of each service/network element. It is necessary to find adequate mapping tools using Application Program Interfaces (APIs) that allow manipulation and customization.

### 2.1 Nokia Networks Reference Architecture

Nokia is a multinational company, manufacturer of data networks and telecommunications equipments and services solutions, based in Espoo, Finland, that uses the brand – Nokia – of the parent company in all products. Nokia Networks is the global expert in mobile broadband, offering efficient mobile network products and services as well as market intelligence solutions that maximize the value of the networks and services allowing them to work in the best possible conditions.

The “Network Architecture” is considered a specific domain of an operator due to its huge importance. In terms of Nokia Networks products for operators, a special emphasis has been put on linking the “Network Architecture” with the Nokia Networks Reference Architecture, enabling an operator to make key decisions on its investments. Rather than trying to fit everything into one gigantic view, the Nokia Networks Reference Architecture is split into six viewpoints of smaller areas.

One of the multiple products developed by Nokia is the Service Quality Manager (SQM), an application inserted in the Nokia Networks Reference Architecture that introduces new features for quality assurance of basic services in the product.

The Nokia Networks Reference Architecture is based on open standards to ensure it suits the needs of most operators.

The starting point for the concept of the Nokia Networks Reference Architecture was the TeleManagement Forum (TMF)'s Frameworx, which consists of four enterprise layers: the Enhanced Telecom Operations Map (eTOM) Business Process Framework, the SID Information Framework, the TAM Application Framework and the Integration Framework. Figure 2.1 illustrates this model and defines the processes related with Assurance and Fulfillment.

The Nokia Networks Reference Architecture has also been extended with several enhancements that address high-priority areas requested by operators, such as Customer Experience Management,
Self Organizing Networks and Network and Service Operations.

2.2 Nokia Networks

Nokia is highly focused on Mobile BroadBand (MBB) and considers fundamental to have Operations Support System (OSS) applications able to answer all faults in the network in the shortest possible time. All this paradigm is situated in the business area named Customer Experience Management and Operations Support System (CEMOSS).

In Figure 2.2 this business area of Nokia is illustrated, namely the components of the Network Management products that the company sells to operators. The NetAct, is the key Element Manager, thereby allowing other applications of the company to work over this system [4].

A very wide range of OSSs such as Inventory, Order Management, Provision and Service Activation are included in the Fulfilment process area. Network failures may also be pro-actively identified by Service Assurance, initiating resolution actions and notifying high-priority customers. The “Order-to-Cash” cycle, in which service providers give requested services to customers in a timely and correct manner, is supported by Service Fulfilment subsystems. Service Assurance solutions do not monitor service performance based on the network manager’s view but on the customer’s view, defining the
following key indicators:

- **Key Performance Indicators (KPIs);**

- **Key Quality Indicators (KQIs);**

- **Service Level Agreements (SLAs).**

These solutions help service providers to connect network and service performance with the end-user experience.

In Nokia, until recently, the interface to all those services was available as a web interface accessible via a browser using SQM, making easy to an operator to identify if the quality of its basic services was being degraded. Via these processes operations technicians are able to prioritize the resolution of problems through a set of corrective actions. The Technical Officer (TO) module of the solution produces, usually daily, reports of all issues identified through the various tools to be typically delivered to the CTO of the operator, as represented in Figure 2.3.

### 2.3 Development for Mobile Application Platforms

There are many different platforms for mobile devices: **Google’s Android**, **Apple’s iOS**, **Microsoft’s Windows Phone**, **Blackberry**, among others. It is therefore a very competitive and fragmented market scenario with very rapid changes. In addition to this diversity of systems, each platform adopts different programming language paradigms, development tools and environments.
Native applications (Apps) are developed using an IDE requiring a high-level of experience and technological know-how.

One alternative to Native application development, circumventing the aforementioned complexity, corresponds to Web development frameworks that use just web technologies such as HyperText Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript for the web Apps, facilitating the development with consolidated technologies and also presenting sets of APIs to access some features of the hardware and operating system of mobile platforms.

Other alternatives are Hybrid application development frameworks that embed HTML5 Apps inside thin Native application containers where the source code to be executed by a browser is packaged within the application. Hybrid applications are installed on the device and allow access to the device hardware.

These frameworks have as objective the development of applications suitable to run in various platforms, encoding only once the application [5–7]. Among the frameworks that stand out are the Phonegap [8] and the Titanium Mobile [9].

2.3.1 Native Application Development

For the development of Native applications an IDE is required, providing all the necessary tools and components for building the applications.

The main mobile Operating Systems available nowadays on the market, are undoubtedly Google’s Android (around 80% share) and Apple’s iOS (around 15% share), with Microsoft’s Windows Phone (around 4% share) seeing some growth. For the work of this thesis the first option to consider is Google’s Android, due to its specifications and popularity, with Apple’s iOS as the second option in terms of Native App development.

Apple’s iOS was developed by Apple Inc. for their portable devices (initially the iPod touch then the iPhone in 2007 and later the iPad and the Apple TV). iOS controls the hardware of the devices and Apple supplies all the necessary Development Environment technologies (Xcode) for the implementation and testing of native applications. iOS is a mobile (embedded) version of Apple’s OS X operating system, a
Berkeley Unix system.

Google’s Android is an operating system based on the Linux kernel with a user interface based on direct manipulation, designed primarily for touchscreen mobile devices such as smartphones and tablet computers using touch inputs. Touch inputs loosely correspond to real-world actions, like swiping, tapping, pinching, and reverse pinching, used to manipulate on-screen objects or a virtual keyboard. Despite being primarily designed for touchscreen input, Android is also used in embedded systems such as televisions, game consoles, digital cameras, and other consumer electronics devices [10].

Android’s source code is released by Google under open-source licenses, although most Android devices ultimately ship with a combination of open-source and proprietary software [10]. Initially developed by Android, Inc., which Google backed financially and later bought in 2005 [11], Android was unveiled in 2007 along with the founding of the Open Handset Alliance —- a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices [12].

2.3.2 Web Application Development

Web Apps are browser-based applications where the software to run is downloaded from the web. Web Apps are based on common code used on the Internet, such as HTML and JavaScript technologies. One of the major disadvantages of this type of applications is the limited access to hardware and device data. Another disadvantage is the time required to process the web pages and the additional cost required to download a web page from the Internet.

One of the great advantages of Web applications is that they require no installation and subsequent updates. By contrast, since the application is not installed on the device, there are situations, such as when the device is in Airplane mode, where web applications are inaccessible to the end-user.

Today, many software libraries allow the development of web applications that simulate the functionality of native Apps. Examples of such libraries are JQuery Mobile, Sencha Touch, JQTouch, WeApp.net, Xui and many others. It is noteworthy that the development and adoption of the HTML5 standard promises access to hardware and software of the device through the use of APIs. HTML5 is evolving to a mobile friendly version, focuses on standardization and a range of new mechanisms is available on any HTML5 -compliant web browsers [3].

2.3.3 Hybrid Application Development

Hybrid Apps combine the advantages of Web Apps and Native Apps. The Hybrid Apps are built largely resorting to the use of HTML5 and JavaScript, making possible to develop an application without a detailed knowledge of the target platform. Hybrid Apps embed HTML5 code inside a thin Native
App container (*UIWebView* in *iOS* and *WebView* in *Android*). Like Web Apps, the source code is still executed by a thin browser that is part of the final application and can be packaged with the App. Unlike Web Apps, where the source code must be downloaded from the web before running the App, Hybrid Apps are installed in the device and allow access to the hardware. Data access is also done through the use of APIs. *Phonegap* and *Titanium* are examples of the most popular containers for building Hybrid mobile Apps.

The code implementation of the Hybrid Apps can be built using various technologies and development platforms but in order to achieve a native look and feel it is necessary to use specific development libraries, such as *jQuery* [3].

### 2.4 Development Frameworks and Tools

This section describes the research related with development frameworks and tools for producing mobile application platforms. Tools such as *PhoneGap* and *Appcelerator Titanium* will be described, as well as the Javascript framework used for the development of this project and the other complementary tools for Charting and Geo-location.

#### 2.4.1 Phonegap Framework

*PhoneGap* is a mobile development framework produced by Nitobi.

![Phonegap build](image)

**Figure 2.4:** Phonegap build

Nitobi was the original creator and is one of the primary contributors to the *PhoneGap* framework. In October 2011, Adobe acquired Nitobi enabling the team to focus solely on the *PhoneGap* project and continue its work on efficient development across mobile platforms. It enables software programmers to build applications for mobile devices using JavaScript, *HTML5*, and *CSS3*, instead of traditional
languages such as Objective-C [8] or Java.

The resulting applications are hybrid, so they are neither truly native (because the whole layout rendering is done through web views instead of native User Interface (UI) framework) nor purely web based (as they are not only web applications, but applications packaged for distribution and access to native device APIs), as illustrated in Figure 2.4.

Building Native Apps for iPhone, Android, Windows Phone and others requires different frameworks and languages. **PhoneGap** solves this gap by using standards-based web technologies to bridge web applications and mobile devices. Since **PhoneGap** Apps are standards compliant, they are future-proof to work with browsers as they evolve. **Phonegap** features are described in Figure 2.5.

<table>
<thead>
<tr>
<th>Feature</th>
<th>iPhone 7/ iPhone 6s/ iPhone 6c</th>
<th>iPhone 7s/ iPhone 6s Plus</th>
<th>Android 10</th>
<th>BlackBerry OS 5.1</th>
<th>BlackBerry 10</th>
<th>Windows Phone 8</th>
<th>Ubuntu</th>
<th>Firefox OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Camera</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Compass</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Contacts</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>File</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Geolocation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Media</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Network</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Notification (Alert)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Notification (Sound)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Notification (Vibration)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Storage</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Figure 2.5: Phonegap Features**

### 2.4.2 Appcelerator Titanium Framework

Appcelerator is a privately held mobile technology company based in Mountain View, California, USA. Its main products are **Titanium**, which is an open-source development kit for mobile cross-platform software development, and the **Appcelerator Platform**, a suite of business software for mobile application development, testing, deployment and analysis.

**Appcelerator Titanium** is an open and extensible environment for the development of native applications on different mobile devices and operating systems, including **iOS**, **Android** and **BlackBerry**, as well as hybrid and **HTML5**. **Titanium** includes a framework Software Development Kit (SDK) open-source with more than 5,000 devices and mobile operating system APIs. Appcelerator also provides **Studio**, a powerful Eclipse based IDE software, consisting of an MVC framework and Cloud Services for mobile
backend, ready for use. Appcelerator **Titanium** permits to create efficient code modules reducing time to create new applications [9].

**Titanium Studio** simplifies the process of developing mobile applications, enabling developers to quickly build, test, package and publish mobile apps on multiple devices and operating systems. The developed native applications perform and behave exactly as if they were written in Objective-C (iPhone and iPad) or Java (Android phones and tablets) to deliver the best end-user experiences [9].

### 2.4.3 JavaScript framework

**jQuery** JavaScript framework is designed to permit the developer literally “write less, do more.” **jQuery** is a fast, small, and feature-rich JavaScript library. It simplifies coding such as moving through elements in HTML and their manipulation, event handling, animation, and Asynchronous JavaScript and XML (AJAX) much simpler with an easy-to-use API that works through most browsers [13].

**jQuery UI** was created with user interface elements and interactions in mind that usually require too much code to handle the elements. Thus, the library contains features such as the ability to drag and drop elements and get useful information from these interactions. But at the same time, there is a focus on the look and feel, and thus it makes available various types of themes that call those powerful user interface elements from the library [13].

**jQuery Mobile** is a powerful framework that brings together the best **jQuery** and **jQuery UI** designed specifically for mobile platforms, including all the functionality required in mobile environments, such as drag and drop adequately redesigned to support sensitive touchscreens. **jQuery Mobile** is a Touch-Optimized Web Framework and an HTML5-based user interface system designed to make responsive web sites and Apps that are accessible in all smartphone, tablet and desktop devices. This framework offers AJAX navigation with page transitions, touch events, and multiples widgets. Its lightweight code is built with progressive improvement, and has an easy and flexible design [13].

### 2.4.4 Charting Tools

To determine the most suitable Charting Tools libraries several usability tests were carried out based on predetermined heuristics (relative to charting features and functionalities). Table 2.1 lists the Charting Libraries identified as best for mobile devices and satisfying the heuristics: AMCHART, jsChart, CanvasJS, HighCharts and ZingChart, studied in [1].

### 2.4.5 AMCharts Library

**AMCharts** offers JavaScript/HTML5 charts for most needs. The framework includes serial (column, bar, line, area, step line, step without risers, smoothed line, candlestick and OHLC graphs), pie/donut,
radar/polar, y/scatter/bubble, Funnel/Pyramid charts and angular gauges (Table 2.2). Those charts offer unmatched functionality and performance in a modern, standards compliant package, and the JavaScript charting library is responsive and supported by touch/mobile devices [14].

2.4.6 jqChart Library

The jqChart uses pure HTML5 Canvas, does not uses plug-ins. It is one of the most used charting libraries for mobile devices, supporting very high resolution screens (such as Apple’s Retina displays), and presents one of the best performances among this type of frameworks, yet having advanced graphics meters with many features [15].

2.4.7 CanvasJS Library

CanvasJS is an easy to use HTML5 and Javascript Charting library. It runs across devices including iPhone, iPad, Android, Windows Phone, Microsoft Surface, Desktops, etc, allowing the creation of rich dashboards that work across devices without compromising Maintainability or Functionality. CanvasJS presents a very simple and intuitive API with different looking themes, is standalone, does not depends on any other library, has high performance and also excellent support from developers.

The CanvasJS framework is built using Canvas element and can render thousands of Data Points in a matter of milliseconds, equating to over 10 times better performance than traditional Scalable Vector Graphics (SVG) and Flash based JavaScript Charts. CanvasJS is also interactive and can be updated dynamically [16].

2.4.8 Highcharts Library

Highcharts is a product of Highsoft, AS. Highcharts has quickly become one of the industry leaders in standards compatible, JavaScript based charting tools. Highcharts is a charting library written in pure JavaScript, offering an easy way of adding interactive charts on any website or web application. It

<table>
<thead>
<tr>
<th>Table 2.1: Heuristics compliance of Charting Tools [1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>36 Heuristics</td>
</tr>
<tr>
<td>AM Charts</td>
</tr>
<tr>
<td>jqChart</td>
</tr>
<tr>
<td>Canvas JS</td>
</tr>
<tr>
<td>HighCharts</td>
</tr>
<tr>
<td>ZingChart</td>
</tr>
</tbody>
</table>
supports many types of graphs such as line, spline, area, areaspline, column, bar, pie, scatter, angular gauges and others (Table 2.2).

It works on all modern browsers and mobile desktop. On iOS and Android, multitouch support provides an excellent user experience. Standard browsers use SVG for the graphics rendering. Highcharts is solely based on native browser technologies not requiring client side plugins like Flash or Java, nor server side applications or tools.

Setting the Highcharts configuration options requires no special programming skills. The options are given in a JSON structure, which is basically a set of keys and values.

The API functions allow to add, remove and modify series and points or modify axes at any time after chart creation. Numerous events supply hooks for programming against the chart. In combination with jQuery, MooTools or Prototype’s AJAX API, Highcharts is opened to solutions like live charts constantly updating with values from the server, user supplied data and more [17].

<table>
<thead>
<tr>
<th>Chart Types</th>
<th>amCharts</th>
<th>CanvasJS</th>
<th>Highcharts</th>
<th>ZingChart</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Timeline</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Scatter</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Area</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Pie</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Donut</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Bullet</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Radar</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Funnel</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Gantt</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Network</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Grouped</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

### 2.4.9 ZingChart Library

ZingChart has hundreds of functions and can be used to create all kinds of views, from simple graphics that adhere to flat design principles to complex dashboards and graphs that are strongly stylized. Most projects require more than 100 or even 500 data points. However, the performance gains in some libraries seem to disappear with larger data size. ZingChart has a good performance in charts that use large amounts of data. Table 2.2 summarises the types of charts supported by each application.

The ZingChart framework provides users multiple ways to engage with graphics. Many resources are embedded and has a JavaScript API for even more. ZingChart uses the latest technologies such as AJAX, JSON, HTML5, and JavaScript, enabling the same work everywhere, (Android, iPad, iPhone, etc.), without any dependencies [18].
2.5 Geo-Location

Geo-Location is a very important point for this application, because by showing the location of the network components in the App, allows to identify more effectively the location of problems.

In this section the two frameworks for the creation of maps, Google Maps and Here Maps, are studied. The Geo-Location API should support the reception of data in a standard format, such as GeoJSON.

2.5.1 GeoJSON

GeoJSON is a particular case of JSON, but in this case it is used to encode geographic data structures, so there are now specific members in the format structure. A GeoJSON object may represent a geometry, a feature, or a collection of features. These objects support the following geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection. Features in GeoJSON contain a geometry object and additional properties, and a feature collection represents a list of features. A complete GeoJSON data structure is always an object (in JSON terms). The following code snippet is an example of a GeoJSON object [19].

```json
{
    "type": "FeatureCollection",
    "features": [
        {
            "type": "Feature",
            "geometry": {
                "type": "Point",
                "coordinates": [102.0, 0.5],
                "properties": {
                    "prop0": "value0"
                }
            },
            "properties": {
                "prop0": "value0",
                "prop1": 0.0
            }
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "LineString",
                "coordinates": [
                    [102.0, 0.0], [103.0, 1.0], [104.0, 0.0], [105.0, 1.0]
                ],
                "properties": {
                    "prop0": "value0",
                    "prop1": 0.0
                }
            }
        },
        {
            "type": "Feature",
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
                    [100.0, 1.0], [100.0, 0.0]
                ],
                "properties": {
                    "prop0": "value0",
                    "prop1": {"this": "that"}
                }
            }
        }
    ]
}
```
2.5.2 Google Maps

The Google Maps API allows developers to integrate Maps into their websites. It is a free service, requires only a registration on the framework site. By using this API, it is possible to embed Google Maps site into an external website, on to which site specific data can be overlaid. Google Maps API is a JavaScript API that includes a service for retrieving static map images and web services for performing geocoding, generating driving directions, and obtaining elevation profiles. This service allows the inclusion of points and define areas on maps, creating custom maps for each case. This information is received by the API through a standard format, GeoJSON [20].

2.5.3 HERE Maps

HERE Maps is developed inside the Nokia group, and its mission is “to make maps for a better life”. The HERE Maps API for JavaScript is a set of programming interfaces that enable developers to build Web applications with feature rich, interactive HERE Maps at their center. The API consists of libraries of classes and methods to implement the functionality of an interactive application similar to Google Maps, and this API receives data also in GeoJSON format, allowing to create and manipulate the maps according to what is required in each case. Unlike Google Maps, HERE Maps requires licensing that involves costs, but in this case, given that the work is developed inside the Nokia group, the cost factor does not weight in deciding the best platform to use [21].

2.6 Platform Components

Other components required for the Platform are the Portal interface and Communication Modules. The Portal interface is used as front-end, serving to support the application. This portal is responsible for all communications between the application and the remaining system components. Regarding the Communication Modules they will be used for making requests for the application, therefore the best option is based in Representational state transfer (REST).

2.6.1 Liferay Portal framework

The Liferay Portal is used in Nokia products to login in various applications and is also responsible for communication between the various components in the network, responding to all REST requests. The Liferay Portal allows to have a centralized point of access to information of all components of a network.

The Liferay Portal is used to increase value in corporate intranets and extranets. Liferay Portal is a open source project written in Java and distributed under the GNU Lesser General Public License, being
possible to acquire a commercial license. The Liferay project includes many practical projects such as Liferay Social Office, Liferay Sync and Liferay AlloyUI, among others [22, 23].

Liferay Portal is a web platform with features commonly required for the development of websites and portals. Liferay includes a built-in web content management system allowing users to build websites and portals as an assembly of themes, pages, portlets/gadgets and a common navigation. Liferay is sometimes described as a content management framework or a web application framework. Liferay’s support for plugins extends into multiple programming languages. Although Liferay offers a refined programming interface for developers, no programming skills are required for basic website installation and administration.

Liferay Portal is Java-based and runs on any computing platform capable of running the Java Runtime Environment and an application server. Liferay is available bundled with a servlet container such as Apache Tomcat [22, 24].

2.6.2 Communication Modules

Using a WebService is one of the most common way to integrate different applications. There are different types of architectures for web services. The REST architecture (and the RESTful approach) is simpler compared to other web services, which generally use Simple Object Access Protocol (SOAP). Given this simplicity, this makes the RESTful architecture a popular choice mainly for services open to the public. For example, both Twitter and Flickr have an API that follows the principles of REST architecture [25, 26]. Services that need to communicate between different architectures make available an API, supported on REST, usually because there is a growing need for these services to be integrated with various kinds of applications [27].

Following the trends, the communications between the various systems of the Nokia SQM are made using the RESTful approach. In the RESTful approach, each system is identified by a Uniform Resource Locator (URL), generally referred to as RESTful URL and the corresponding resources. These resources are entities or objects with information, typically represented in JSON encoding. In general, the URL to access the resource will be the same, but if that URL is changed, existing Hypertext Transfer Protocol (HTTP) (GET, POST, PUT, DELETE) methods are able to identify the current URL [28].

JSON encoding is a standard for simple data, derived from the syntax of objects in Javascript format. And although it is directly related to JavaScript, it is a pattern with multiple parsers in different languages, and can serve different purposes. The main current use of JSON is as alternative to Extensible Markup Language (XML) in data transmission between client and server, popularized with the use of AJAX. The following code snippet is an example of a JSON object.
One of the advantages of using JSON is that its format is usually less demanding than the equivalent in XML, and simpler to interpret, especially in the case of JavaScript, which is the language used to make AJAX requests.

JSON with padding (JSONP) is a complement to data format JSON. Provides a method to send data requests from a server to a different domain, something prohibited by typical browsers because of same-origin policy. As browsers do not enforce the same origin policy in <script> tags, the JSONP takes advantage of it. It is necessary to bear in mind that for JSONP to work, a server must know how to reply with JSONP-formatted results. JSONP does not works with JSON-formatted results. The JSONP parameters passed as arguments to a script are defined by the server. The following code snippet is an example of a JSONP object [29].

```
callback({
   'Student':{
      'name':'John', 'evaluation':[8,9,7] ,
      'name':'Sarah', 'evaluation':[8,15,7] ,
      'name':'Alex', 'evaluation':[10,11,15] ,
   }
})
```
3.1 Platform Design ......................................................... 25
3.2 Development Environment ........................................... 28
3.3 Development Process .................................................. 29
This chapter describes the design and development of the application (SQM Mobile), and the integration and connection to existing servers. The chapter is divided in three parts, being the first dedicated to the conceptual design of the Application, the second describing the development environment where all the frameworks used will be identified, and the third describing the development steps of the hybrid application.

### 3.1 Platform Design

The Platform consists of a Communication Module and the SQM Mobile App to be integrated with the SQM System.

The SQM is based on services. Services are defined as the business entities that are sold to operator’s customers, such as voice, data, SMS, music streaming, etc. SQM models these services internally, and monitors the impact of network state (existing problems, faults, performance degradations, etc) on those services, thus allowing an operator to support the Service Assurance process understanding if delivered services meet agreed service levels.

![Figure 3.1: SQM Mobile Platform](image)

The Communication Module will be implemented following the concepts of RESTful, with objects of the data model encoded in JSON/JSONP, as currently used by Nokia for communications between the various systems of products. The response to these requests is the responsibility of Liferay Portal, as already described, and this is a central point for all the required information. Figure 3.1 illustrates the current architecture of SQM, composed of Database Servers (DSs), Application Servers (ASs), Load Balancers (LBs), Element and Manager Systems (EMSs) and Graphical User Interfaces (GUIs).
modules, into which the SQM Mobile App will be integrated.

A load balancer, not shown in Figure 3.1, will be responsible for determining which of the front-end GUIs the SQM Mobile will bind. After determining the GUI, the connection between the equipment and the communication module of the GUI is established enabling to send requests and to receive data in the mobile device.

SQM Mobile will then be responsible for the analysis and manipulation of the data received, to be subsequently made available to the user as intended. The data will be requested from the current system through REST requests, with these requests returning JSON objects.

3.1.1 Communication Module

One communication module will be installed for each existing GUI. The mobile device will communicate with the communication module using REST calls in secure transport channels. If the GUI was not assigned a Public Internet Protocol (IP), meaning that the front-end would be in a Demilitarized Zone (DMZ), then a Virtual Private Network (VPN) connection between the mobile device and SQM will have to be established. As the data model to be used is the same as the one already used in communications between the various sub-systems of the SQM solution, it will not be necessary to define a new data model. The HTTP methods for sending and getting data (GET, SET, DELETE, ...), in order to have the proper interpretation of the data and therefore adequately display it in the dashboards, will also have to be defined, as illustrated in Figure 3.2.

![Communication Module Diagram](image)

**Figure 3.2: Communication Module**

Considering that SQM Mobile App will manipulate existing data to other Nokia services, the communication modules to be used already exist and are used in other products from Nokia. Therefore the main focus of this work will be the efficient manipulation of data.
3.1.2 SQM Mobile App

The data processed by the SQM Mobile App is constantly changing, as it corresponds to events (alarms) that are generated in real-time by the Services (Service Platforms or Network Elements) of an Operator.

As such, in each “view” of the SQM Mobile App (a mockup of the SQM Mobile App UI is illustrated in Figure 3.3), the details of a Service must be loaded in order to ensure that the data shown corresponds to the latest available in the system. This requires a management a large amount of data. This is the main challenge of the SQM Mobile App, as it should be able to process large amounts of data in near real-time.

One of the objectives of the work was the analysis and comparison of performance between an application developed as Native and an application developed using a multi-platform development framework. The development work started with a multi-platform framework environment, using HTML, CSS, and JavaScript, and the charting library AMCHART, in order to display the behavior of different types of Service Platforms or Network Elements, according to the patterns already identified in the work of [1]. The application was then compiled for the Android OS target, and tested following a pre-defined evaluation suite (briefly described in Section 4) following standard Nokia procedures.

Although initially considered in the scope of works, the Native application was not developed, as it was determined that the duration of the thesis work would not be sufficient for the full development of two mobile applications. However there existed reference values for application performance, provided by Nokia, documented in Section 4 that were used to carry out several performance tests, from reception times and loading data to the amount of data transferred. The resulting values from the developed application were used to compare with the reference values.

In relation to the mapping API component of the SQM Mobile App, the necessary requirements
such as, support receiving data in GeoJSON structures and allow the construction of Geometry Objects (Points, Multipoints, Polygons and MultiPolygon) were met by both Nokia’s HERE Maps and Google Maps, the fact that the HERE Maps solution belongs to the Nokia group, weighted on its selection.

3.2 Development Environment

The multi-platform framework environment for the development of the SQM Mobile App was the PhoneGap framework.

It was therefore necessary to create that development environment by installing several plugins, frameworks and programs, such as:

• Java JDK v1.6.0_45
• NodeJS v0.10.35
• Phonegap 4.1.2-0.22.10
• Eclipse with Android Development Tools (ADT)
• Eclipse Web Tools Platform (WTP)

The ADT extends the capabilities of Eclipse making possible to develop new Android projects faster and still allows to debug applications using the Android SDK tools, and even export .apk files in order to distribute the applications.

Considering that the SQM Mobile App will be developed as a hybrid application using HTML, Javascript and CSS it is required to add to the Eclipse plugin WTP, in order to interpret the code correctly. The WTP project extends the Eclipse platform with tools for developing Web and Java EE applications. It includes source and graphical editors for several of languages, wizards and built-in applications to simplify development, and tools and APIs to support deploying, running, and testing Apps [30].

To install the Phonegap, the following commands in NodeJS console are executed.

```
> npm install -g phonegap
$ phonegap create name-app
$ cd name-app
$ phonegap run android
```

From this procedure 2 projects are created, the project of the App (nameApp) and a project with the libraries Cordova/Phonegap (CordovaLib). These two projects were then imported into Eclipse.

28
An adaptation of the build-path of the project adds a library of CordovaLib “classes.jar” located in CordovaLib/bin. All the code being developed in nameAPP within the directory assets/www.

As Nokia’s servers are only available in internal networks, it was also required to setup a VPN link, and to configure the forwarding routes to use only the VPN connection in the range of private IP addresses 10.0.0.0/8 (range of addresses of the internal servers from Nokia), as the use of Internet (public network) to load some data and application of images is required. The following Figure 3.4 shows the configuration of the VPN.

![Edit VPN network](image)

**Figure 3.4: Configuration of VPN**

### 3.3 Development Process

The SQM Mobile App is composed of six main screens, i.e., **Login**, **Home**, **Map**, **Graph**, **Service** and **Attributes**. Due to its nature, the same development process is applied to all of the App screens.

The JavaScript libraries selected for the development of the hybrid App are **jQuery** and **jQuery Mobile**, that greatly facilitate all programming, namely due to a variety of simplified commands to perform standard procedures and the specific features for mobile development, as well as the use of **AJAX** that
makes possible to easily create requests to servers.

Because of the “same origin” policy, it is not possible to make cross domain AJAX requests. However, with JSONP it is possible to have script tags that load javascript files from other domains, turning this JSONP exception capable of making cross domain requests by dynamically creating a script tag with the necessary URL.

The method can be described as follows: the Server includes data, usually in JSON format, in a function call; during the load of the script, the browser calls that function and passes the loaded data; this situation implies that a third party server needs to know the local javascript function name, which is not practical, but can be circumvented by passing the function name as a parameter to the request URL.

In order to support the JSONP requests, it was necessary to make an “adaptation” to the Nokia servers, by adapting the level of the response format, from the response to be sent within a certain function by a jsonpCallback attribute. This attribute is the same for all requests. The following code snippet is an example of an AJAX request using JSONP.

```javascript
jQuery.ajax({
    url:"U",
    type:"GET",
    dataType:"jsonp", // cross domain
    jsonpCallback: "...", // cross domain
    data:[data block]{
    },
    success:function(response){
    },error:function(request, status, error){
    }});
```

In general all screens have an associated JavaScript file in order to be possible to process the information in isolation and independently from other screens.

The roles of the six main screens, Login, Home, Map, Graph, Service and Attributes are as follows.

**Login Screen:** Page entry, illustrated in Figure 3.5, where the login to the Platform is performed. The prototype App performs the automatic login on servers without the need for the user to insert username or password as those credentials were “hardcoded”, but it was built in a way that will allow in future to enable receiving input data for those fields. The login is performed through an AJAX request, using the login URL.
**Home Screen:** This screen displays all dashboards created for the Services, as illustrated in Figure 3.6 where it is possible to observe the pre-configured Services, as well as a list of the latest 5 “problems” and their location.

**Figure 3.6:** Home Menu

By selecting (touch) on each service, it is possible to then select the next screen such as the
Graph (dependency graph), Attributes or a location Map, as illustrated in Figure 3.7.

As Services are grouped into logical views it is possible to select which of the logical views the user wants to see. For that purpose a “burger menu” has been used in order to list all the logical views available, as illustrated in Figure 3.8.

For the “burger menu” construction the “Ultimate Burger Menu” tool was used because it enables fully customizable jQuery mobile menu resolution to create responsive and animated “dropdown” menus or side-push menus [31].

Graph Screen: The purpose of the Graph screen is to display the dependencies of the selected Service, showing all the other Services that are associated with the one that was selected. The graph with the dependencies of each Service allowed to identify a Root node, and the left and right Children.

Given that the development of a full graph of dependencies represents a very high load in mobile devices, a different solution for the presentation of the data was proposed to Nokia, consisting in splitting the display screen into five columns, each identifying a level of Service hierarchy showing only their leads. This proposal was accepted and it turned possible to display only the necessary data with a much lower development cost and improvements in the loading of the information in the mobile device. Figure 3.9 illustrates an example of the Graph menu, with a Service in the Root node and two Services that depend on the Root.
Figure 3.8: Burger Menu

Figure 3.9: Screen Service Dependencies
Map Screen: The Map view screen, illustrated in Figure 3.10, shows the location of Services. This screen was implemented using the Nokia HERE Maps framework, with data received in GeoJSON format. Once imported the data the framework is responsible for identifying the type of object shown on maps, which can be “areas” (identifying a region where the problem is occurring) and points (identifying Network Elements with problems).

![Map Screen](image)

Figure 3.10: Localization Service Screen

Problem Screen: The Problem screen, illustrated in Figure 3.11 shows the data for a particular problem related to a Service that was previously selected. This screen loads a web page in an iframe, which is already available in the application management provided to customers.

Attributes Screen: The Attributes screen, illustrated in Figure 3.12 shows a table with all the attributes of a given Service, and when there is data available shows the corresponding graph with the historical values of the selected attribute. The graph charts were developed using the AMCHARTS framework, with the following parameters:

```javascript
"type": "serial",
"pathToImages": "https://cdn.amcharts.com/lib/3/images/",
"categoryField": "date",
"dataDateFormat": "YYYY-MM-DD HH:NN:SS",
"categoryAxis": {
  "minPeriod": "ss",
  "parseDates": true
}
```

The AMCHARTS framework receives the data in JSON format so it is not necessary to make any adjustments to the data.
Figure 3.11: Problems of Service

Figure 3.12: Attributes of Service Screen
4 Evaluation

Contents

4.1 Test Environment ......................................................... 39
4.2 Analysis and evaluation of Test Results ............................. 40
4.3 Final Remarks ............................................................. 45
To evaluate the SQM Mobile App, several tests were performed in order to analyze its performance and usability. These tests were carried out under different end-user environments, including stress tests and end-user interactivity. The identified suite of tests is briefly listed in Table 4.1.

<table>
<thead>
<tr>
<th>Type of Test</th>
<th>Objectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stability</td>
<td>To analyze the behavior of the application during a certain period of time (between 1 to 2 weeks)</td>
</tr>
<tr>
<td>Traffic data</td>
<td>Measuring the amount of data between the application and the server, and vice versa</td>
</tr>
<tr>
<td>Response Time</td>
<td>Measuring the changing time between application screens</td>
</tr>
<tr>
<td>Usability</td>
<td>End-user tests to verify usability and if the application is intuitive</td>
</tr>
</tbody>
</table>

4.1 Test Environment

The Stability tests were performed using the SQM Mobile App during 3 weeks, for all the tasks it allows. Tests were performed at least 3 times a day at intervals of (approximately) 3 hours.

The application tests were performed with a Samsung Tablet 4 10.1 with 16 GB of internal storage memory and 1.5 GB of RAM.

The test scenario used for Stability is described in Table 4.2. The tests were performed several times a day because the data produced by Services are constantly changing, making therefore possible to determine the behavior of the application in view of this constant change of data.

<table>
<thead>
<tr>
<th>#</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open and Login in the Application</td>
</tr>
<tr>
<td>2</td>
<td>Choose all available services one by one, and check the list of problems and verification of location of the problems in map</td>
</tr>
<tr>
<td>3</td>
<td>Then also choose all services, one by one, and check the following details:</td>
</tr>
<tr>
<td></td>
<td>1-Check the attributes</td>
</tr>
<tr>
<td></td>
<td>2-Checking the geographic area</td>
</tr>
<tr>
<td></td>
<td>3-Check the dependencies</td>
</tr>
<tr>
<td>4</td>
<td>Close the application</td>
</tr>
</tbody>
</table>

To test the Usability of the application a test scenario described in Table 4.3 was used. A group of ten users tested the application taking into account the predetermined settings. The responses of each user, in relation to the expected behavior of the application, were then collected for analysis.

For Traffic tests and Response time tests, the Chrome browser Debug Tool (in the mobile Tablet) was used. With the Debug Tool it was possible to check each application screens loading time and the
Table 4.3: Test Scenario B

<table>
<thead>
<tr>
<th>#</th>
<th>Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Open the Application</td>
</tr>
<tr>
<td>2</td>
<td>Login in the application</td>
</tr>
<tr>
<td>3</td>
<td>Choose the specific Service</td>
</tr>
<tr>
<td>4</td>
<td>Check the problem list (the upper right side of the screen) Select one of the problems listed for details on that problem</td>
</tr>
<tr>
<td>5</td>
<td>Back to Home</td>
</tr>
<tr>
<td>6</td>
<td>Choose again the same Service</td>
</tr>
<tr>
<td>7</td>
<td>Check the attributes of the selected Service Change a certain attribute and check the change in the graph</td>
</tr>
<tr>
<td>8</td>
<td>Back to Home</td>
</tr>
<tr>
<td>9</td>
<td>Choose again the same Service</td>
</tr>
<tr>
<td>10</td>
<td>Checking the geographic area of the respective Service</td>
</tr>
<tr>
<td>11</td>
<td>Back to Home</td>
</tr>
<tr>
<td>12</td>
<td>Choose again the same Service</td>
</tr>
<tr>
<td>13</td>
<td>Check the dependencies of the selected Service</td>
</tr>
<tr>
<td>14</td>
<td>Close Application</td>
</tr>
</tbody>
</table>

When selecting a Service will have access to an overview of the respective Service. By clicking again on the Service the user can access the details of the respective Service.

In order to test all application scenarios, the test suite was carried out taking under three types of connection:

- Wi-Fi connections in business environment (internal network)
- Wi-Fi connections in residential environment (home network)
- Mobile data connection (4G)

Each screen of the SQM Mobile App was opened 20 times, so the “charging” of each screen was carried out 20 times in a row with both the charging times and transferred data collected for analysis.

4.2 Analysis and evaluation of Test Results

Due to the type of tests performed, the results are better analyzed using “box plots”, making possible to verify the maximum, minimum and mean values for each sample in the same graph and for the different connection types. One of the factors that led to the choice of this type of chart is that outliers are “discarded”. This property is highly valued in the analysis of such data, since the connections to the servers may suffer traffic peaks momentarily, causing values beyond “expected”, which would lead to incorrect analysis of the performance of the application.
4.2.1 Stability

During 3 weeks the App was tested, at least 3 times a day, for all the tasks it allows. The application always responded as expected, showing no data load problems.

4.2.2 Traffic and Response Times

The Traffic and Response time values were collected from the Chrome browser Debug Tool (in the mobile Tablet) measured for each of the screens that were loaded.

From the graph in Figure 4.1 it can be observed for the mean Login time in Internal Network a value of 1.546 seconds, with slightly higher values for a Home Network (1.586 seconds) and a mobile network (1.571 seconds), which agree and are within Nokia’s standard values. The data transferred for this screen was of 1.5 KB.

For the Home screen, the values obtained for the Internal Network (8.851 seconds), for a Home Network (8.868 seconds) and a mobile network (8.337 seconds) were also very similar, as can be observed in the graph of Figure 4.2 and within Nokia’s standard values. The data transferred for this screen was of 330 KB.

The mean time for loading the Problem screen that lists problems of a particular Service are of 4.088 seconds (for the Internal Network), 4.337 seconds (for a Home Network) and 4.183 seconds (in a Mobile Network), as can be observed in the graph of Figure 4.3, and once again, values that are within Nokia’s standard values. The data transferred for this screen was of 340 KB.
Figure 4.2: Loading time of the Home screen

Figure 4.3: Loading time of the Service Problem screen
For the **Dependencies** screen, as can be observed in the graph of Figure 4.4, the mean values were also very similar for the different types of connection, i.e., 1.789 seconds for the Internal Network, 1.828 seconds for the Home network and 1.774 seconds for a mobile network. The data transferred for this screen was on average 1.5 KB.

For the **Map** screen, even rendering the geo-referenced map with the location of “problems” in a Service, as can be observed in the graph of Figure 4.5, also exhibits values within Nokia’s standard, i.e., in Internal Network of 5.908 seconds, in a Home Network of 6.527 seconds and for a Mobile Network of 6.5205 seconds. The data transferred for this screen was on average 481 KB.

The last screen, the **Attributes** that lists the Attributes of a Service and the corresponding historical evolution of their values in a chart, as can be observed in the graph of Figure 4.6, exhibits values for loading also within Nokia’s standard. This values were 4.695 seconds for the Internal Network, 7.098 seconds for a Home Network and 5.995 seconds in a Mobile Network. The data transferred for this screen was on average 4.7 KB.

In order to determine a possible maximum value for the transferred data on a possible usage of the application two tests were also made:

- Check all the attributes of a service
- Check all the attributes of all the services of a logical view

For the first case a value of 381.9 KB was reached, with 691.4 KB for the second case.
Figure 4.5: Loading time of the Service Map screen

Figure 4.6: Loading time of the Attributes screen (with a Chart)
4.2.3 Usability

In relation to Usability tests, each screen of the SQM Mobile App was opened 20 times by the group of 10 users. All users indicated that the application responded as intended and that their perception about loading (charging) times of the displays were within the prescribed standards. Half of the testers were employees of Nokia, and they further indicated that the application was behaving as expected, having additionally the “form” of data presentation similar to other company’s products.

As expected, when users were connected to the Nokia’s internal network the loading times were lower than when connected to a home network or to a mobile network.

4.3 Final Remarks

The SQM Mobile App was developed assuming that it would be used mainly by technicians when they are away of their workplace, or when they are on the move, so the most used connection would be the mobile network.

From the results of the tests, it could be concluded that there was not observed a major impact on the performance of the App, even when using a mobile networks.

Regarding the amount of data that is transferred from/to the SQM Platform, it is also easily supported by existing mobile networks.
Conclusion and Future Work
The telecommunications industry considers OSS an essential mechanism to automate and control businesses operations, and for driving improvements in internal operational performance.

One of the key areas in OSS is the Service Fulfilment and Assurance (SFA), for which Nokia is developing mobile applications able to produce dashboards with key information required by CTOs in the telecommunications industry.

The objectives of the work proposed for this project was to develop a mobile application capable of displaying OSS and Service Fulfilment and Assurance (SFA) data from Network Elements in real-time, and to evaluate the performance of an application developed in a multiplatform environment when compared to the same application but developed in Native mode.

A preliminary research on specific telecom operations process models allowed to consolidate the understanding of how a large telecommunications operator works, and the key requirements for products and services targeted to these types of customers.

Another study gathered information on the diversity of mobile Operating Systems and developments frameworks, as well as on the diverse charting libraries available, allowing a better understanding on the level of difficulty for the design and on the performance to be expected from applications developed using those frameworks and libraries, targeted to the main mobile platforms in the market (Android and iOS).

From the prototype developed and the analysis of its performance in terms of Stability, Usability, Response Times and Traffic in the network, it was possible to conclude that the development using a multiplatform environment such as the Phonegap framework is a good alternative to the development in a Native mode. The developed application responds as desired, matching user expectations and the loading times for all of the screens of the application fell within Nokia’s standard values. As such, it has been considered by Nokia not worth the additional effort of developing the application in Native mode, just for comparison, as the objectives were met using the multiplatform environment.

In addition, the development using a multiplatform environment also lowers the overall cost of development.

Regarding future work, one of the aspects that needs to be addressed is related with the “graphic design” of all the screens of the application, as for the prototype only a basic layout was considered.

This prototype contains a very small number of features but in order to make the more interesting and rich, many other functionalities related with OSS and SFA can be added.

Last but not least, the login method in the prototype application was “hardcoded” and not made in the safest way, and due to the sensitivity of the information that is used, this area should be considered for improvement.
Bibliography


