Technical University of Crete, Greece

Department of Electronic and Computer Engineering

Implementation of a Client-Server Mashup System using the Android Platform

Georgios Komninos

Thesis Committee

Professor Minos Garofalakis

Professor Stavros Christodoulakis

Assistant Professor Antonios Deligiannakis

Date

Abstract

In today's world, Web services offer great opportunities so that applications of different kinds may interact and produce modern and more sophisticated products. The mashup programming paradigm is a perfect example of how to combine such services in order to achieve a dynamic interaction with ease and flexibility. Moreover, mobile devices have invaded our everyday life more than ever, offering attractive features such as larger displays, powerful CPUs, GPS sensors and fast internet access using 4G networks. This thesis suggests a methodology on how to take advantage of the mashup programming paradigm using the conveniences of a mobile device. Following the mashup concept, we propose a simple way to access data from different sources in a mobile device and offer the ability to use that data in order to produce sophisticated mobile applications.

Contents

1 Introduction **1**
1.1 Thesis contribution 1
1.2 Thesis Outline 2

2 Background and Related Work **3**
2.1 Mashups 3
2.2 Mobile Applications[2] 4
2.3 Related Work 5

3 Mashup Platforms **7**
3.1 Current State 7
3.2 Platform Selection 8
3.3 Apatar's Architecture 9
3.3.1 Core Engine 9
3.3.2 Connectors 11
3.3.3 GUI and Data Representation Layer 11
3.3.4 Extensibility 11

4 Mobile Operating Systems **13**
4.1 Current State 13
4.2 Mobile OS Selection 15
4.3 Android Application Fundamentals[7] 17
4.3.1 Application Components 19
4.3.2 Activating Components 22
4.3.3 The Manifest File 23
4.3.4 Application Resources 23

5 Apatar Android Version **25**
5.1 Client 26
5.2 Server 28
5.3 Functionality 31

6 Demonstration **34**

7 Conclusions and Future Work **42**

References **44**

List of Figures

3.1 Apatar's architecture 10
4.1 Android architecture 17
4.2 Android application components 20
5.1 Client-server interaction 25
5.2 Application homepage 26
5.3 Connector selection screen 27
5.4 Example of connector properties input-MySQL connector . . . 28
5.5 Record source selection screen 29
5.6 MySQL connector added 30
5.7 List of operations screen 32
5.8 Enter join operation details screen 33
5.9 Qualifying attributes screen 34
6.1 The mashup application 35
6.2 Application homescreen 36
6.3 Route to next destination 37
6.4 All destinations 38
6.5 Android MySql connection 39
6.6 Selecting a destination 40
6.7 Symbol explanation 41

1 Introduction

Nowadays, the continuous growth of the Internet and related technologies, has made possible the interaction of heterogeneous data and different existing Web services. That interaction maximizes efficiency and extends the limit of possible service combinations. Currently, a single application may well perform the same workload that in the past required manual interaction of many different applications and do it rapidly, automatically and therefore more effectively.

The mashup paradigm also favors the continuous development of new Web services to cover the ever more demanding needs of Web users. These new Web services are, in reality, the combinations of already existing Web services that interact in order to produce their results quickly and straightforwardly. While the required data was already available since long ago, what was missing was a tool that would allow developers to access the data and make it work altogether. The mashup approach offers the opportunity to compose data quickly and with as little effort as possible, even by end-users with either very limited or even no programming expertise.

On the other hand, another ever-growing trend of today is the use of mobile devices. Smartphones and tablets with their new impressive features are used more and more to the point that they can almost substitute the use of PCs or laptops, especially in the everyday use of a non-expert user and the trivial tasks that they need to perform. The current range of available mobile applications is rich and fulfills the needs of an average user, but having the ability to use the mashup programming paradigm in mobile applications would greatly increase the variety.

1.1 Thesis contribution

In this thesis we attempt to offer mobile application developers the opportunity to take advantage of the privileges of mashups. We develop an application that makes it possible to implement mashups on Android mobile

devices, using Apatar platform for data integration. This way any developer can create through their mobile device or emulator the datamaps that are required for the application they have in mind. Then, using a Java class that we provide they can use the output files of the application in order to implement an application of their desire. That way, we believe that we can expand the already existing capabilities of mobile devices and offer mobile users more sophisticated applications.

1.2 Thesis Outline

The remainder of this thesis is organized as follows. In Chapter 2, we present the background and related work to our project. We introduce the mashup prorgamming paradigm as well as an introduction to moblie applications. Chapter 3, briefly introduces the most popular mashup platforms that are used today and explains the reasons behind our choice of Apatar as our core platform on which we will add the mobile device capability. Chapter 4 consists of a presentation of the best-known mobile device operating systems and a documentation about why we chose to work with Android. In Chapter 5 we present the implementation of our mobile version of Apatar and in Chapter 6 we demonstrate it by using it to produce a mobile mashup application. Finally, in Chapter 7 we discuss the conclusion and the future work for our project.

2 Background and Related Work

2.1 Mashups

The Internet and related technologies have created an interconnected world in which we can exchange information easily, process tasks collaboratively and form communities among users with similar interests to achieve efficiency and improve performance. Web services are emerging as a major technology for deploying automated interactions between distributed and heterogeneous applications and for connecting business processes which might span companies' boundaries. As a result, there is an emerging need for users to combine multiple services and data sources to best serve their goals. Mashups are applications developed specifically to satisfy that need.

The purpose of mashups is to allow users to control data in a self-service way, without any interference of experts in order to produce a result that would be perfectly suited to the individual needs of each user. As implied, a mashup should be easy to implement, so that the experts' interference is unnecessary, and reusable in order to reduce the time and effort needed to develop similar mashups over and over again.

In order to demonstrate the effectiveness of mashups, we present their use in a product delivery service. Assume that a courier company is responsible for delivering products from different e-shops. Each e-shop has its stores in different locations in a city so that the employees of the courier company have to visit all the locations before they deliver their orders. Considering that different e-shops may use different DBMS to store their records it would be impossible for the courier company that works together with all of them to use all the data in a homogeneous way. Using a mashup, the courier company can access data from its database and integrate it with all the different DBMS of its partners in a universal environment, that way, saving time and effort.

2.2 Mobile Applications[2]

In the past couple of years we have witnessed tremendous growth in mobile users all over the world as the entry of smartphones in the market at affordable prices has augmented their usage. We have experienced a major shift in the way we access the internet today with mobiles becoming the primary access point for internet usage. In today's fast paced world, phones are not used just for calling, playing games etc. but with smartphones we can schedule our complete day, check emails, make conference calls, connect using social network and perform a host of other activities.

The growth of mobile phone market has generated a huge demand for various mobile applications. Numerous mobile phone applications are available that simplify various tasks for the users due to which we saw an accelerated growth of software/application development for mobile devices. Mobile application development is the course of action by which application software is designed and developed for hand-held devices like mobile phones, tablets etc.

Earlier mobile developers face many difficulties while writing applications as they had to build better, unique, competing and hybrid applications which would incorporate command tasks like messaging and contact list calling in a user friendly manner. The launch of Android smartphones in the market brought a revolution in Mobile Application Development. If you have some basic knowledge about scripting and coding you can start building your own applications and as a result Mobile Application Development was never as easy as it is now.

Every mobile phone company platforms like Android, Apple iPhone OS, Rim, Blackberry OS, Symbian Os provide their own SDK (Software Development Kit) to the developer enabling them to create applications and even publish them to the world at the provided market.

Considering the current market trends online businesses like web hosting, shopping, job portals, etc. have developed mobile applications for their

clients to provide better services. As per market research experts there were 8.2 billion mobile app downloads in 2010 globally and is expected to reach 76.9 billion in 2014.

2.3 Related Work

To our knowledge, there has not been much work on using the mashup programming paradigm on a mobile platform. The only two similar projects we are aware of are The TELAR Mobile Mashup Platform for Nokia Internet Tablets described in [3]and the Dynamic Mashup Platform for Mobile Web Applications described in [4].

The TELAR Mobile Mashup Platform for Nokia Internet Tablets was, in our knowledge, the first complete idea of combining mobile devices, tablets in this case, with mashup applications. The implemented program was a map-based application in which maps were obtained from Google Maps [4], other kind of data such as WLAN access point locations and pictures of the specific places were obtained by other services and using the GPS sensor of the mobile device, it presented the local points of interest which were updated as the user was moving.

[4] was the first attempt to implement a dynamic mashup platform which was adapted according to the user's preferences. Two significant assumptions are made in order to implement this idea. Firstly, it is assumed that the mashup relations, such as the connection between the location of a place on the map and pictures of the place, are already known. Secondly, the user preferences are specified by by their usage patterns or programmed by the user-specified setting and therefore according to these preferences a list of mashups is proposed. After the user selects the target services or categories, a mashup engine generates recommendations of possible mashup services and finally the user selects the specific mashup service of their desire.

The main difference between the two projects and our work is that our work is a general purpose application. Although it does not straightly pro-

duce an application ready for use, it provides the mashup infrastructure to implement any application a user desires according to their needs. This is achieved by taking advantage of the conveniences of a mashup platform that is already implemented for desktop computers in a mobile environment.

3 Mashup Platforms

3.1 Current State

Many mashup tools have been developed to support the creation and execution of both consumer-focused and enterprise mashup applications. Here, we briefly introduce some of the most popular tools, which are more extensively analyzed in [4,5].

• ***Yahoo! Pipes***[1] (developed by Yahoo! Inc.) is a Web-based, consumer-oriented mashup platform. Mashups here called pipes = are created by connecting widgets provided by the platform. Currently data from Web feeds, Web pages and other services like flickr[2], can be mashed. Output can be accessed by a client as RSS or JSON, or can be visualized on a Yahoo! Map, or through an HTML page.

• ***Damia***[8],[12] is an enterprise-oriented mashup platform developed by IBM. It enables users to create mashups by assembling data feeds from Internet as well as enterprise data sources. It mainly focuses on data feed aggregation and allows additional tools like feed readers to be used at the presentation layer for the data feeds that it provides.

• ***Apatar*** is a mashup data integration platform developed by Apatar Inc. It allows users to aggregate and integrate locally-stored data with the Web by using visual editor to create mashups. Apatar mainly aims in manipulating data that will be used from other applications and this its output can be consumed by external tools.

• ***Exhibit***[9] is a framework for creating web pages with dynamic and rich visualizations of structured data. It enables its users to aggregate data obtained in various formats, like RDF/XML and Bibtex. Exhibit uses HTML pages as output but it also provides functionality for exporting its output to different formats, such as RDF/XML or Exhibit JSON.

[1] http://pipes.yahoo.com/pipes
[2] http://www.flickr.com

• ***MashMaker***[10],[11] is an interactive Web-based tool developed by Intel Corporation for editing, querying, manipulating and visualizing semi-structured data. It differs from other tools in the sense that it works directly on Web pages and allows users to create mashups when browsing by combining content from different Web pages. The final goal of MashMaker is to suggest mashups or widgets for the visited Web pages, that the user may want to use.

There are certainly many more mashup tools and platforms currently available, but describing and analyzing them is beyond the scope of this thesis.

3.2 Platform Selection

The goal of our work is to give mobile device users the opportunity to take advantage of the mashup paradigm and use it to develop sophisticated applications that perfectly match their demands. We wanted our application to be easy to use as we want it to be used from people with merely basic knowledge of scripting and coding. Therefore we believed that a platform that offers a visual editor to the users and involves limited coding would be an ideal choice.

Apatar is a mashup data integration tool that best meets our needs for several reasons. First of all, it essentially uses the mashup paradigm to support data integration. It provides a visual job designer through which the user may intuitively connect widgets to create a data integration schema. The fact that at worse very limited coding is needed in order to use it enables the platform to be used by people with very limited technical knowledge. Apatar also provides mechanisms called connectors for accessing and manipulating data stored locally or data from corporate resources. Additionally, connectors for accessing content on the Web, such as RSS feeds and some popular Web 2.0 APIs as Flickr, Salesforce.com and amazon Simple Storage Service (Amazon S3) are provided. Therefore, one can access data stored in spreadsheet documents in a local hard drive and mash it with data residing in a

corporate database or even data stored in the cloud (eg. in Amazon S3). This feature also qualifies Apatar as a potential enterprise-oriented mashup tool. Finally, Apatar supports operators for manipulating data by performing aggregations, filtering, joins, transformations and so on. Hence, we concluded that as far as functionality is concerned, this set of capabilities makes Apatar a good choice as a base to start building on.

However, there are also some technical aspects that we had to consider. The most important one is the source code availability. Apatar is open source software, distributed under the GNU General Public License (GPL). This is a key feature because it allows us to actually use its source code, modify it and extend it to meet our goals. We must note here, that Apatar is one of the very few options offering source code availability since the source code of most mashup platforms is not publicly available. Another significant feature is that Apatar's source code is very well structured thus simplifying the procedure of reading and understanding it, which is necessary in order to modify it and extend it. Finally, Apatar is designed to be extensible, and thus, it provides a standard procedure for the developer to create new functionality and plug it in the core application engine. These features, as well as those mentioned in the previous paragraph, make Apatar the best available choice for a starting point for our work.

3.3 Apatar's Architecture

Apatar is an open source Extract Transform and Load (ETL) project. As illustrated in Figure 3.1, it is structured in three basic components: The core component, the connectors' component and the user interface component.

3.3.1 Core Engine

The core component consists mainly of the application's ETL engine, which is where the actual data processing takes place. For every operation,

3.3 Apatar's Architecture

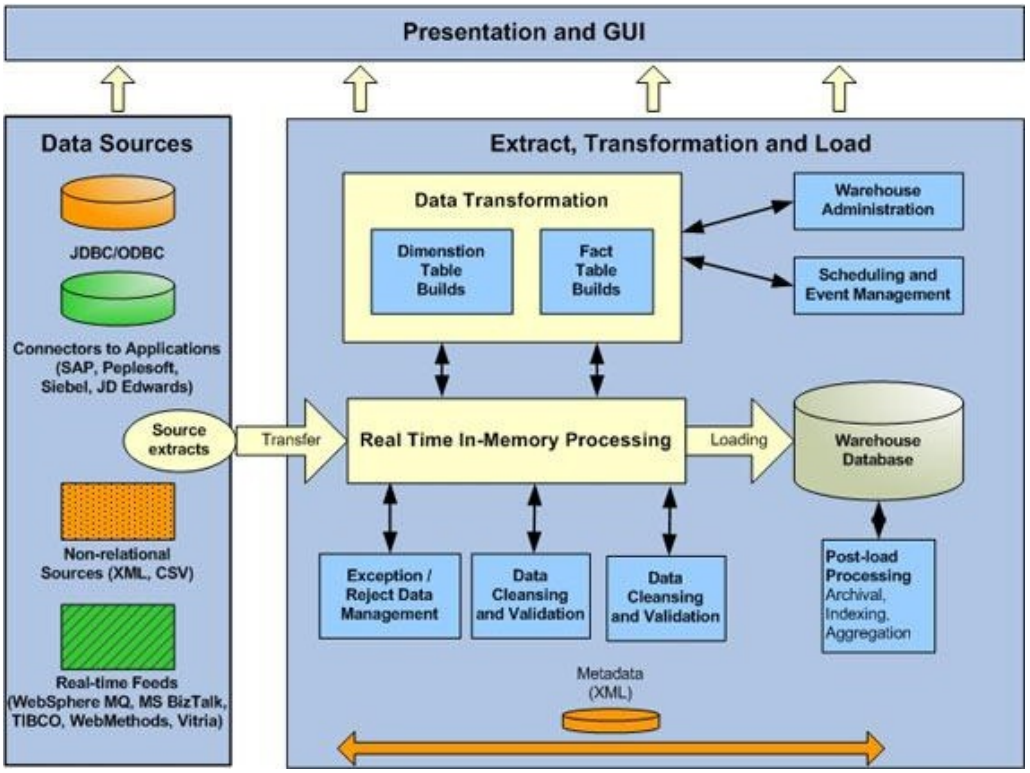

Figure 3.1: Apatar's architecture

data is retrieved from one or more data sources through the connectors' component and is transformed to tuples in Apatar's internal database. In this form, data is processed by the application's engine, and then loaded again to one or more connectors and probably to the presentation layer. Some of the operations that are currently available are high level operations, such as joins, selections, aggregations, filtering and so on, as well as lower level operations, such as transformations between different data types.

The core component is also responsible for defining fundamental structures to hold information relevant to the data manipulation, as well as for providing a mechanism to support and ensure the platform's consistency and extensibility. For example, the core component defines structures to represent the platform's internal database, its tables and its records, and also abstractly defines the way that a connector must be structured in order to be functional.

3.3.2 Connectors

The connectors' component is used to connect the application's core engine with data sources. Every connector provides a connection point for a specific data source through which data can be read, written, or both. Currently, a large set of connectors are provided and the supported data sources can vary from corporate databases and personal files, to e-mails and Web 2.0 APIs.

3.3.3 GUI and Data Representation Layer

Finally, the third component consists of a graphical user interface and a simple data presentation layer that simplifies both the use of the application and user control over data. Through the GUI, users interact with Apatar, to create, modify, publish or run mashup applications, while the data presentation layer enables data supervision at any stage of the workflow.

The main application window is divided in two areas. The connectors and functions' area, where the different connectors and functions are displayed as widgets, and the work area which is used for creating mashups called datamaps. To create a mashup application, the user just needs to drug and drop the necessary connectors in the work area, configure them and connect them together to form a datamap.

3.3.4 Extensibility

As stated earlier, Apatar is designed to be an extensible platform, a goal achieved though the use of the Java Plug-in Framework (JPF). JPF provides a run-time engine which can dynamically discover and load plug-ins. A plug-in is considered to be a structured component that describes itself to the use of a manifest. Plug-ins and the functions they provide are added to a registry at start-up-time or at run-time but are not loaded until they are called. In this manner, applications using JPF avoid paying any memory or performance penalty for plug-ins that are installed but not used.

3.3 Apatar's Architecture

Everything in Apatar is implemented as a plug-in. Every component, from core components and functions to connectors and GUI, is described by an XML document called plugin.xml. This document contains all the necessary information to describe the plug-in to JPF so that it can be registered in the framework and loaded upon call. This information would be the plug-in's identification, the path of the implementation classes, references to other plug-ins that are required and so on. Every time the application starts, a predefined plug-in folder is scanned for the manifest files, the available plug-ins are registered to the JPF and from this point on they are ready to be used on demand.

4 Mobile Operating Systems

4.1 Current State

A mobile operating system, also referred as mobile OS, is the operating system that operates a smartphone, tablet, PDA or other digital mobile devices. Modern mobile operating systems combine the features of a personal computer operating system with touchscreen, cellular, Bluetooth, WiFi, GPS mobile navigation, camera, video camera, speech recognition, voice recorder, music player, Near field communication, personal digital assistant (PDA) and other features.

The most common mobile operating systems are:

• ***Android*** was developed by a small startup company that was purchased by Google Inc. in 2005, and Google continues to update the software. Android is a Linux-derived OS backed by Google, along with major hardware and software developers (such as Intel, HTC, ARM, Samsung, Motorola and eBay, to name a few), that form the Open Handset Alliance. Released on November 5th 2007, the OS received praise from a number of developers upon its introduction. Android releases prior to 2.0 (1.0, 1.5, 1.6) were used exclusively on mobile phones. Most Android phones, and some tablets, now use a 2.x release. Android releases are nicknamed after sweets or dessert items like Cupcake (1.5), Frozen Yogurt (2.2), Honeycomb (3.0), Ice Cream Sandwich (4.0) and Jelly Bean (4.1). Most major mobile service providers carry an Android device. Since the HTC Dream was introduced, there has been an explosion in the number of devices that carry Android OS. From Q2 of 2009 to the second quarter of 2010, Android's worldwide market share rose 850% from 1.8% to 17.2%. On November 2011, Android reached 52.5% of the global smartphone market share.

• ***bada*** is a mobile system being developed by Samsung Electronics. Samsung claims that bada will rapidly replace its proprietary feature phone platform, converting feature phones to smartphones. The name 'bada' is derived

from the Korean word for ocean or sea. The first device to run bada is called 'Wave' and was unveiled to the public at Mobile World Congress 2010. The Wave is a fully touchscreen running the new mobile operating system. With the phone, Samsung also released an app store, called Samsung Apps, to the public. It was close to 3000 mobile applications.

Samsung has said that they don't see bada as a smartphone operating system, but as an OS with a kernel configurable architecture, which allows the use of either a proprietary real-time operating system, or the Linux kernel. Though Samsung plans to install bada on many phones, the company still has a large lineup of Android phones.

• ***BlackBerry OS*** released from RIM, is focused on easy operation and was originally designed for business. Recently it has seen a surge in third-party applications and has been improved to offer full multimedia support. Currently Blackberry's App World has over 50000 downloadable applications. RIM's future strategy will focus on the newly acquired QNX, having already launched the BlackBerry PlayBook tablet running a version of QNX and expecting the first QNX smartphones in early 2012.

• ***iOS*** from Apple inc. is used by the Apple iPhone, iPod Touch, iPad, and second-generation Apple TV and is derived from Mac OS X. Native third-party applications were not officially supported until the release of iOS 2.0 on July 11th 2008. Before this, jailbreaking allowed third-party applications to be installed, and this method is still available. Currently all iOS devices are developed by Apple and manufactured by Foxconn or another of Apple's partners.

• ***S40*** from Nokia is used by Nokia in their low end phones (aka feature phones). Over the years over 150 phone models have been developed running S40 OS. Since the introduction of S40 OS it has evolved from monochrome low resolution UI to full touch 256k color UI.

• ***Symbian OS*** from Nokia and Accenture has the largest smartphone share in most markets worldwide, but lags behind other companies in the relatively small but highly visible North American market. This matches the

success of Nokia in all markets except Japan. In Japan Symbian is strong due to a relationship with NTT DoCoMo, with only one of the 44 Symbian hand sets released in Japan coming from Nokia. It has been used by many major handset manufacturers, including BenQ, Fujitsu, LG, Mitsubishi, Motorola, Nokia, Samsung, Sharp and Sony Ericsson. Currently Symbian-based devices are being made by Fujitsu, Nokia, Samsung, Sharp ans Sony Ericsson. Prior to 2009 Symbian supported multiple user interfaces, i.e. UIQ from UIQ Technologies, S60 from Nokia, and MOAP from NTT DOCOMO, As part of the formation of the Symbian OS in 2009 these three UIs were merge into a single OS which is now fully open source. Recently, through shipments of Symbian devices have increased, the operating system's worldwide market share has declined from over 50% to just over 40% from 2009 to 2010. Nokia handed the development of Symbian to Accenture, which will continue to support the OS until 2016.

- ***Windows Phone*** from Microsoft was unveiled on February 15th, 2010 as their next-generation mobile OS. The new mobile OS includes a completely new over-hauled UI inspired by Microsoft's Metro Design Language. It includes full integration of Microsoft services such as Microsoft SkyDive and Office, Xbox music, Xbox Video, Xbox Live games and Bing, but also integrates with many other non-Microsoft services such as Facebook and Google accounts. The new software platform has received some positive reception from the technology press.

4.2 Mobile OS Selection

Although Windows Phone and BlackBerry OS are looking for the opportunity to exploit any possible opening and weakness in the smartphone market, it is quite obvious that the battle of the giants is between iOS and Android.

The best solution would be to develop multi-platform apps in order to allow them being used from both OS. However Android apps are Java-based while iPhone are Objective-C-based so apps from one platform cannot run on

the other. Some multi-platform app development tools have shown up lately but they have proven to be ineffective in displaying the original information on another mobile OS, so multi-platform development is currently not viable.

As mentioned previously in the presentation of the most common mobile OS, Android is the most commonly used mobile OS. On November 2011, it reached 52.5% of the global smartphone market share. Hence, it starts with an edge in order to be our OS of choice. The technical aspects that follow it enhance that edge making it the ideal OS for our work.

Android allows its developers to use open development platforms and enables the use of third-party tools. This offers the opportunity to play around with more specific aspects of the developed application and therefore makes it more sophisticated and increases its functionality. This feature is key to the success of Android, specifically given the fact that it comes to a great variety of devices. On the contrary, iOS is quite restrictive concerning its developers, giving them specific developer guidelines. A fixed set of tools is available but nothing else can be used apart from those, which significantly limits the options of the programmer.

Testing your mobile app is made an easy task for Android developers, as it offers an excellent testing environment. All the testing tools available are neatly indexed and the IDE offers a good model of the source code. Therefore testing the app and even more importantly debugging whatever needs it before publishing the app to Google Play store is done with ease and comfort. Apple's Xcode is fare behind Android's level and has a long way to cover before it can even be compared with it.

Furthermore, Android is far easier in getting your app on the device and publishing it to Google Play store, as the App Store review process has proven to be unpredictable and inconsistent.

Finally, focusing in our specific work, after as already mentioned choosing Apatar, which is written in Java as our mashup platform, it would be really convenient to be able to write our mobile app also in Java. This would allow the client and server parts of our program to interact easily and be compatible

in many aspects. Moreover, the fact that we can even use the same IDE, Eclipse, for both parts, as there are Eclipse plugins both for Apatar and Android app development, makes choosing Android as the mobile OS we are going to use certain and undoubted.

4.3 Android Application Fundamentals[7]

Android applications are written in the Java programming language. The Android SDK tools compile the code-along with any data and resource files-into an Android package, an archive file with an .apk suffix. All the code in a single .apk file is considered to be one application and is the file that Android-powered devices use to install the application.

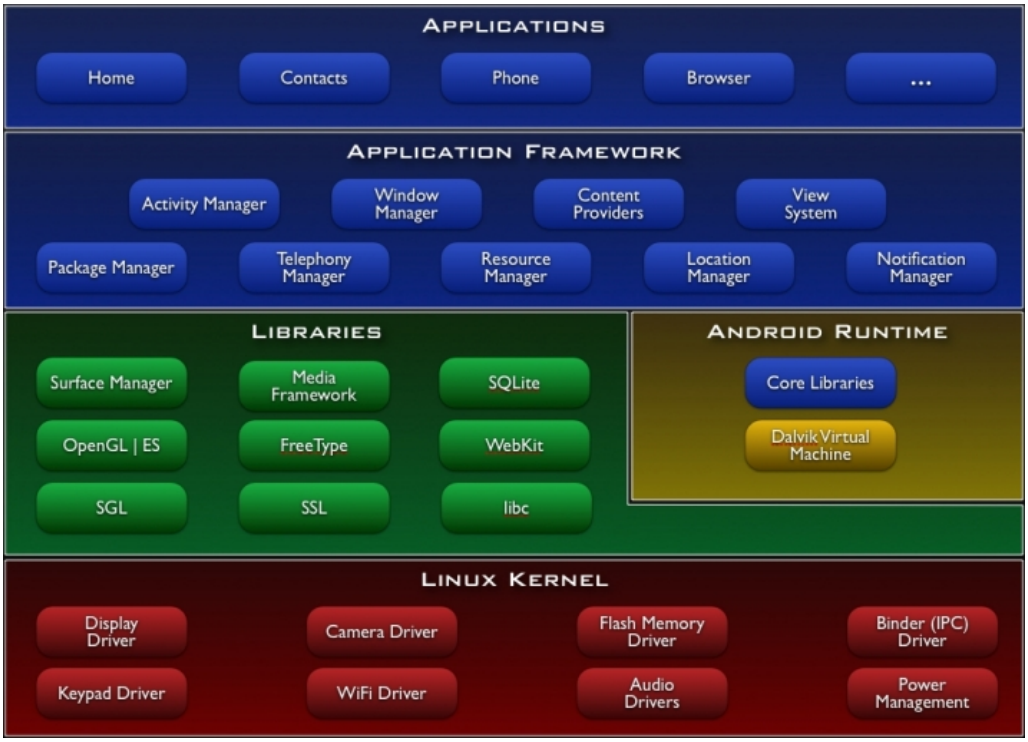

Figure 4.1: Android architecture

Once installed on a device, each Android application lives in its own security sandbox:

• The Android operating system is a multi-user Linux system in which each application is a different user.

• By default, the system assigns each application a unique Linux user ID (the ID is used only by the system and is unknown to the application). The system sets permissions for all the files in an application so that only the user ID assigned to that application can access them.

• Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications.

• By default, every application runs in its own Linux process. Android starts the process when any of the application's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other applications.

In this way, the Android system implements the principle of least privilege. That is, each application, by default, has access only to the components that it requires to do its work and no more. This creates a very secure environment in which an application cannot access parts of the system for which it is not given permission.

However, there are ways for an application to share data with other applications and for an application to access system services:

• It's possible to arrange for two applications to share the same Linux user ID, in which case they are able to access each other's files. To conserve system resources, applications with the same user ID can also arrange to run in the same Linux process and share the same VM (the application must also be signed with the same certificate).

• An application can request permission to access device data such as the user's contacts., SMS messages the mountable storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.

That covers the basics regarding how an Android application exists within the system. The rest of this section introduces you to :

• The core framework components that define your application.

• The manifest file in which you declare components and required devices features for your application.

• Resources that are separate from the application code and allow your application to gracefully optimize its behavior for a variety of device configurations.

4.3.1 Application Components

Application components are the essential building blocks of an Android application. Each component is a different point though which the system can enter your application. Not all components are actual entry points for the user and some depend on each other, but each one exists as its own entity and plays a specific role-each one is a unique building block that helps define your application's overall behavior.

There are four different types of application components. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

Here are the four types of application components as shown in figure 4.2[13]:

An **activity** represents a single screen with a user interface. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. Although the activities work together to form a cohesive user experience in the email application, each one is independent of the others. As suck, a different application can start any one of these activities (if the email application allows its). For example, a camera application can start the activity in the email application that composes new mail, in order for the user to share a picture.

A **service** is a component that runs in the background to perform long-

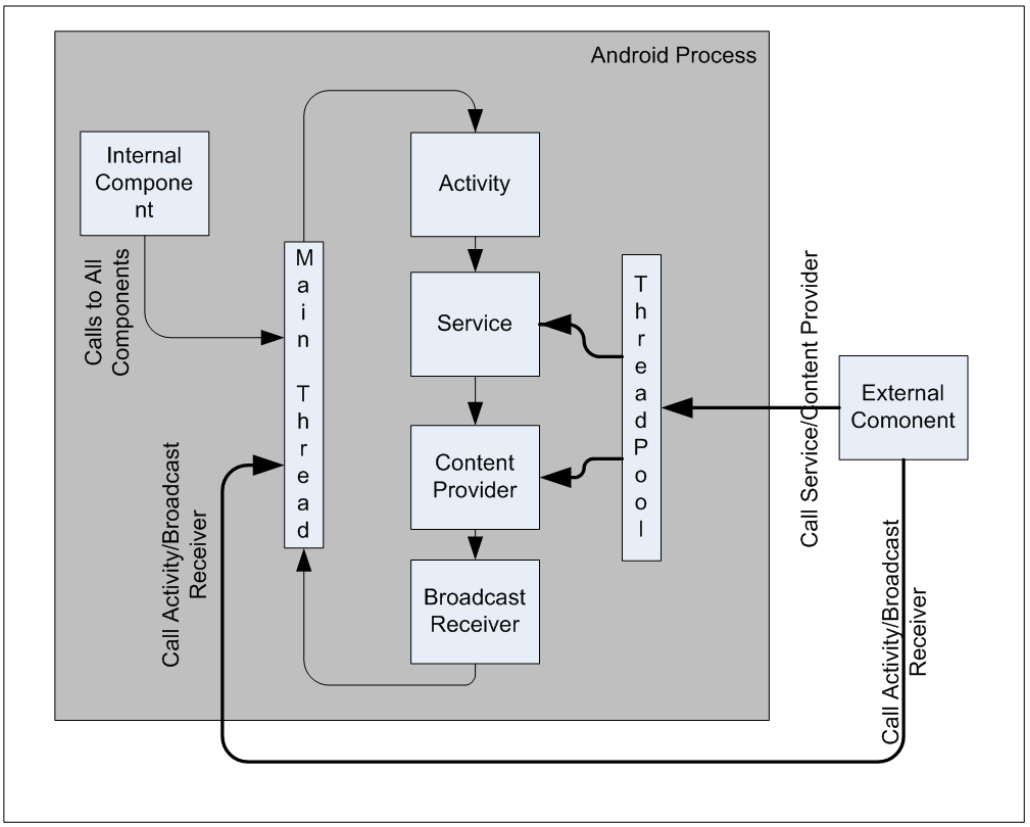

Figure 4.2: Android application components

running operations or to perform work for remote processes. A service does no t provide a user interface. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity. Another component such as an activity, can start the service and let it run or bind to it in order to interact with it.

A **content provider** manages a shared set of application data. You can store the data in the file system, an SQLite database, on the web, or any other persistent storage location your application can access. Through the content provider, other applications can query or even modify data (if the content provider allows it). For example, the Android system provides a content provider (such as ContactsContract.Data) to read and write information about a particular person.

Content providers are also useful for reading and writing data that is private to your application and not shared.

A **broadcast receiver** is a component that responds to system-wide broadcast announcements. Many broadcasts originate from the system-for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts-for example, to let other applications know that some data has been downloaded to the device and is available for them to use. Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a gateway to other components and is intended to do a very minimal amount of work. For instance, it might initiate a service to perform some work based on the event.

A unique aspect of the Android system design is that any application can start another application's component. For example, if you want the user to capture a photo with the device camera, there's probably another application that does that and your application can use it, instead of developing an activity to capture a photo yourself. You don't need to incorporate or even link to the code from the camera application. Instead, you can simply start the activity in the camera application that captures a photo. When complete, the photo is even returned to your application so you can use it. To the user, it seems as if the camera is actually a part of your application.

When the system starts a component, it starts the process for that application (if it's not already running) and instantiates the classes needed for the component. For example, if your application starts the activity in the camera application that captures a photo, that activity runs in the process that belongs to the camera application, not in your application's process. Therefore, unlike on most other systems, Android applications don't have a single entry point (there's no main() function, for example).

Because the system runs each application in a separate process with file permissions that restrict access to other applications, your applications can-

not directly activate a component from another application. The Android system, however, can. So, to activate a component in another application, you must deliver a message to the system that specifies your intent to start a particular component. The system then activates the component for you.

4.3.2 Activating Components

Three of the four component types-activities, services, and broadcast receivers-are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime (you can think of them as the messengers that request an action from other components), whether the component belongs to your application or another.

For activities and services, an intent defines the action to perform (for example, to view or send something) and may specify the URI of the data to act on (among other things that the component being started might need to know). For example, an intent might convey a request for an activity to show an image of to open a web page. In some cases, you can start an activity to receive a result, in which case, the activity also returns the result in an intent (for example, you can issue an intent to let the user pick a personal contact and have it returned to you-the return intent includes a URI pointing to the chosen contact).

For broadcast receivers, the intent simply defines the announcement being broadcast (for example, a broadcast to indicate the device battery is low includes only a known action string that indicates battery is low).

The other component type, content provider, is not activated by intents. Rather, it is activated when targeted by a request from a ContentResolver. The content resolver handles all direct transactions with the content provider so that the component that's performing transactions with the provider doesn't need to and instead calls methods on the ContentResolver object. This leaves a layer of abstraction between the content provider and the component requesting information (for security).

4.3.3 The Manifest File

Before the Android system can start an application component, the system must know that the component exists by reading the application's AndroidManifest.xml file (the manifest file). Your application must declare all its components in this file, which must be at the root of the application project directory.

The manifest does a number of things in addition to declaring the application's components, such as:

• Identify any user permissions the application requires, such as Internet access or read-access to the user's contacts.

• Declare the minimum API Level required by the application, based on which APIs the application uses.

• Declare hardware and software features used or required by the application, such as a camera, bluetooth services, or a multitouch screen.

• API libraries the application needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

• And more.

4.3.4 Application Resources

An Android application is composed of more than just code-it requires resources that are separate from the source code, such as images, audio files, and anything relating to the visual presentation of the application. For example, you should define animations, menus, styles, colors, and the layout of activity user interfaces with XML files. Using application resources makes it easy to update various characteristics of your application without modifying code and-by providing sets of alternative resources-enables you to optimize your application for a variety of device configurations (such as different languages and screen sizes).

For every resource that you include in your Android project, the SDK build tools define a unique integer ID, which you can use to reference the resource from your application code or from other resources defined in XML.

One of the most important aspects of providing resources separate from your source code is the ability for you to provide alternative resources for different device configurations. For example, by defining UI strings in XML, you can translate the string into other languages and save those string in separate files. Then, bases on the language qualifier that you append to the resource directory's name and the user's language setting, the Android system applies the appropriate language strings to your UI.

5 Apatar Android Version

The key contribution of our work is to offer app developers the ability to take advantage of the mashup programming paradigm in their mobile applications. We developed a simple Android client application that simulates the Desktop GUI of Apatar in a more simple and explanatory way so that experienced Apatar users were familiar with the applications environment and newer users got easily accustomed with it. We also developed the server part of the application, which is an expansion of the standard Apatar program. Its responsibility is to receive input from the client, process it and produce output, sending it to the client.

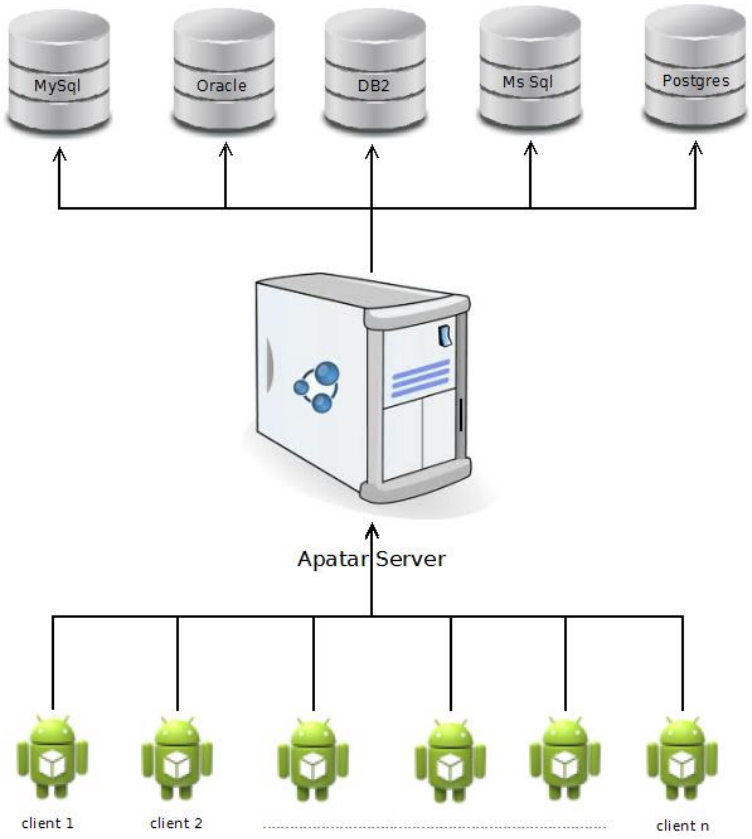

Figure 5.1: Client-server interaction

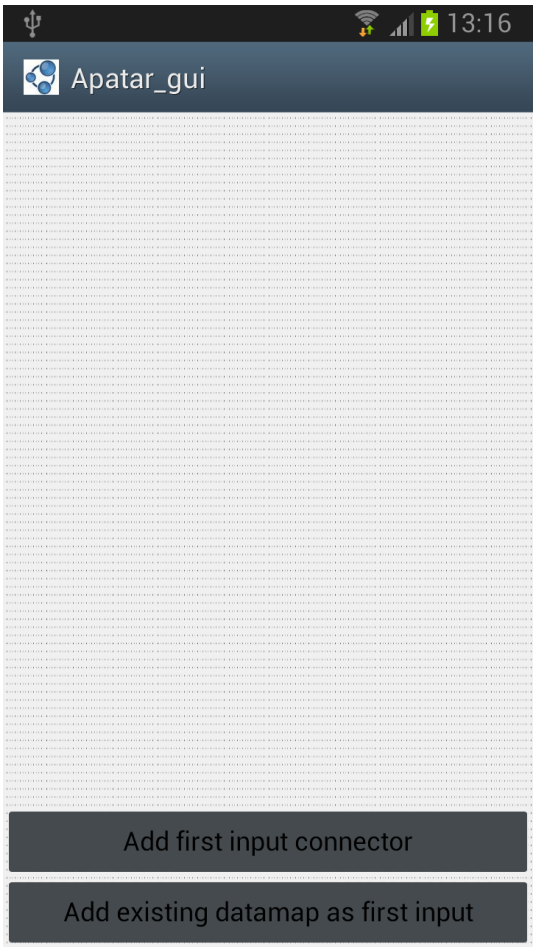

Figure 5.2: Application homepage

5.1 Client

The main challenge in order to create an implementation of Apatar's GUI in a mobile platform such as Android is to adapt all the required data in a smaller display losing as less information as possible. As already mentioned, the main window of the desktop application is divided in two areas, the connectors' and functions' area, and the work area. Since our display resource in a mobile device could be quite small (smallest Android device display is 2.7") we have decided that these two areas should be divided into two screens.

Another important decision we had to make was where to place the dif-

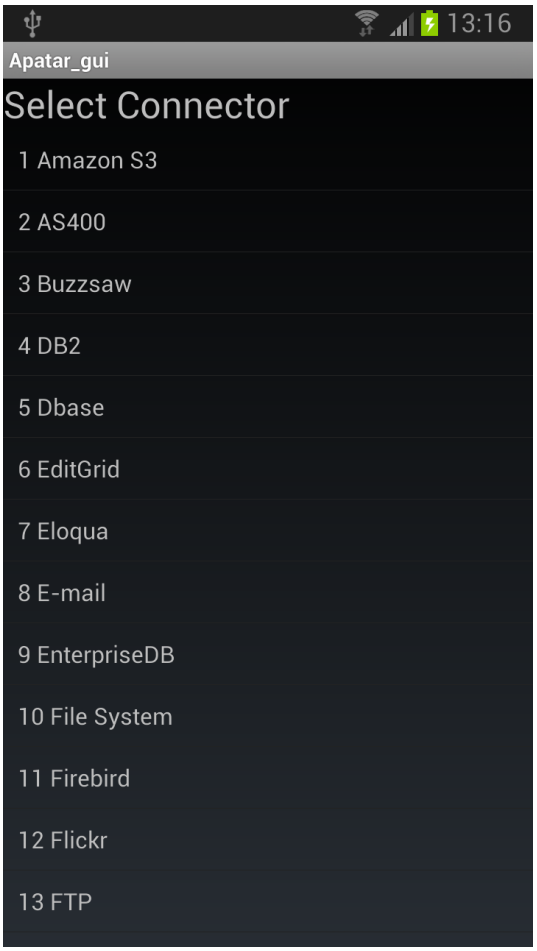

Figure 5.3: Connector selection screen

ferent options from which the user can choose. We decided to place buttons in the bottom of each screen so that the user could see the options they had at any given time, simply by scrolling down each page. The input methods of the application, is the touch screen of the device and the default keyboard whenever input is required. We tried to use the most standard input mechanisms such as text fields, checkboxes and menus so that the application would be as easy as possible to use.

The latest Android versions do not allow the application to access any network in the main thread of the former. Hence, in order to communicate with the server, our client will have to create multiple threads that send

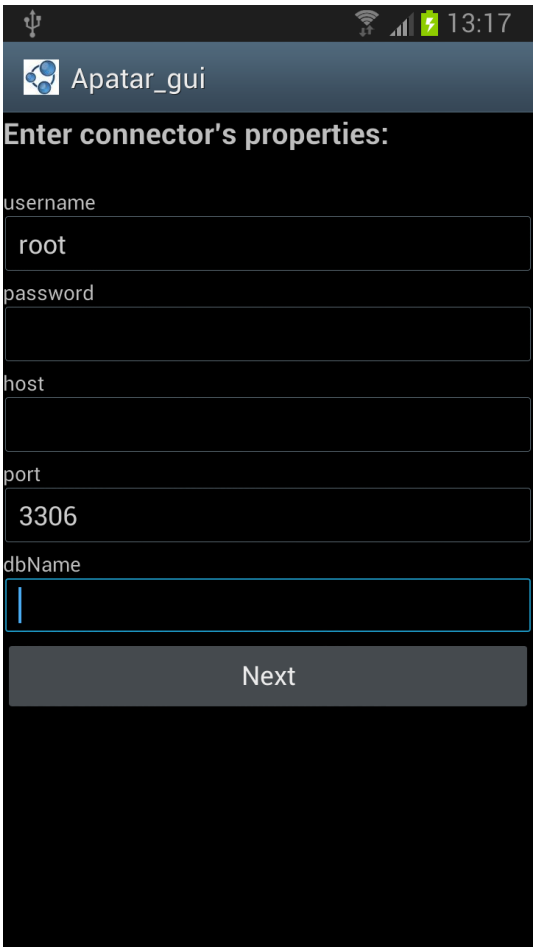

Figure 5.4: Example of connector properties input-MySQL connector

and receive data. The most common way of doing so is by using the class AsyncTask. AsyncTask.class is a Java class that enables proper and easy use of the UI thread. It allows to perform background operations and publish results on the UI thread without having to manipulate threads and handlers. Therefore it is ideal for our purpose.

5.2 Server

The server part of our application is where most of the work is done. When the server is run, apart from the main Apatar application another

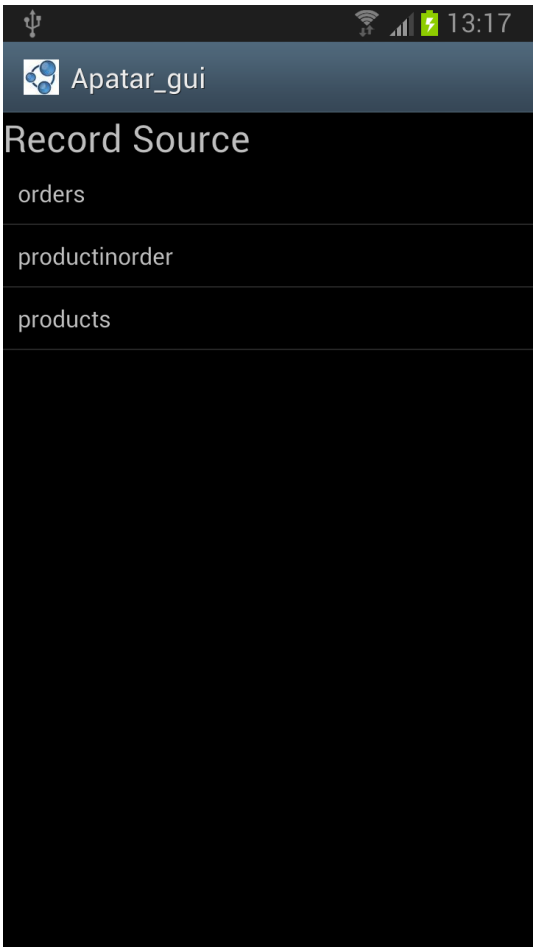

Figure 5.5: Record source selection screen

thread is executed which awaits for data to be received. Depending on the data that is received, the server performs the according actions.

When data is received, firstly another server thread initiates, so that the server can handle multiple and simultaneous requests. After that, the data string that was sent by the client is analyzed. Each data string consists of two parts. The first one, contains the serial number of the requested operations and the second one, the variables needed to adjust the operation. For instance, if the user wants to create a new MySql connector, the first part of the string contains the serial number of the function Create new connector and the serial number that corresponds to MySql; the second part, contains

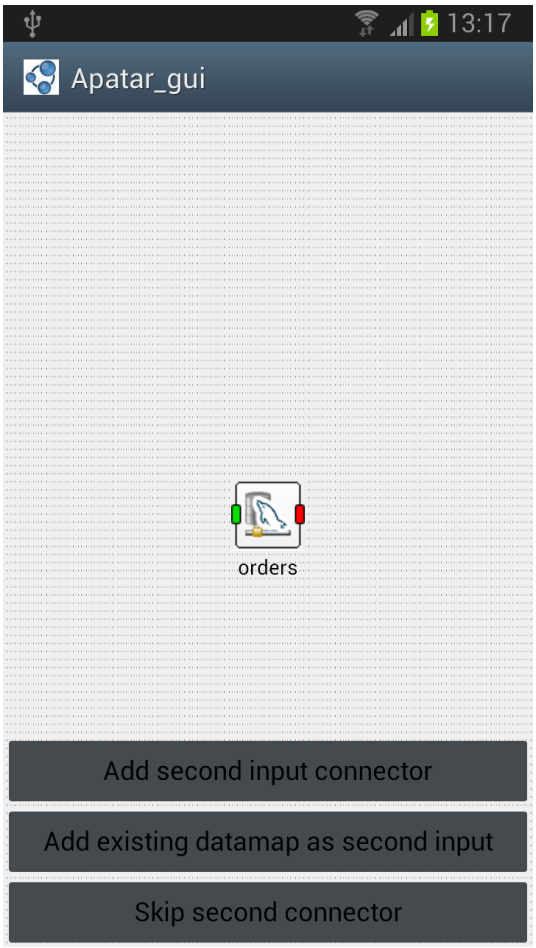

Figure 5.6: MySQL connector added

the values of the variables of the connector such as username, password, host, port and DB name.

After analyzing the data string, the server gives the received values to the appropriate variables and if needed asks the client for further information, such as the desired record source of the DB. In the end, when the server has all the information needed, it performs the action requested by the client and awaits for further requests. Each server window corresponds to a specific client, so that confusion between requests from different users is avoided.

Apart from the application part of the server, a server directory was essential. In it, we have stored resources needed by the application. We have

files which contain the currently available connectors and operations, the properties of each connector, and images used to make the GUI of the application more pleasant. Using this directory has two benefits. Firstly it allows us to perform trivial tasks such as loading standard data from the server without having to write complicated source code and merge it to the main application code. These trivial tasks are performed with a simple php script that is accessed whenever the client attempts to connect to the server. Secondly it makes extensibility easier. More specifically, if the server administrator wishes to add a new connector of the already existing kind, such as DB connector, all they need to do, is enter the name of the connector in the connector.txt file of the directory, create a new file with the connector serial number as a name and the properties that need to be completed as a body and add a picture file for the connector.

5.3 Functionality

After creating the connectors needed, they choose the operation they want and the results are stored in a custom table connector, which is the most versatile Apatar connector. After this one (or two) input connector datamap is completed, the user may view its results and decide whether they want to save or discard the datamap. As already mentioned, the small display of some Android devices is a significant limiting factor. Therefore, we felt that in order to make the functionality of the application easier, we had to use the divide and conquer approach. More specifically, users, initially have the option to create datamaps with either one or two input connectors. Datamaps are saved in the SD card of the device and may be used in the future as one of the inputs of a new datamap. In this way, by creating small, simple and independent datamaps consisting of one or two inputs, an operation and a custom table to store the output and by combining them, users may create more complicated datamaps in order to suit their needs.

Saved datamaps are simple text files that contain all necessary information to reproduce the initial datamap. When one of them is selected as input

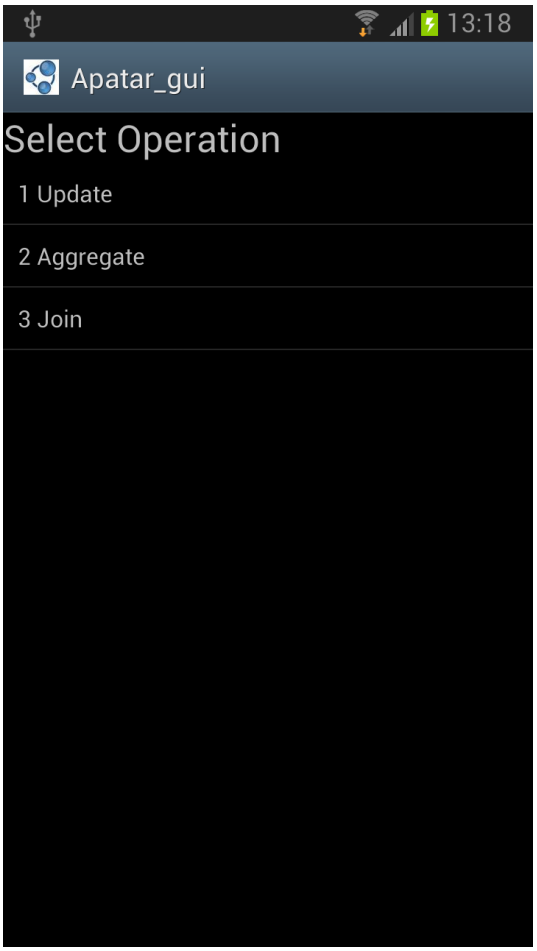

Figure 5.7: List of operations screen

the server instantly reproduces the datamap and waits for the rest of the inputs. These files can be used on the later stage of the app development. The developer needs to use the apatarClient.class we created and give to it as input the desired datamap file. The class, will send the necessary information to the Apatar server. The server will create the datamap described by the input file, execute it and send the resulting records back the apatarClient.class. Hence, the records can be accessed in application.

ApatarClient.class's implementation is quite straightforward. It consists of a constructor and a 'run' function. The constructor needs a String type argument that defines the path of the datamap file that the developer wants

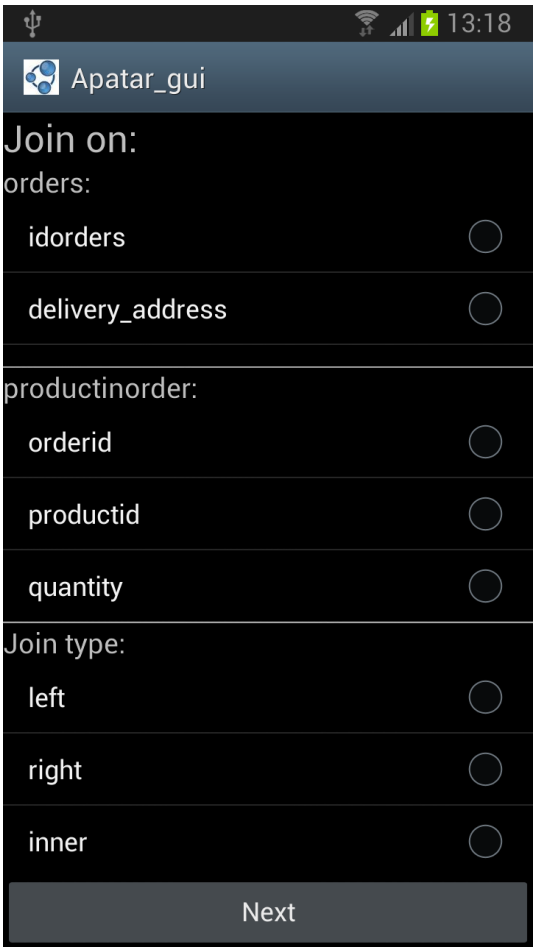

Figure 5.8: Enter join operation details screen

to be executed. After constructing an ApatarClient object we call the 'run' function that sends the datamap to the server and receives the records in a two-dimension array. the first row of the array keeps the names of the attributes that are fetched and the rest of the rows keep the records. It has to be mentioned that since an Android app cannot access the Internet in its main thread of execution, the run function must be called in another thread, otherwise it will not be able to connect to the server. AsyncTask.class is ideal for that use.

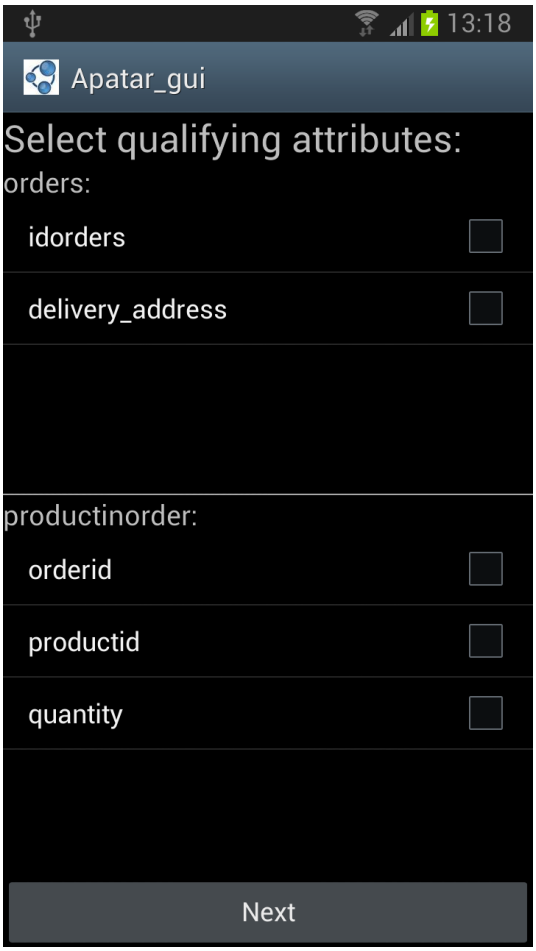

Figure 5.9: Qualifying attributes screen

6 Demonstration

To demonstrate our work we have created an application that would take advantage of the mashup programming paradigm, the Google Maps API that exists for Android app development and the fact that we are using a mobile device. The senario of the application is that a courier who works for an e-shop begins his route from the central shop of the company to deliver the customer orders. Some of the products that they need to deliver are available in the central shop whereas others are available in warehouses that are distributes in random locations. To deliver an order, the courier must first

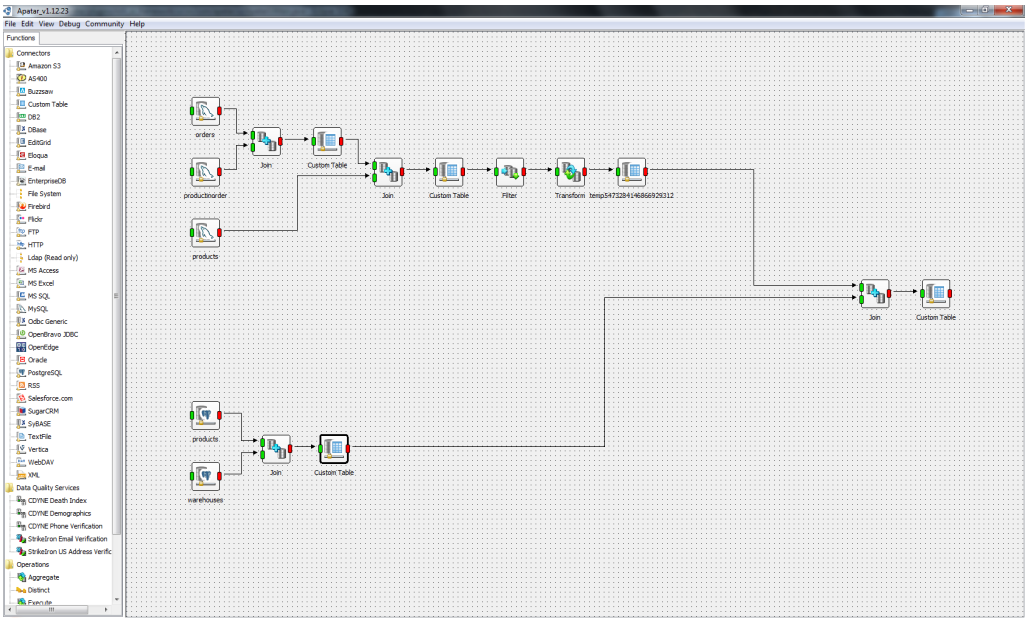

Figure 6.1: The mashup application

collect all its products from the warehouse in which they are available. Our goal is to produce the shorter route that exists so that the courier delivers all the orders that can be delivered without making unnecessary detours.

We are assuming that the e-shop company and the warehouse company are different, so they keep their records using different databases. The e-shop company keeps the records that contain orders and product availability in a MySQL database whereas the warehouse company keeps them in a PostgreSQL database. Therefore we need a mashup in order to integrate the records of the two heterogenerous databases. Figure 6.1 depicts the complete mashup application.

In the beginning of the datamap we join the order and productInOrder table of the e-shop database in order to find the products that exist in an order and we join the result with the product table so that we have extra information about these products such as product name and product availability. After that we filter out the products that are available since we need to keep the unavailable products. Those unavailable products will be the first input of our mashup.

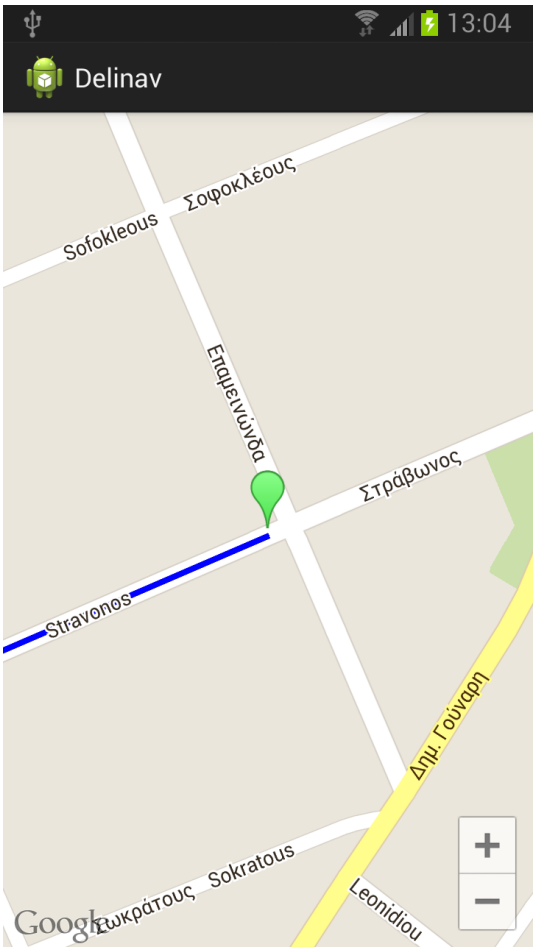

Figure 6.2: Application homescreen

The second input is simpler. We just joined the products table and the warehouse table of the warehouse database so that we keep track of which product is available in which database.

The actual mashup takes place here, where we join the products that are unavailable in the central shop and the products that exist in the warehouses. The output will be a table with all the information we need about a product such as product name, productId etc, information about the order that it belongs and information on the warehouses in which it is available. Using all these information we can create a list that contains all the addresses that the courier must visit and the order in which they have to visit them.

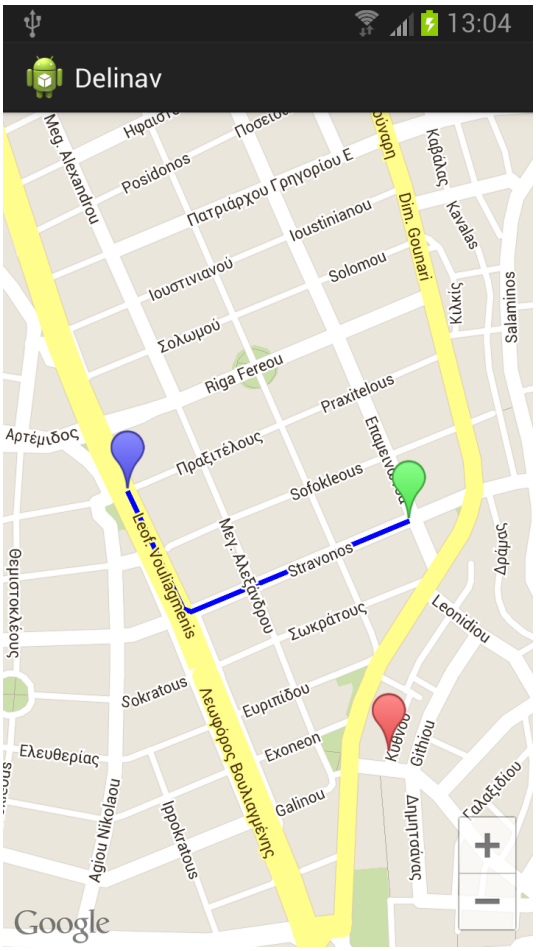

Figure 6.3: Route to next destination

By reviewing the datamap fuctionality carefully, one sees that the data accessed from it does not contain the whole order table records. Hence, apart from the mashup we needed to access the MySQL database of our eshop in order to gain access to all the orders that need to be delivered. Android cannot accurately connect to a DBMS. In order to overcome that problem we wrote a simple php script that fetches the data from the order table and we connected our Android app to it (figure 6.5).

This is where we needed a service that would provide us with the maps and a means to mark locations and routes on them. Google Maps is the perfect choice for our needs as it offers us whatever we need and also has an available

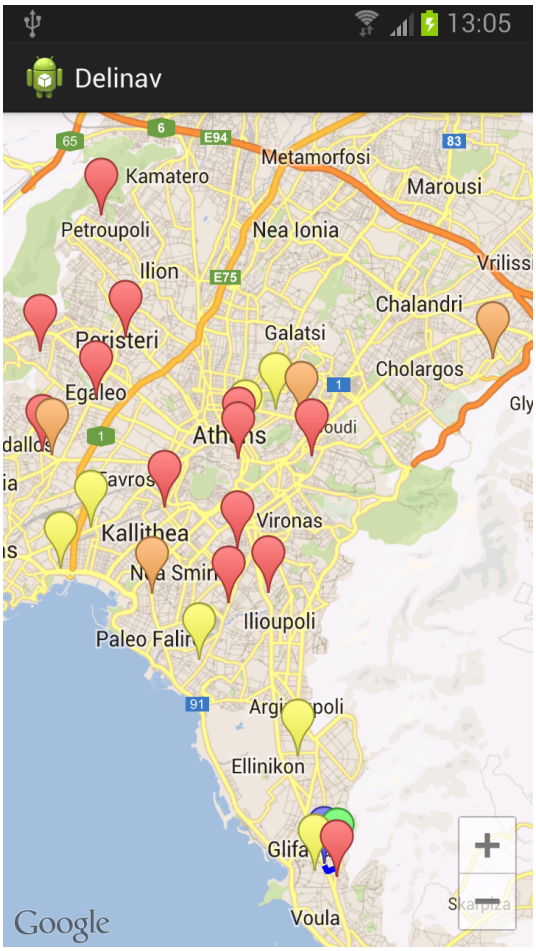

Figure 6.4: All destinations

API for Android. From the information we had from the mashup we create a list with all the places that the courier has to visit. Google maps informs us about the distance between each destination and our location, which is found using the GPS sensor of the mobile device. We then use a simple algorithm to create the best available route. We check if the closest destination is a warehouse. If indeed it is, then we visit it. If it is a delivery address we check if all its products are available and if they are we deliver it; if not, we lower it to the priority list and check the next closes destination. Finally we have two lists, one priority list of the future destinations in the order that we need to visit them and one list of the orders that cannot be delivered as

Figure 6.5: Android MySql connection

the required products are not available in the required quantities neither in our store nor in any warehouse. When the user runs the application it shows the map centered near their current location, a route to the next destination and different colour spots for different categories of future destinations.

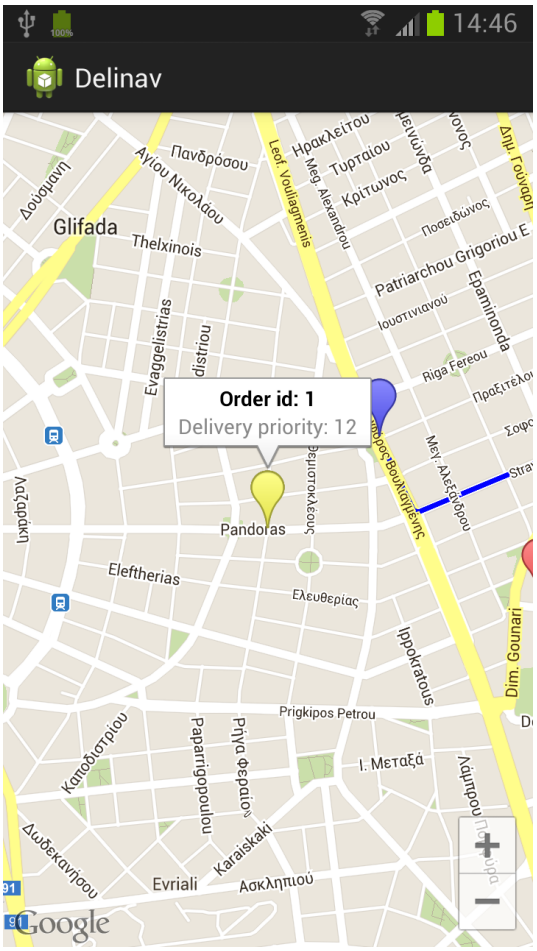

Figure 6.6: Selecting a destination

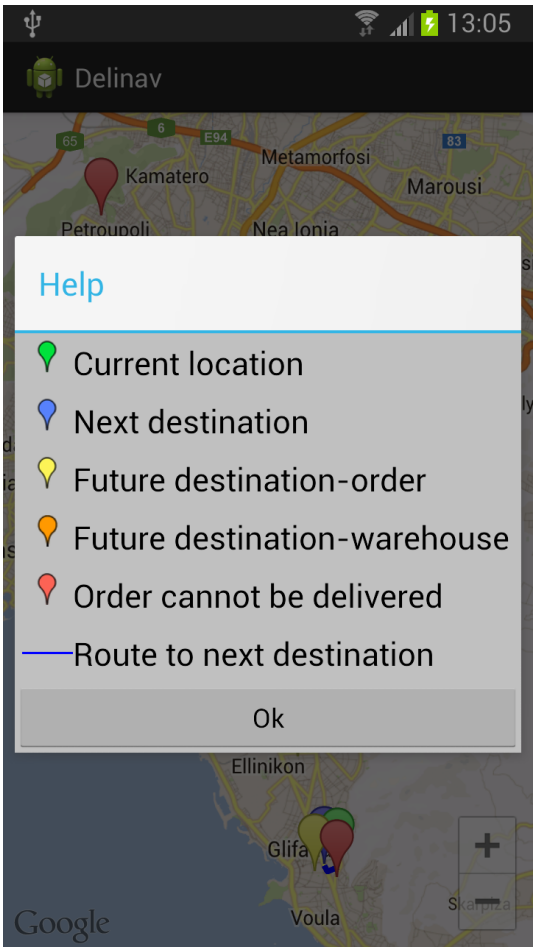

Figure 6.7: Symbol explanation

7 Conclusions and Future Work

Expanding a powerful tool such as the Apatar open-source mashup platform in a mobile version enabled mobile applications to enjoy the benefits of the mashup programming paradigm. This, by itself, opens countless avenues of possible applications that can be developed and combine mobility and data integration. We believe that our work can help so that the transformation from a sophisticated idea to a mobile application is possible. We also demostrated the power of such a combination through a simple application we created.

We intend to perform a lot of improvements to our system so that it is more stable, pleasant to work with and as close as possible to the original Apatar version. Apart from debugging and optimization we plan to add the option of a drag-and-drop GUI so that connector nodes and connection arrows can be manipulated using the touch screen of any mobile device, instead of just browse back and forth via menus.

Apart from creating applications using our platform, work can be done in order to add functionality to the platform itself. Mobile devices have in their inventory various devices such as GPS sensor, compass, thermometer etc. We believe that using our initial mobile version of the Apatar server one could develop connectors that recieve data from these devices and use them to perform SQL queries on them. For instance, it would be useful for location based applications to send GPS indications to the server and include them to the queries that are excecuted.

Moreover, as mentioned apart from Android there are many more mobile device operation systems. It would be a good idea to develop an equal plaftorm for iOS, Windows Mobile or any other popular mobile OS.

All our work is based on open source systems such as Android and Apatar. We consider it our obligation to return our work back to the open-source community so that it can be used and improved by anyone. Therefore, we intend to open-source our system.

7 Conclusions and Future Work

References

[1] Apostolos K. Nidriotis, "Dynamic Web Service Mashups", Technical University of Crete, Department of Electronic and Computer Engineering, December 2010

[2] http://blog.teamgrowth.net/index.php/mobile-application-development/a-brief-introduction-to-mobile-application-development

[3] Andreas Brodt, Daniela Nicklas: The TELAR mobile mashup platform for Nokia internet tablets.

[4] Seongho Cho, Hyoungshick Kim, Dongshin Jung, Hoyeon Park: Dynamic Mashup Platform for Mobile Web Applications

[5] G. D. Lorenzo, H. Hacid, H. young Paik. Data Integration in Mashups. SIGMOD Record, 38(1), March 2009

[6] G. D. Lorenzo, H. Hacid, H. young Paik, B. Benatallah. Mashups for Data Integration: An analysis. Technical Report UNSW-CSE-TR-0810, 2008

[7] developer.android.com/guide/components/fundamentals.html

[8] D. E. Simmen, M. Altinel, V. Markl, S. Padmanabhan, A.Singh. Damia: Data Mashups for Intranet Applications. SIGMOD '08, 2008.

[9] D. F. Huynh, D. R. Karger, R. C. Miller. Exhibit: lightweight structured data publishing. WWW '07, New York, USA, pages 737-746, 2007. ACM.

[10] R. Ennals, E. Brewer, M. Garofalakis, M. Shadle, P. Gandhi, Intel Mash Maker: Join the Web. SIGMOD Record, 36(4), December 2007.

[11] R. Ennals, M. Garofalakis. MashMaker: Mashups for the Masses. SIGMOD'07, June 2007.

[12] M. Altinel, P. Brown, S. Cline, R. Kartha, E. Louie, V. Markl, L. Mau, Y. H. Ng, D. Simmen, A. Singh. Damia - A Data Mashup Fabric for Intranet Applications. VLDB '07, 2007.

[13] http://www.knowledgefolders.com