Självständigt arbete på grundnivå

Independent degree project - first cycle

Computer Engineering

A framework for communicating with Android apps from the browser

Karl Lindström
A framework for communicating with Android apps from the browser
Karl Lindström 2014-05-29

MID SWEDEN UNIVERSITY
Avdelningen för Informations- och kommunikationssystem (IKS)

Examiner: Ulf Jennehag, ulf.jennehag@miun.se
Supervisor: Magnus Eriksson, magnus.eriksson@miun.se
Author: Karl Lindström, kali1112@student.miun.se
Degree programme: International Bachelor's Programme in Computer Engineering, 180 credits
Main field of study: Computer Engineering
Semester, year: 1, 2014
Abstract

With the recent growth of the mobile market, companies want to target mobile devices while at the same time keeping product development costs low. One way to do this is to develop web applications, which are accessed from a mobile device’s web browser, instead of native applications. The same web application can then be used on different platforms such as Android and iOS. However, devices such as smart phones and tablets often include cameras and sensors that a web application may want to access, but which are only accessible from native applications. A framework was developed that enables web applications to communicate with native Android applications. Native applications are launched by clicking a link in the browser, and the result produced is made available to the web application through a HTTP POST request or a local web server running on the device. Key characteristics of the framework include ease of extension and the ability to enable secure (SSL) communication if desired. The ZXing Barcode Scanner application was integrated with the framework so that a scanned barcode can be displayed in the browser. Performance measurements were conducted measuring the time taken from clicking a link to start a test application to the result being available in the browser. The mean times measured were between 323 and 394 milliseconds. This indicates that the method used is sufficiently fast to not detract from the user experience. Future work could expand on the measurements or perform a feature and performance comparison with PhoneGap.

Keywords: Android, Web applications, Java.
# Table of Contents

Abstract .............................................................................................................. iii
Foreword ............................................................................................................. vi
Terminology ......................................................................................................... vii

1 Introduction ............................................................................................... 1
1.1 Background and problem motivation ....................................................... 1
1.2 Scope .......................................................................................................... 2
1.3 Detailed problem statement .................................................................... 2
1.4 Outline ....................................................................................................... 3

2 Theory .......................................................................................................... 4
2.1 Android Components .............................................................................. 4
2.2 Android Intents ........................................................................................ 4
2.3 Android Configuration Changes .............................................................. 5
2.4 The Hypertext Transfer Protocol (HTTP) ............................................... 5
2.5 The Hypertext Transfer Protocol Secure (HTTPS) .............................. 6
2.6 The Document Object Model (DOM) ..................................................... 6
2.7 XMLHttpRequest .................................................................................... 6
2.8 JavaScript Object Notation (JSON) ....................................................... 7
2.9 Browser same-origin policy .................................................................... 7
2.10 Cross-Origin Resource Sharing (CORS) .............................................. 7
2.11 PhoneGap ............................................................................................... 8

3 Methodology .............................................................................................. 9
3.1 The measurement environment .............................................................. 9
3.1.1 Tablet specification ........................................................................... 9
3.1.2 Laptop specification ........................................................................... 9
3.2 Measurement specifics .......................................................................... 9
3.3 Performing the measurements ............................................................... 12
3.4 Timing considerations ........................................................................... 12

4 Implementation .......................................................................................... 13
4.1 Starting an application by clicking a link .............................................. 13
4.2 Getting the result into the web page ...................................................... 13
4.3 Designing the URI scheme ................................................................. 14
4.4 Creating the web application ................................................................. 14
4.5 Creating the web application ................................................................. 14
4.6 Defining a common format for returning the result ............................ 16
4.7 Executing a plugin application ............................................................. 17
4.8 Returning a result to the web page ...................................................... 17
4.8.1 The HTTP handler ........................................................................ 18
4.8.2 The HTTPS handler ..................................................................... 18
4.8.3 The web server handler ................................................................. 18
A framework for communicating with Android apps from the browser
Karl Lindström

4.9 Returning a result to the web page ................................................................. 18
4.10 Handling configuration changes ................................................................. 19
4.11 User configuration .................................................................................. 20
4.12 Creating the measurement JavaScript .................................................... 20
4.13 Integrating the ZXing Barcode Scanner ................................................. 20

5 Results ........................................................................................................ 22

6 Conclusions ............................................................................................... 24
  6.1 Future work ............................................................................................. 26
    6.1.1 Expanding the measurements ........................................................... 26
    6.1.2 Investigating security issues with implicit intents ............................ 26
    6.1.3 Comparing performance with PhoneGap ........................................ 27

References ..................................................................................................... 28

Appendix A: The settings screen ................................................................. 31
A framework for communicating with Android apps from the browser
Karl Lindström

Foreword

The main ideas behind this thesis originated at the Dewire company where the thesis work was also performed. I would like to express my gratitude to my supervisor at Dewire, Jonas Bäckström, for patiently answering any questions I had. I would also like to thank my supervisor at the Mid-Sweden University, Magnus Eriksson, for providing feedback that helped to improve this report.

All source code that was developed and the complete measurement data may be accessed by visiting the following URL: https://github.com/kl/thesis-work
## Terminology

**Framework**
A software system that defines a general structure or algorithm needed to complete a certain task. The generic framework code calls business specific code that is supplied by the framework user so that the framework can be tailored for a specific application.

**Plugin**
An independent piece of code that is used to extend the functionality of a program or framework in some way.

**Web application**
A web application as defined in this paper refers to any software application that is accessed with a web browser.

**Native mobile application**
An application that is built using the mobile platform’s standard tools and libraries for application development. On Android a native application is written in Java and packaged as an .apk file.

**Polling**
A technique used by one software component to retrieve a result from a second component by continually querying the second component if the result is available.
1 Introduction

1.1 Background and problem motivation

The mobile computing market has seen an explosive growth in the last decade. The introduction of smartphones in the early 00s enabled consumers to carry a general purpose computer in their pockets for the first time. In the fourth quarter of 2010, more smartphones than PCs were sold worldwide [1]. With the introduction of the Apple iPad, also in 2010, the mobile computing market received another big boost.

With the widespread consumer adoption of mobile devices, companies naturally want to make their services available on the mobile market. Mobile devices such as smartphones and tablets also often enable companies to develop new, value-adding solutions that make use of special device features such as cameras and sensors.

Although the emergence of the mobile market provides many new business opportunities for companies, it also brings new challenges. One example is the cost of developing new software. If a company wants to migrate a major software system that runs in a traditional desktop environment to a mobile environment the company is going to have to invest a considerable amount of time and money in doing so. The costs are further increased by the fact that there are several incompatible competing mobile platforms that have a varying degree of market-share – the two largest being Google's Android platform and Apple's iOS platform. A mobile system developed for one mobile platform will not run on another platform without costly modifications. One potential solution to this problem is to use the web as much as possible, as all modern mobile platforms have good capabilities for integrating web-based content. If a company can develop their system as a web application instead of as a native mobile application, costs can be reduced because the same web application can run in the browser on both Android and iOS, for example.

If a web-based approach is taken, one challenge that needs to be addressed is how to integrate the native mobile platform features with the Web application. For example, if a web application needs to scan a barcode, that functionality could be delegated to a native application on the mobile device that uses a built-in camera to perform the barcode scanning, and then sends the result back to the web application. It is not possible for the web application to scan the barcode directly, because it cannot access the native hardware and software features of the mobile device. In fact, the web application may not even be aware that it is accessed on a mobile device.
1.2 Scope

The practical implementation was done on the Android operating system, version 4.1, using the standard Android web browser.

1.3 Detailed problem statement

A general framework of communication between a web application and a mobile application running on Android will be developed. When creating such a framework the two main things that need to be considered are how the web application should start the Android application and how the Android application should send its result back to the web application. The framework should have the following characteristics:

- The user should be able to start the Android application by clicking a link in the web page served by the Web application.
- The result derived from the Android application should be sent back to the web application.
- There should not be a limit on the number of applications that can be started via the interface.
- It should be easy for the developer to integrate a new Android application with the framework. The developer should only need to write a single Java class that knows how to start the application and how to process the result returned by the application. The remaining functionality, such as how the communication with the web browser should happen, is handled by the framework.

The security aspects of the communication will also be considered. The following points will be investigated:

- When transmitting sensitive data between the web application and the mobile application, the data should be sent over a secure channel (i.e. SSL) in order to minimize the risk that the data is compromised by an attacker.
- It should be optional whether to send the data encrypted or unencrypted.

The following performance measurements will be performed:

- Time taken from the user starting the mobile application to a result being displayed in the user's browser.

In addition to the above-mentioned aspects, a prototype web application will be developed that can be used to test the framework. This web application will gen-
erate HTML pages where the use can click links in order to start third party applications using the framework. The web application will also handle the application results sent by the framework, and display these results in the web page. The following third party applications will be integrated into the framework and accessible from the prototype web application:

- The ZXing Barcode Scanner application. This application can scan barcodes and QR codes of various formats.

### 1.4 Outline

Chapter 2 is a theory chapter that describes background information that pertains mainly to the technical implementation considerations. Chapter 3 is the methodology chapter that describes how the performance measurements were conducted. Chapter 4 explains the design and implementation of Android framework, and to a lesser degree the prototype Web application. Chapter 5 presents the results of the measurements. Finally, Chapter 6 gives the conclusions and suggestions for future work.
2 Theory

2.1 Android Components

An Android application is made up of one or more components. There are four types of components in Android: Activities, Services, Broadcast Receivers and Content Providers [2]. Each component fulfills a specific purpose in the application. For example, the user interface is displayed by an Activity, while a Service can perform long-running background operations. A component is implemented as separate Java class in the application.

2.2 Android Intents

The concept of an Intent is central when it comes to how components are started in Android. An Intent is an object that is used to facilitate message passing between components. When a developer wants to start a new component, an Intent object is used [3]. The Intent object contains enough information for the Android system to identify what component should be started. For example, if an application contains many different screens, each screen is contained in its own Activity. When the user navigates from one screen to another, for example by clicking a button, an Intent to start the second screen’s Activity is created. This is an example of when an Intent is used to start a component within the current application, but Intents are also used to start components in other applications. An example of this is when the user clicks a contact name in the Contacts application and an Activity to call the contact is started in the Phone application.

Intents can be of one of two types: explicit or implicit [4]. An explicit Intent contains an explicit reference to the component that should be started (specified as a package and class), so that there is no ambiguity in what component the Intent will start. This type of Intent is used when the developer knows exactly what component to start, which is the case when starting a local component within the same application. An implicit intent on the other hand does not specify the package and class of the component to start. Instead certain metadata such as an action and category are set on the Intent. When the Intent is processed, the Android system will use the Intent metadata to find all components that are registered to handle Intents that match the specific metadata. Implicit intents are used for example when a user selects a photo in the file explorer application. Then an implicit intent with an action to view a photo is created, and the Android system will find all the applications on the device that are capable of viewing photos. If there is more than one such application the user is prompted to choose one.

Every Android application contains a manifest file where metadata about the application (such as what components it has) is registered. A developer can register intent filters for the application components in the manifest file. An intent filter
specifies the type of intents (if any) that the component should handle. The Android system checks the intent filters for all components for matches when it performs intent resolution for an implicit intent.

2.3 Android Configuration Changes

In Android, a device can change certain fundamental configurations at runtime such as screen orientation, language and available input devices. A change of this type is known as a configuration change [5]. Configuration changes may happen at any time during the application lifecycle. For example, if the user rotates the device a screen orientation configuration change is triggered. When a configuration change happens, the default Android behavior is to restart all Activities so that they can adjust themselves to the new configuration. It is the developer’s responsibility to ensure that the Activity can be safely restarted when a configuration change occurs.

2.4 The Hypertext Transfer Protocol (HTTP)

The Hypertext Transfer Protocol (HTTP) is the standard protocol used on the web for transmitting resources such as HTML pages and images. HTTP is a client-server protocol where a client can instruct a server to send it data, or upload data to the server. When using HTTP, it is always the client that initiates communication with the server.

A HTTP client tells a server what action it should perform by using a protocol defined method. The HTTP/1.1 protocol defines eight methods: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE and CONNECT [6]. Out of these methods the two most commonly used on the web today are GET, which a client uses to request that a server send it a resource, and POST, which a client uses to send data to the server.

An HTTP request consists of a request line, a set of header fields (also known as headers), and a (sometimes optional) request body [7]. The request line contains a method, the URI of the requested resource relative to the domain name of the server and the HTTP protocol version. The headers are used to specify data about the request such as as the type of response the client will accept (e.g. Accept: text/plain) and data and time when the message was sent (e.g. Date: Wed, 16 Dec 1999 08:15:39 GMT). The request body is where the client puts the data that it wants to send to the server. The request body is used with some HTTP methods and not with others. For example, the request body is used when a client sends a POST request to a server (it contains the data that the client sends to the server), but it is not used when a client sends a GET request for a server resource, because GET simply instructs the server to send the client a resource, so no request body is needed.

A server responds to a client request by sending back a HTTP response. The response is structured in the same way as a request with a response line (the equiva-
A framework for communicating with Android apps from the browser
Karl Lindström 2014-05-29

lent of the request line), a set of response headers and possibly a response body. The main difference between a request and a response is that the response line contains a status code instead of a method and server domain name. The status code indicates the status of the request, for example, the status code 200 means that the request was successfully served, while the status code 500 means that the request could not be served because of an internal server error.

2.5 The Hypertext Transfer Protocol Secure (HTTPS)
HTTPS is a version of the HTTP protocol where all data sent between the client and server is encrypted using SSL/TLS [8]. When a browser initiates a HTTPS connection with a domain (for example www.google.com), it has to verify that the responding server is the real server of the domain, and not an impostor server that is used to initiate a man-in-the-middle attack. In order to perform the validation a HTTPS certificate is used that is sent by the server to the client. The certificate is cryptographically signed by a trusted third-party organization, so that it is verifiable by the browser. A certificate may also be self signed which means that it was signed without the use of a trusted third-party. If this is the case the HTTPS connection is encrypted but it is vulnerable to man-in-the-middle attacks because it’s not possible for the browser to verify that the certificate is valid.

2.6 The Document Object Model (DOM)
The Document Object Model (DOM) is “a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents” [9]. HTML, XHTML and XML documents support the DOM. In a browser, JavaScript can be used to manipulate the DOM of a HTML page. This enables JavaScript to dynamically change the structure of the HTML document (and therefore also what the user sees on the screen).

2.7 XMLHttpRequest
XMLHttpRequest is a JavaScript object that enables a JavaScript program to asynchronously perform HTTP requests in the background, without reloading the web page [10]. It is supported in all modern browsers. The HTTP request performed is often a GET request for some server data, which is later dynamically added to the web page through the DOM by JavaScript. The data format used by the server when serving a request from a XMLHttpRequest object is usually XML or Javascript Object Notation (JSON).

2.8 JavaScript Object Notation (JSON)
JavaScript Object Notation (JSON) is a data interchange format that is lightweight and language independent. JSON data is stored as a plain text, human-readable, key-value pair structure that supports a few basic data types including
numbers, strings, booleans and arrays [11]. JSON is often used as an alternative to XML as a data interchange format on the web.

2.9 Browser same-origin policy

The same-origin policy is a security policy that is implemented in all browsers, with slight differences among different browsers. The basic idea is to permit scripts from a certain origin to have access to all resources that stem from that origin, while disallowing access to resources from other origins [12]. For example, if two HTML documents come from the same origin, JavaScript code running in one document can access the DOM of the other document, but if the origins are different the browser will not allow this to happen. The origin is defined as the triple {protocol, host, port}. All three parts must be the same for the same-origin check to go through. The same-origin policy also applies to asynchronous HTTP requests (i.e. sent from the browser’s XMLHttpRequest object), so a script is not allowed to use the XMLHttpRequest object to access data from a different origin.

2.10 Cross-Origin Resource Sharing (CORS)

Cross-Origin Resource Sharing (CORS) is a specification that, when supported by the browser and the server, can be used to relax the same-origin policy for asynchronous JavaScript HTTP requests [13]. Using CORS a server can specify what domains, if any, should be allowed to make JavaScript HTTP requests to the server. CORS works by defining new HTTP headers. When a HTTP request is send from JavaScript, the Origin header is added and set to the value of the origin of the requesting domain. For example, if a request is made from a script that has the origin http://www.example.com to the http://www.test.com origin, the origin header on the request would be Origin: http://www.example.com. If the server wanted to allow the request, it would send back a response with the header Access-Control-Allow-Origin: http://www.example.com. If the server did not set the Access-Control-Allow-Origin header appropriately the response would not be delivered to the JavaScript program. If a server wants to allow all JavaScript HTTP requests from all domains, it can set the Access-Control-Allow-Origin to *.

2.11 PhoneGap

PhoneGap is a technology that enables developers to create mobile applications for various platforms using web technologies such as JavaScript, HTML and CSS instead of using the native language of the platform [14]. The same PhoneGap application can run on Android, iOS, Windows Phone and other mobile platforms. PhoneGap accomplishes this by running the application as a web application in the browser on the mobile device. The way that PhoneGap differs from the traditional web application model, where a server on the internet is used to deliver the web application to the browser, is that a PhoneGap application is packaged
A framework for communicating with Android apps from the browser
Karl Lindström

and installed as a native application that consists of a WebView (browser window) where the application specific HTML, JavaScript and CSS code is executed [15].

PhoneGap applications can access native device APIs (such as the camera, accelerometer and geolocation) using plugins [16]. A PhoneGap plugin is written in the platform language (for example Java on Android) and is used to create an API that can be used by the JavaScript running in the WebView to access native functionality. The PhoneGap SDK includes built-in plugins that provide access to the camera, file system, accelerometer and more. Users can also write their own plugins to enable additional functionality.
3 Methodology

The time it took from the user clicking the link in the browser to the result being displayed in the browser was measured. In order to perform this measurement, a separate Android application was developed that returns a pre-defined result as soon as it is started. Using this test application for the measurements ensures that the tests reflect the time it takes to start up an application and return a result, and not the time it takes to use the application.

3.1 The measurement environment

When conducting the measurements the web application was served from a web server running on a laptop, and the Android code was running on a Samsung Galaxy Note 10.1 GT-N8010 tablet. All communication between the web server and the tablet was conducted over 802.11.n Wi-Fi on a local network. The Web application was written in Ruby 2.1.1 [17] and the web server used was thin [18].

3.1.1 Tablet specification

- Android 4.1.1
- 2 GB RAM
- Quad-core 1.4 GHz CPU

3.1.2 Laptop specification

- Ubuntu 12.04 64 bit
- 2 GB RAM
- Intel Core 2 Duo T5450 Dual-core 1.66 GHz CPU

3.2 Measurement specifics

The system is capable of returning the result to the browser in two different ways. The first way is to send the result with a HTTP POST to a web application, and then the browser retrieves the result by polling the web application with JavaScript using HTTP GET. This process is illustrated in Figure 3.1.
The second way is that the Controller Application starts up an internal Java-based web server when it receives the result from the Test Application. The web browser then polls this internal web server for the result instead of polling a web application. This is illustrated in Figure 3.2.

In the two scenarios above, the Controller Application is started when the user clicks the link in the web browser. The Controller Application then parses the link and decides what application should be started (the Test Application in this case) and how the result should be returned to the browser. If the application that should be started when the link is clicked is not recognized by the Controller Application, a system wide Intent can be sent out that attempts to find a Plugin Application that is capable of starting the application specified in the link and return the result to the Controller Application. The difference between the built-in approach and the Plugin Application approach is in how the result is retrieved and returned to the Controller Application. Once the Controller Application has received the result, the process of sending the result to the browser is the same. This is illustrated in Figures 3.3. and Figure 3.4.
A framework for communicating with Android apps from the browser
Karl Lindström

2014-05-29

Figure 3.3 - Using a Plugin Application with Web Application polling
When the internal approach is used, two different Android applications are started, whereas three different applications are started when the system approach is used.

### 3.3 Performing the measurements

The time taken from the first step to the final step in figures 3.1 - 3.4 were measured one thousand times for each of the four different approaches. The measurements were performed automatically by a JavaScript program running in the web browser on the Android device. When a link that is used to start the measurement is clicked the JavaScript program records the current timestamp. The script then begins to poll either the built-in web server or the web application for the result. When the result is successfully retrieved, the JavaScript program takes another timestamp and calculates the time it took from the link being clicked to the result being received in the browser. After the JavaScript program records the time taken, it automatically starts a new measurement by simulating a click on the measurement link by assigning the window.location.href property to the link’s URL. This process is repeated the specified amount of times. When all measurements have been taken, the JavaScript sends the result to the Web application where it is saved as a file that can later be analyzed. Refer to the implementation chapter for details on how the JavaScript program is implemented.
3.4 **Timing considerations**

A polling frequency of 10 ms was chosen when performing the measurements. Because polling is employed in order to get the result into the browser, the polling frequency has an effect on the time taken as measured by the JavaScript program. The higher the polling frequency is the more accurate the measurements are. In the worst case scenario the result arrives at the polling destination immediately after a poll has occurred. Then the time measured will be increased by how long the polling time is.
4 Implementation

The purpose of the system is to enable a way to start a native Android application when a hyperlink is clicked in a web page in the Android browser. The application should then execute, and after it is finished it should be able to send its result back to the web page in the browser.

4.1 Starting an application by clicking a link

Executing an application when a hyperlink is clicked in the browser is simple in Android. It is possible to register that an application should handle a custom URI scheme by adding an `<intent-filter>` tag to the manifest file. Listing 4.1 shows how this is accomplished.

```xml
<intent-filter>
  <action android:name="android.intent.action.VIEW"/>
  <category android:name="android.intent.category.DEFAULT"/>
  <category android:name="android.intent.category.BROWSABLE"/>
  <data android:scheme="app"/>
</intent-filter>
```

Listing 4.1 - Creating an intent filter to start an application when a link it clicked

The `<data android:scheme>` tag specifies the scheme of the URIs that the application should handle. In this case the scheme is set to “app”, which means that all URIs that start with “app” will be handled by the application.

4.2 Getting the result into the web page

The final destination of the result data is the web page where the user clicked the link to start the application. In general, there are two ways to update the content of a web page: either a full page refresh is performed that downloads a new version of the web page, or new content is dynamically added via the DOM. If the latter approach is used, the new data can be fetched from the server using the XMLHttpRequest object. This approach was used developing the web page’s JavaScript code. When the user clicks the link the JavaScript starts to poll an URL for the result and when the result is retrieved it is displayed in the page by adding it to the DOM.
4.3 Designing the URI scheme

As outlined in chapter one, one of the key requirements was to make the system generic enough so that it can adapt to changes easily.

The two parts of the system that are likely to change are what application should be started when the link is clicked in the browser, and how the result should be sent back to the browser. It cannot be assumed that the same application should always be started when the link is clicked. Likewise, it cannot be assumed that the method of sending the result back to the browser should be the same. For example, it may be desirable to send the result over a secure channel (i.e. HTTPS) in some scenarios, while in others it may be acceptable to send the data unencrypted. Therefore, the information on what application to start, and how the result should be sent to the browser, must be encoded in the URI.

The only part of the URI that must be present for the application to get started is the scheme. So if the scheme is set to app, then the link URL must begin with app://. Given this, a simple URI format was designed that allows for specifying the application and the way to send the data back to the browser:

app://APPLICATION/HANDLER

The format consists of the scheme followed by a string used to identify the application to start, followed by a forward slash, followed by the method to send the result back to the browser (referred to as the handler).

4.4 Creating the web application

A web application was created that is used to generate the web pages that contain the application links. This web application is also used as a destination by the Android side code to post the result to, and by the browser JavaScript to poll for the result. The web application is written in Ruby using the Sinatra [19] framework. The web application is accessed through a simple test URL that will generate a HTML page that contains the application links. The Android side code can send the result to the web application with a HTTP POST request to a separate URL as soon as the result is available.

4.5 Creating the web application

When the application link is clicked the Android system scans the manifests of all installed applications for intent filters that match the URI. If a match is found the corresponding application is started with an Intent that the application can use to access the URI of the link that was clicked. Because any application should be able to be started when the link is clicked, including applications that are not aware of the special URI format or how to send the result back to the browser, an intermediary application, called the Controller Application, was developed. The Controller Application is the application that is started by the Android system.
when the link is clicked. It is responsible for parsing the Intent URI and starting the application associated with the application name part of the URI. It is also responsible for making the result available to the web page.

The Controller Application is programmed so that it knows about a set of pre-defined applications that it is able to start. This is done by creating a Java class, called a plugin, that knows how to start a specific application and how to parse the result that the application returns to a common format that is then made available to the web page. For example, if the goal is to start a barcode scanning application, the Controller Application would include a plugin class that knows how to start the barcode application, and parse the result that is returned by the barcode scanning application to a common format that is made available to the web page.

The framework also supports system plugins that are installed as separate applications. The Controller Application will attempt to find one of these system plugins if it fails to find an internal plugin to handle the application. The plugin resolution process is summarized in Figure 4.3.

![Diagram of plugin resolution process]

**Figure 4.3 - The plugin resolution process**

Allowing plugins to be installed as separate applications allows for increased flexibility because system plugins are not tied to the Controller Application. It is pos-
sible to install the Controller Application on a device, and then later add new plugins without having to recompile the Controller Application or even have access to its source code.

4.6 Defining a common format for returning the result

In order to keep the system as flexible as possible, the only constraint on what result a plugin is allowed to return is that the result must be formatted as JSON. JSON is used because it is a flexible key-value data structure so each plugin is able to store whatever key-value pairs that are suitable for the specific plugin. For example, a barcode scanner plugin could return a JSON object with a key called code and the value being the scanned barcode. Another reason for using JSON is that it is easily parsable by the web page’s JavaScript that will display the result.

4.7 Executing a plugin application

Once a plugin has been resolved as described in 4.5, the Controller Application asks the plugin for the Intent that should be used to start the application. The application specified by the Intent is then started with the startActivityForResult method. The application will then execute and return its result as an Intent to the Controller Application, which then delegates the result Intent object to the plugin class. The plugin then parses the the result into a JSONObject [20], which is then returned back to the Controller Application. This process is shown in Figure 4.4.

![Figure 4.4 - Plugin execution sequence diagram](image-url)
4.8 Returning a result to the web page

Just like new plugins can be added to the Controller Application to start new applications, new handlers can be added to enable new ways of returning the result to the web page. The framework includes three handlers by default: A HTTP handler that uses a HTTP POST request to send the result to a server, a HTTPS handler that does the same but over a HTTPS connection, and a handler that starts an internal web server that the web page JavaScript can poll for the result. All handlers expect that the result they should handle is supplied in the form of a JSONObject. The JSONObject is supplied by the Controller Application, which in turn got it from the plugin.

4.8.1 The HTTP handler

The HTTP handler takes the JSONObject, converts it to a text representation, and posts it to a server destination that is identified by a the triple {IP address, port number, path}. What server the HTTP handler should post to is configurable by the user, but the most likely destination is to post to the same server that served the web page that the application was started from. The HTTP handler is implemented using a HttpURLConnection object and the HTTP POST request is done in a separate thread.

4.8.2 The HTTPS handler

The HTTPS handler work in the same way as the HTTP handler, expect that it uses HTTPS instead of HTTP. In the implementation, the HTTP handler and the HTTPS handler classes are both subclasses of the abstract class PostHandler that contains the code to perform the POST request. The implementation of the HTTPS handler is suitable for testing but not for production because it disables the SSL certificate validation check. This was done so that the HTTPS functionality could be tested with only a self-signed HTTPS certificate.

4.8.3 The web server handler

The web server handler starts an internal web server based on the NanoHTTPD Java web server. The server then listens for the HTTP GET request sent by the JavaScript and returns the JSON result. When the web server is used all communication between the web browser and the Android application happens locally on the Android device. Even though the destination of the XMLHttpRequest is a server that is running on the same machine that the browser is running on (the Android device), it must still obey the rules of CORS. By default, CORS only allows XMLHttpRequests to be sent to the origin server (the server where the web page came from). For this reason the Access-Control-Allow-Origin header needs to set to * so that all requests are allowed. It is not possible to make the access control more specific (i.e. allowing only certain domains) because the application does not know from which domain the request will come from.
4.9 Returning a result to the web page

The framework includes error handling for some common types of error that may occur. For example, if the plugin that is specified in the application link is not found, a jsonObject containing an error message is returned to the handler (if the handler is not found, the framework terminates). Another example is if the user cancels the result application by pressing the back button on the device. Then a message informing the handler that the user cancelled the application is returned.

Errors may also happen when the result is processed by the handler. If for example the server that the HTTP post handler posts to is not responding, the Controller Application will display a dialog with an error message to the user. When the user dismisses the dialog the framework terminates.

4.10 Handling configuration changes

Even though the Controller Application does not have its own user interface, it must still have custom code to handle configuration changes. To see why this is necessary, consider the following scenario:

1. The user starts a barcode scanning application by clicking the application link in the web page
2. The barcode scanning application scans a barcode and returns the result to the plugin
3. The result is delivered to the HTTP post handler
4. The HTTP post handler initiates a TCP connection with the server
5. Due to network lag it takes three seconds for the TCP connection to be established
6. The handler sends the result to the server

At step 5, the Controller Application (which is the currently active application) is waiting for the handler to report back that it has successfully delivered the result, after which the Controller Application will perform cleanup and then exit. When this is happening, the Controller Application is displaying a dialog box that informs the user that the HTTP post handler is in the process of sending the result to the server. If, when this is happening, the user were to rotate the device a configuration change would be triggered and the Android system would restart the Controller Application. A configuration change could also be triggered if the user rotates the device while using the barcode scanning application started in step 1. Then when the result is returned to the plugin, the Android system would immediately restart the Controller Application because the screen orientation changed.
In order to correctly handle configuration changes the application state needs to be saved and restored when the application is restarted by the Android system. The Controller Application keeps track of the selected plugin and handler by keeping references to them in fields. The plugin and the handler communicate with the Controller Application through a publish-subscribe interface. For example, when the plugin wants to return the JSONObject result of the Controller Application it will do so by notifying the Controller Application with the onPluginResult() callback. For this reason the plugin and handler objects need to be saved and restored on a configuration change. When the Controller Activity is restarted it also needs to add itself as a listener to the plugin and handler objects again.

The strategy used to save the state is to create a fragment called StateFragment which has the setRetainedInstance property set to true. This ensures that the Android system will not recreate the fragment on a configuration change. The selected plugin and handler objects are then assigned to fields in the state fragment. When the Controller Application is restarted due to a configuration change, it will retrieve the state fragment using the fragment manager and get the plugin and handler objects to from state fragment.

4.11 User configuration

The system has many user configurable settings such as the IP address and port number that different handlers should use, whether system plugins should be enabled or disabled, and protocol details such as the JSON key names that are used when errors are reported. The user can configure these settings using a settings fragment that can be started as a normal application from the application launcher. See Appendix A for a screenshot of the settings screen.

4.12 Creating the measurement JavaScript

The JavaScript program used for the performance measurements measures the time taken from the link being clicked to the result being available to the web page. The measuring process is initiated by clicking a link that has an onClickListener set to the function that starts the measurement. This function takes the link that is used to start the application, the number of times the measurement should be performed, and the delay in milliseconds between measurements. The function then starts the Android application by setting the window.location.href attribute to the link URL. It then starts polling either the web application or the local server depending on the type of measurement performed.

4.13 Integrating the ZXing Barcode Scanner

The ZXing Barcode Scanner application [23] can be used to scan barcodes and QR codes using the built-in camera on the Android device. ZXing can be started by other applications by sending an Intent with the action set to com.google.zxing.client.android.SCAN. A plugin class was developed that creates the Intent that
is used by the Controller Application to start ZXing. Scanning parameters can be set by using the putExtra(key, value) methods on the intent object. Listing 4.2 shows the parameters that are set by the plugin.

```java
intent.putExtra("SCAN_FORMATS", "UPC_A,UPC_E,EAN_8," +
    "EAN_13,CODE_39,CODE_93," +
    "CODE_128,ITF,RSS_14,RSS_EXPANDED");

intent.putExtra("SCAN_WIDTH", 500);

intent.putExtra("SCAN_HEIGHT", 50);

intent.putExtra("SAVE_HISTORY", false);
```

**Listing 4.2 - ZXing Intent parameters**

The list of formats used with the SCAN_FORMATS key is given so that ZXing will scan barcodes, and not other code types such as QR codes.

After the user has scanned a barcode, ZXing returns the result as an Intent that is delivered to the Controller Application, that then forwards it to the barcode scanner plugin where the Intent is parsed to a JSONObject. The barcode is extracted from the Intent by calling getStringExtra("SCAN_RESULT") on the Intent object.
5 Results

The results of measuring the time taken from the user clicking the link in the web browser to the result being received in the browser are presented in Table 5.1. The measurements were performed one thousand times for each combination. All units are in milliseconds.

Table 5.1 - Measurement results

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>Minimum</th>
<th>Maximum</th>
<th>95 % Confidence Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Built-in web app</strong></td>
<td>323.23</td>
<td>79.7</td>
<td>153</td>
<td>627</td>
<td>±4.94</td>
</tr>
<tr>
<td><strong>Built-in web server</strong></td>
<td>304.16</td>
<td>78.9</td>
<td>151</td>
<td>634</td>
<td>±4.89</td>
</tr>
<tr>
<td><strong>System web app</strong></td>
<td>394.87</td>
<td>95.9</td>
<td>197</td>
<td>914</td>
<td>±5.94</td>
</tr>
<tr>
<td><strong>System web server</strong></td>
<td>382.12</td>
<td>90.9</td>
<td>187</td>
<td>676</td>
<td>±5.63</td>
</tr>
</tbody>
</table>

Figures 5.3 presents the percentile plots for the different approaches:
Figure 5.3 – Measurement time percentile plot
6 Conclusions

The overall goal was to create a system that can start a native Android application when a link is clicked in the web browser. The application should then produce a result, such as for example scanning a barcode, that the system should be able to return to the web page. Some of the main requirements were to make the system flexible enough so that it is easy to change and extend, to make it sufficiently fast so that it does not negatively affect the user experience, and to provide communication over a secure connection (SSL) if desired.

As a proof-of-concept, the ZXing Barcode Scanner application was integrated with the framework. It enables a user to scan a barcode using the Android device’s camera, and have the resulting barcode displayed in the web page.

To test the speed of the system, the time taken from the link being clicked to the result being available in to web page was measured. All measurements were performed on a local network over WiFi. Measurements for four different combinations of methods to start the application and return the result to the web page were performed one thousand times each. The combinations were: built-in HTTP, built-in server, system HTTP and system server.

The measurements show that the mean time for the different combinations was between 323 and 382 milliseconds. This is sufficiently fast to not detract from the user experience. However, given these results, a few thing should be taken into consideration:

1. The measurements were conducted over a local network which has less delay than if the communication had occurred over the Internet. This is only a factor if the HTTP post handlers are used, as the server handler does not need network access.

2. A test application was executed that returned a result immediately after starting. Since this was a minimal application, it is likely that a real application will take a longer time to start than the test application.

For the different combinations, the maximum time taken was on average about four times longer than the minimum time. It it likely that the time taken depends on the current state of the Android system when the measurements are started, i.e. if the system was serving a background service the measurements would take longer to complete than if the system was idle.

It is interesting to note that when the built-in plugin was used used the server handler is consistently faster than the HTTP handler, but when the system plugin
A framework for communicating with Android apps from the browser
Karl Lindström

was used the HTTP handler was faster than the system handler. More detailed measurements and tests would have to be conducted to find out why this happens.

System flexibility was achieved by allowing the user to configure many system parameters through the settings screen. Flexibility was also achieved on the code level by extracting the code that knows how to start applications and parse the result (the plugins) and the code that knows how to make the result available to the browser (the handlers) behind separate interfaces. This makes it possible to add new plugins and handlers by writing a single Java class.

The security issues were addressed with the HTTPS post handler. The other handlers are not considered to be secure as the HTTP handler sends the data in plain text, and it is possible for a malicious third party application to read the result from the server handler before it is read by the web page.

Another security concern is that it is not possible for the framework to verify that the web page that started the application is not malicious. An attacking web page could use a link of the format that the framework expects, and if the user clicks the link the native application will be started. This attack would not be automatic in that the user would still have to use the native application (for example the Barcode Scanner) in order to produce a result. This attack is similar to many phishing attacks in that the best defense against it is user awareness. If the user is aware that the web site they are visiting should not be able to start native applications, then they will immediately recognize the attack and navigate away from the web site. Server side checks should also be in place to help mitigate such attacks (for example using sessions to verify user actions). It’s important to weigh the costs against the benefits before employing a solution using the approach described in this paper in a production environment. If the framework is employed on devices with unrestricted access to the Internet and is used by a relatively naïve user base it could very well lead to a security risk.

PhoneGap could be used to enable much of the same functionality that the framework provides. For example, it would be possible to integrate the ZXing Barcode Scanner application by writing a PhoneGap plugin for it. If this approach was used it would be analogous to using the framework barcode plugin with the web server handler, because the scanned barcode would not have to be sent from the Android device to an external server. The main difference between the framework that was developed and PhoneGap is that a PhoneGap application is packaged and installed like a native application. Depending on the situation, there are both advantages and disadvantages of having the web application act like a native application. Consider a complex existing web application that is currently accessed by mobile users via the browser. If the goal is to add native functionality to the web application, it could either use the framework or be converted to a PhoneGap application. Depending on the circumstances, it may take longer time and be more expensive to convert the application to a PhoneGap application than it would be to integrate it with the framework. Another scenario is if a company provides
many different mobile applications that all need to access native functionality of some sort. Then every application would need to be converted to a PhoneGap application, whereas if the framework was used, each application could simply generate a link to start the plugin that it needed. On the other hand, when using a PhoneGap application it is no longer necessary to have a domain where the web application is accessed from and the application can be distributed through the mobile platform’s normal distribution channel (for example the Google Play app store on Android). Another advantage of this approach is that from a user perspective only a single “app” needs to be installed. If the framework is used, the user must install the framework application on their device before the use the web application. Users may be reluctant to install a separate application that has not visible functionality. This may not be an issue in for example a corporate environment where employees are given mobile devices where an administrator has already installed a preexisting set of applications.

6.1 Future work

6.1.1 Expanding the measurements

The focus of the measurements were to find out the time it takes to use the system from start to finish with a test application that immediately returns a result. Future work could investigate the time it takes to start real applications using the framework. One way to measure this accurately would be to record the time when the link is clicked with JavaScript in the browser, and then take the time again in the real application’s onCreate method. In order to add the timing code to the real application, it could be decompiled and modified using a tool such as apktool [24].

6.1.2 Investigating security issues with implicit intents

The use of implicit intents when starting components may lead to security vulnerabilities [25]. The problem is that the Android system will find all applications with matching intent filters that are installed on the device. This includes potentially malicious third-party applications that could be started when the implicit intent is resolved. If more than one application is found when resolving an implicit intent, the user will be prompted to choose one by the Android system. Therefore, a malicious application could potentially inject itself in the framework by declaring an intent filter that matches a plugin intent. This problem can be avoided if system plugins are disabled (which can be done in the settings). The severity of the security risk is also dependent on the device policy when it comes to installing new applications. For example, if the device is locked down so that new applications may not be installed without the explicit approval of an administrator, using implicit intents may be acceptable.

Future work may investigate additional ways of improving the security of the system plugin resolution. For example, it may be possible to develop a scheme where a passphrase is used to prevent malicious applications from responding to the system plugin intent resolution. If a compile-time defined passphrase is added as a
category attribute to the system plugin intent, only applications that have the same
category element in their intent filters would be resolved by the Android system.
It is then assumed that only trusted applications know the passphrase. Additional
points of interest if such a scheme were to be developed are what happens if the
passphrase is compromised, and if it is possible to change the passphrase without
recompiling the framework.

6.1.3 Comparing performance with PhoneGap

Because the framework is similar in functionality to PhoneGap, it would be inter-
esting to develop a PhoneGap application and use it to conduct the same perform-
ance measurements that were done for the framework. A test PhoneGap plugin,
similar to the test plugin used in the performance measurements, could be de-
veloped that immediately returned a result. Then the time from the user clicking a
link in the PhoneGap application to the the result being displayed could be meas-
ured.
References


A framework for communicating with Android apps from the browser
Karl Lindström


    http://json.org/

    http://code.google.com/p/browsersec/wiki/Part2#Same-origin_policy

    http://www.w3.org/TR/cors/#introduction

[14] phonegap.com, “FAQs”
    http://phonegap.com/about/faq/
    Retrieved 2014-05-16

    http://docs.phonegap.com/en/edge/guide_overview_index.md.html#Overview
    Retrieved 2014-05-16

[16] docs.phonegap.com, “Plugin Development Guide”
    http://docs.phonegap.com/en/edge/guide_hybrid_plugins_index.md.html#Plugin%20Development%20Guide
    Retrieved 2014-05-16

[17] ruby-lang.org, “Ruby Programming Language”
    https://www.ruby-lang.org

[18] macournoyer.com, “Thin - yet another web server”
    http://code.macournoyer.com/thin/

    http://www.sinatrarb.com

A framework for communicating with Android apps from the browser
Karl Lindström


[22] github.com/NanoHttpd, “What is nanohttpd”
https://github.com/NanoHttpd/nanohttpd

[23] github.com/zxing, “README.md”
https://github.com/zxing/zxing


work Management (IM2013): Short Paper, 2013
Appendix A: The settings screen

![Settings Screen]

**HTTP SERVER SETTINGS**
- **IP**
  - The IP address of the HTTP server
- **Port**
  - The HTTP server port
- **Post path**
  - The server path (without a leading '/') that should be posted to

**HTTPS SERVER SETTINGS**
- **IP**
  - The IP of the HTTPS server
- **Port**
  - The port of the HTTPS server
- **Post path**
  - The server path (without a leading '/') that should be posted to

**LOCAL SERVER SETTINGS**
- **Port**
  - The local server port
- **Timeout**
  - How many seconds the local server should wait for a request before terminating

**PROTOCOL SETTINGS**
- **Plugin type key**
  - The JSON key with the name of the plugin
- **User cancel key**
  - The JSON key used when if the user cancels the plugin
- **User cancel value**
  - The JSON value that is used if the user cancels
- **Plugin not found key**
  - The JSON key used when if the plugin is not found
- **Plugin not found value**
  - The JSON value that is used if the plugin is not found

**OTHER**
- **System Plugins**
  - If system plugins should be resolved
- **System plugin Intent category**
  - The Intent category that is used for system plugin resolution