Dashboards of Service Fulfilment and Assurance for Mobile: Multi-platform versus Native
António Cardoso Jr.

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 Operation 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.

Index Terms—Mobile Application; Android; PhoneGap; iOS; OSS; Assurance; Fulfilment;

1 INTRODUCTION

The telecommunications industry considers OSS not just an essential mechanism to automate and control businesses operations but also the must-have tool for driving improvements in internal operational 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 type 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. 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.

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 capabil-
Fig. 1. Mockup of the Mobile Application Main Menu

ities, 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). [1]. However, under certain scopes, a multi-platform 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 multi-platform 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. 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 objective of the work to be developed is the scope of this thesis, is the analysis of performance a mobile App multiplatform development framework. This mobile App will be based in dashboards. A mockup of the dashboards is illustrated in Figure 1.

2 BACKGROUND

In this section it will be shown all the conducted studies for the development of the project. Initially it is introduced Nokia as a company and also the OSS network architecture. Then I show the investigated Development for Mobile Application Platforms that enable the development of mobile applications. It is also described all the studied development frameworks and tools, like the hybrid apps development frameworks and also the development support tools including charts and maps.

Given that the application to be developed will make a great use of the capabilities to create and analysis charts, and taking into account the cross-platform development, it is a requirement to use charts libraries. The application to be developed will include a map with the location of each service. It is necessary to ensure that maps using API that allows manipulation and customization according to the desired one.

2.1 Nokia Networks Reference Architecture

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.

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. 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 Framework, 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.

2.2 Nokia Networks

Nokia is highly focused on Mobile BroadBand (MBB) and considers fundamental to have Operation 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). 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 [2].

A very wide range of OSSs such as Inventory, Order Management, Provision and Service Activation are included in the Fulfillment process area. Network failures may also be proactively 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 Fulfillment 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) and Service Level Agreements (SLAs).

In Nokia, until recently, the interface to all those services was available as a web interface accessible via a browser using Service Quality Manager (SQM), making easy to an operator to identify if the quality of its basic services was being degraded. Through 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.

Fig. 2. Reporting Workflow

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 Integrated Development Environment (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 (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 Application Program Interfaces (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 [3] [4] [5]. Among the frameworks
that stand out are the Phonegap [6] and the Titanium Mobile [7].

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 [8][9].

2.3.2 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 (UI-WebView 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 [1].

2.4 Development Frameworks
In this section shows the investigated development for mobile application platforms that enable the development of mobile applications.

2.4.1 Phonegap Framework
PhoneGap is a mobile development framework, it enables software programmers to build applications for mobile devices using JavaScript, HTML5, and CSS3, instead of traditional languages such as Objective-C [6] 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).

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.

2.5 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). The
Charting Libraries identified as best for mobile devices and satisfying the heuristics: AMCHART, jsChart, CanvasJS, HighCharts and ZingChart, studied in [10].

2.5.1 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. 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 [11].

2.6 Geo Location
The GeoLocation is a very important point in this application, because by showing the location of the network components in the application identified which allows more effectively the location of problem.

In this section studied the frameworks for the creation of maps used for the application. The API should support the reception of data in a standard format, such as GeoJSON.

2.6.1 GeoJSON
GeoJSON is a particular case of JavaScript Object Notation (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. 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). [12].

2.6.2 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 weigh in deciding the best platform to use [13].

3 System Design and Development
This chapter describes the design and development of the application, 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 System 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.

The Communication Module will be implemented following the concepts of RESTful, with objects of the data model encoded in JSON/JSON with padding (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 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, 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 Representational state transfer (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 IP (meaning that the front-end would be in a DMZ), then a 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 su-systems of the SQM solution, it will not be necessary to define a new data model. The HTTP methods for sending and getting data, in order to have the proper interpretation of the data and therefore adequately display it in the dashboards, will also have to be defined.

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, 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. 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 [10].

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.

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 Process

This section will describe the project organization. Must be applied to development of all application screens, as well as their use.

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 Asynchronous JavaScript and XML (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 Uniform Resource Locator (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.

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.

1) **Login Screen**: Page entry where the login to the Platform is performed. The login is performed through an AJAX request, using the login URL.

2) **Home Screen**: This screen displays all dashboards created for the Services, as illustrated in Figure 4, where it possible to observe the Services, as well as a list of the latest 5 “problems” and their location. By selecting on each service, it is possible to view the properties about service. 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. For this menu construction the “Ultimate Burger Menu”.

3) **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 informa-

![Fig. 4. Home Menu](image-url)
4) **Map Screen:** The Map view screen 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).

5) **Problem Screen:** The Problem screen 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.

6) **Attributes Screen:** The Attributes screen, illustrated in Figure 5, 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.

![Fig. 5. Attributes of Service Screen](image)

**4 Evaluation**

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 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>

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 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.

To test the **Usability** 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 amount of data transferred.

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.

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.

4.1 Analysis 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.

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

The Traffic and Response time values were collected from the Chrome browser Debug Tool measured for each of the screens that were loaded. The following values were obtained (Internal refers to when the tablet is connected network in business environment, Home refers to connected in residential environment and Mobile refers to when the tablet is connected to a mobile data connection):

1) Login: Internal: 1.546 seconds, Home: 1.586 seconds and Mobile is 1.571 seconds. To load this screen were transferred 1.5 KB.
2) Home Screen: Internal: 8.851 seconds, Home: 8.868 seconds and Mobile is 8.337 seconds. To load this screen were transferred 330KB.
3) Service Problem Screen: Internal: 4.088 seconds, Home: 4.337 seconds and Mobile is 4.183 seconds. To load this screen were transferred 340KB.
4) Dependencies Screen: Internal: 1.789 seconds, Home: 1.828 seconds and Mobile is 1.774 seconds. To load this screen were transferred 1.5KB.
5) Map Service Screen: Internal: 5.908 seconds, Home: 6.527 seconds and Mobile is 6.5205 seconds. To load this screen were transferred 481KB.
6) Attributes Screen (Figure 6): Internal: 4.695 seconds, Home: 7.098 seconds and Mobile is 5.995 seconds. To load this screen were transferred 4.7KB.

![Fig. 6. Loading Time of screen Attributes](image)

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.

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.

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.

5 CONCLUSION

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 in this project is 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 whether the performance of an application developed in a multi-platform environment. 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).

Finally, 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.

REFERENCES


