UNIVERSITY OF GOTHENBURG

Mapping Reference Architecture on Third Party Mobile Devices

- a Case Study from Volvo IT

BACHELOR OF SCIENCE THESIS IN SOFTWRAE ENGINEERING AND MANAGEMENT

MARYAM SEPASI

ELNAZ SHAHMEHR

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, June 2012

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet.
The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

Mapping Reference Architecture on Third Party Mobile Devices

- a Case Study from Volvo IT

Maryam Sepasi
Elnaz Shahmehr

© Maryam Sepasi, June 2012
© Elnaz Shahmehr, June 2012

Examiner: Helena Holmström Olsson

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

[Cover:
an explanatory caption for the (possible) cover picture
with page reference to detailed information in this essay.]

Department of Computer Science and Engineering
Göteborg, Sweden June 2012

Mapping Reference Architecture on Third Party Mobile Devices

- a Case Study from Volvo IT

Maryam Sepasi
Department of Computer Science and Engineering
IT University of Gothenburg
Gothenburg, Sweden
mary.sepasi@student.gu.se

Elnaz Shahmehr
Department of Computer Science and Engineering
IT University of Gothenburg
Gothenburg, Sweden
elnaz.shahmehr@student.gu.se

***Abstract*— The increasing demands on using mobile technologies, leads to the raise of lacking the unstructured mobile applications. Some companies have introduced guidelines and standards within their organizations to reduce the problems and augment the consistency between their software applications and reusing the components in different systems. This study presents the result of an exploratory case study at Volvo IT Company where Android and iOS platforms were mapped to the reference architecture. This paper presents the results by exploring mapping between the architectures and building a stand-alone application based on the obtained architectures. Also this paper can be studied as a guideline for mapping related mobile platform architecture.**

Keywords- Software Architecture, Android Platform, iOS Platform, Reference Architecture

I. INTRODUCTION

The need of using mobile technology is growing and becomes more powerful than before. Recently, the utilization of mobile operating systems has become extensive all over the world (Teng & Helps, 2010). Also, mobile phones are becoming a new popular platform for business applications. The number of users of mobile devices is increasing daily, and so is the need for efficient mobile data access and mobile application technology (Natchetoi, Kaufman & Shapiro, 2008). Therefore, the request for complex software applications in mobile device platforms has increased dramatically (Gasimov et al., 2010). Recently, since mobile applications are ubiquitous, many companies are adopting mobile technologies to increase their operational efficiency (Unhelkar & Murugesan, 2010). So, enterprise applications have been developing in different development tracks by many companies to improve their responsiveness and competitiveness, capitalize on the mobile revolution, and meet the new demands of customers (Unhelkar & Murugesan, 2010).

Although mobile technologies and applications introduce many new opportunities for enterprises, they also present new challenges (Unhelkar & Murugesan, 2010). They hold constraints such as limited power, network bandwidth, processor speed, and memory (Malek et al., 2009). Therefore, a traditional approach to enterprise applications and database design is not suitable for mobile devices (Natchetoi, Kaufman & Shapiro, 2008). Thus, making new applications, which do not suffer from the lack of standardization in their structures, is the new challenge for developers. A lack of structured mobile applications increases the cost and makes maintenance difficult.

Most companies have introduced many guidelines and standards within their organization, which cover different aspects of software development. For example, architectural principles, in order to save time, reusing the components, having more secure software, and raising the consistency between all the software applications developed in their system. Also, the companies cannot dictate their customers' choice in mobile devices; they aim for developing applications for most of the leading mobile platforms.

Research Question- In this case study, *we explore the challenges for mapping an existing reference architecture to third party architectures*, by investigating the differences between Volvo IT's reference architecture, Volvo Group Target Architecture (VGTA), and the software development kits for the Android and iOS platforms. It is significant for Volvo IT to find out if the components of VGTA can be reused for mobile application development. Therefore, this research aims to find a specific software architectural pattern for Android and iOS platforms. On the other hand, the functionality of the Software Development Kit for Android and iPhone is investigated and eventually the result will be mapped to VGTA based on the Volvo Group Architectural Principles.

This exploratory case study contributes in the following ways:

(i) By investigating the architectural differences of these two platforms, in order to assess the challenges of each platform and the similarities between them, since the applications must coexist effectively with the other systems within the organization.

(ii) By evaluating the possibility of having reference architecture templates for each platform. Accordingly, we represent the generic reference architecture for Android

and iOS that can be reused in designing the software architecture of different applications.

(iii) By developing an application for each platform, to evaluate the existence of the potential problems with the mappings.

Overview- The rest of the report is organized with the following sections: Section II introduces the Android and iOS platform models, the software architecture of each platform, and the challenges of each operating system. In section III, the research method used in this exploratory case study is going to be introduced. At section IV, the result will be concluded and explanation of the obtained result will be presented. Section V provides with discussion. Section VI, introduced the related work of this study and section VII, concludes the paper and presents the contribution done in this research.

II. BACKGROUND

This section will introduce the basic information, which required for this study such as: concept of quality attributes and software architecture and the relevant mobile platforms.

A. *Quality Attributes*

Quality attributes are requirements of the system that specify the system criteria, which apply to evaluate the functionality of the system, and are separate from the functional requirements (Microsoft Patterns & Practices Team, 2009a). A good system and a bad system can be distinguished by the quality attributes of them. Depending on the requirements, the architect should consider quality attributes throughout the design, implementation and deployment process (Bass, Clements and Kazman, 2003). Quality attributes describe the properties achieved within the system, for example, the response time from a user request should be less than 300 ms. Functional requirements often get the most focus in the development projects, but there are quality attributes, which drive the architecture. As Bass, Clements and Kazman (2003) argued, the focus on the software, which performs a particular function, can introduce some problems later on. For example, systems are always redesigned not because of functional requirements, but because of difficulties in maintaining them, extending them, being hard to use, having poor performance, and so on. According to Bass, Clements and Kazman (2003), there are three different categories of quality attributes, as follows:

- System quality attributes, including runtime and non-runtime quality attributes such as maintainability, performance, and security.
- Business quality attributes, such as costs and benefits.
- Architecture quality attributes, such as correctness and conceptual integrity.

B. *Software Architecture*

Architecture has emerged as a crucial part of the design process. It covers the constructions of large software systems (Bass, Clements and Kazman, 2003). The architectural outlook of the system is usually abstract, filtering away the system implementation details and focusing on the behavior and communication of the "black box" components (Bass, Clements and Kazman, 2003). Similar to many other constructions, the applications should be created on a solid foundation. The rapid improvement of the technologies for mobile applications causes design problems (Mazhelis et al., 2005). Poor architecture increases the risk of having unstable software, therefore it either cannot support the current and future system requirements or it will be difficult to organize and succeed in a production process (Microsoft Patterns & Practices Team, 2009a). In order to design a system, three different success criterions should be considered: user, system, and business goals. Moreover, these metrics should come into consideration in different areas of the software development process (Microsoft Patterns & Practices Team, 2009a).

Nowadays, Software Architecture is developed in the first stage of planning the software system, which composed a collection of different desired components. There is no commonly fixed definition of Software Architecture, but some agreed characteristics could be observed. Two popular definitions of architecture are as follows:

- "The software architecture of the program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them" (Bass, Clements and Kazman, 2003).
- Kruchten (2004) refers to the Rational Unified Process (RUP®) definition that an architecture is "the set of significant decisions about the organization of a software system: selection of the structural elements and their interfaces by which a system is composed, behavior as specified in collaborations among those elements, composition of these structural and behavioral elements into larger subsystem, architectural style that guides this organization"[1].

Although the explanations are somehow dissimilar, the large degree of similarities can also be seen, for example, both of them specify that the behavior and the structure of the architecture is influenced by the system environment and

1 Based on an original definition by Mary Shaw, expanded in 1995 by Grady Booch, Kurt Bittner, Philippe Kruchten and Rich Reitman.

their stakeholders (Eeles, 2006). Consequently, the structure of the system is the most often mentioned and essential characteristic in the definition of the software architecture.

Architecture can be served as a significant interaction, and also reasoning, evaluation and development tool for systems. The requirements of the system are the foundations to design the architecture, which is the result of a set of business and technical decisions. Although there are many influences depending on the environment in which the architecture is required to perform (Bass, Clements and Kazman, 2003). In the development process, the requirements make explicit the anticipated properties of the ultimate system, but not all of the requirements are concerned directly with those properties. As Clements et al. (2002) claimed, based on the different requirements and goals of the system, the assigned software architect determines if a component can be considered as a part of the architecture or if it can be left away.

C. *Architectural Styles*

An architectural pattern or architectural style is a set of principles, which provides an abstract framework for software systems (Microsoft Patterns & Practices Team, 2009a). On the other hand, a solution to a problem in a framework is a pattern, which includes a basic principle and construction schema that shape software systems and can help in reusing the design by making the key solution available for common problems. Different patterns can be utilized between distinctive components when designing the software systems (Buschmann et al., 1996). According to Microsoft Patterns & Practices Team (2009a), there are various styles of architecture, which are characterized by their main focus area characterizes them. This study covers two architectural patterns, which are common and applicable for Android and iOS applications.

1) *Service Oriented Architecture*

Microsoft Patterns & Practices Team (2009a) mentioned that Service-Oriented Architecture (SOA) is a set of rules and methodology for designing and developing software, which allows the functionality of the applications to be provided as a set of services. These services are loosely coupled because each of these services uses standard-based interfaces, which has no or little knowledge of the other services. Services make a schema available, and use a message-based interaction across interfaces. The services are application-scoped and not component or object-based. SOA style uses different protocols and data formats to transfer data information, which set the business process in the form of interoperable services. As Microsoft Patterns & Practices Team (2009a) claimed the SOA architecture style contains the following important principles:

- Services are autonomous
- Services are distributable
- Services are loosely coupled
- Services share schema and contract, not class
- Compatibility is based on policy

Likewise, Microsoft Patterns & Practices Team (2009a) argued that using the SOA architecture style result in the following key advantages:

- Domain alignment: it means reusing the similar services with the standard-based interfaces. Thus, it will make better technology and as a result the cost will be decreased.
- Abstraction: autonomous services can interact over a formal and well-defined convention, which delivers loose coupling and abstraction.
- Interoperability: The provider and consumer of the service can be constructed and organized on distinctive platforms, so they can work with each other without any restricted access.

Therefore, the SOA style can be considered if there is a need to reuse the appropriate services. Also, it can be used in the applications that constitute different services with a specific user interface.

2) *MVC Architecture*

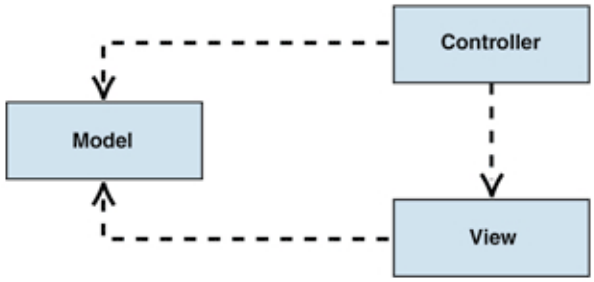

Figure 1. MVC Class Structure (Retrieved April 5, 2012, from: http://msdn.microsoft.com/en-us/library/ff649643.aspx)

According to Iulia-Maria & Ciocarlie (2011), there is a way for disintegrating an application to three modules, which is the Model-View-Controller (MVC) architecture. The three modules are: the model, the view and the controller. MVC was originally used for the graphical user interaction model of input, processing, and output.

Model: A model demonstrates the application's data including the access and operation logic of the data. The model object similarly includes any data, which is part of the persistent form of the application. The model releases a number of services, which should be standard and universal in order to support different users from different sectors.

Leff and Rayfield (2001) stated that the understanding of how to control the model's behavior should not be difficult, by looking at the model's public method list. The model groups linked data and operations together in order to provide an explicit service. These types of processes bind and abstract the functionalities of the business processes, which are being modeled. The model also implements compound processes inside it by presenting different approaches to access and bring the state of the model up to date. The controller accesses the model services in order to investigate or achieve a change in the model state. When a state change happens in the model, the model informs the view.

View: A view takes care of rendering the state of the model. The demonstration semantics are condensed into the view. Thus, one can apply one model data for a number of different users. According to Leff and Rayfield (2001) the view varies and adjusts itself when a change in the model is joined to the view, then the view forwards the user inputs to the controller.

Controller: Mazhelis, Markkula, and Jakobsson (2005) mentioned that the interception and translation of user inputs into actions are the main duties of a controller. Then, the model handles the actions. The selection of the next view, which is based on the user inputs, and the results of the model operation are also in the authorities of the controller.

D. *Android*

Google Android platform is a new generation of the mobile platforms, which was started by the Open Handset Alliance consortium on November 12, 2007[2]. It is a software package or software stack for mobile devices, containing an operating system, middleware and, main applications. The Software Development Kit (SDK) of android provides the required APIs and tools to develop applications using the Java programming language (Shu et al., 2009). Shu et al. (2009) mentioned that the Android platform supports a different kind of user skills, which has improved system graphics, media support, and a strong browser. For that reason, Android can be considered as the open, flexible, and adaptable system development platform. It allows reuse of components or element's replacement with making a database support available.

1) *Android Architecture*

The architecture and the major components of the Android platform are shown in Figure 2. As Shu et al. (2009) mentioned, architecture contains the following different layers:

- The red part represents the Linux kernel, which the Android architecture is based on. Android uses the Linux kernel as a hardware abstraction layer. The reason that Android uses the Linux kernel is because it provides a supported driver model. It also provides memory management, process management, a security model, networking, and different core operating systems for structures, which are robust and can be improved over time. The next level is contained native libraries. Everything that is illustrated in green is written in C/C++. This level includes a lot of core powers of the Android platform.
- The next level is Android runtime. The main component in the Android runtime is the Dalvik virtual machine. The Android runtime was designed specifically for Android to meet the needs for running the embedded environments where you have limited battery, limited memory, or limited CPU power.
- The next level is application framework. This is all written in Java, and the application framework is a toolkit that all applications can use. These applications include the core applications that come from the phone. Developers have full access to the same framework APIs used by the core applications. For example, the phone application includes applications written by Google as well as applications written by other developers. Consequently, all the applications use the same services in the same APIs.
- The final level is core applications. This is where all the applications get written and it includes SMS applications, calendar, and so on. Everything in this level is again using the same framework provided by the layers below.

2) *Developing an Android Application*

The first step to develop an Android application is to decompose the application into the components that are supported by the Android platform. As Shu et al. (2009) argued, there are four building blocks to an Android application. Not each application requires using all of them, thus the developer should choose what components are necessary for the application. These four major blocks contain the following:

- **Activity:** An activity is essentially a piece of the UI and typically corresponds to one screen (Shu et al., 2009). For example, an e-mail application decomposed to possibly three different major activities: one activity that lists the e-mails, the second activity to show the individual mail message, and the third activity to put together an outgoing e-

2 [Android Software Architecture]. Retrieved April 2, 2012, from: http://developer.android.com/guide/basics/what-is-android.html

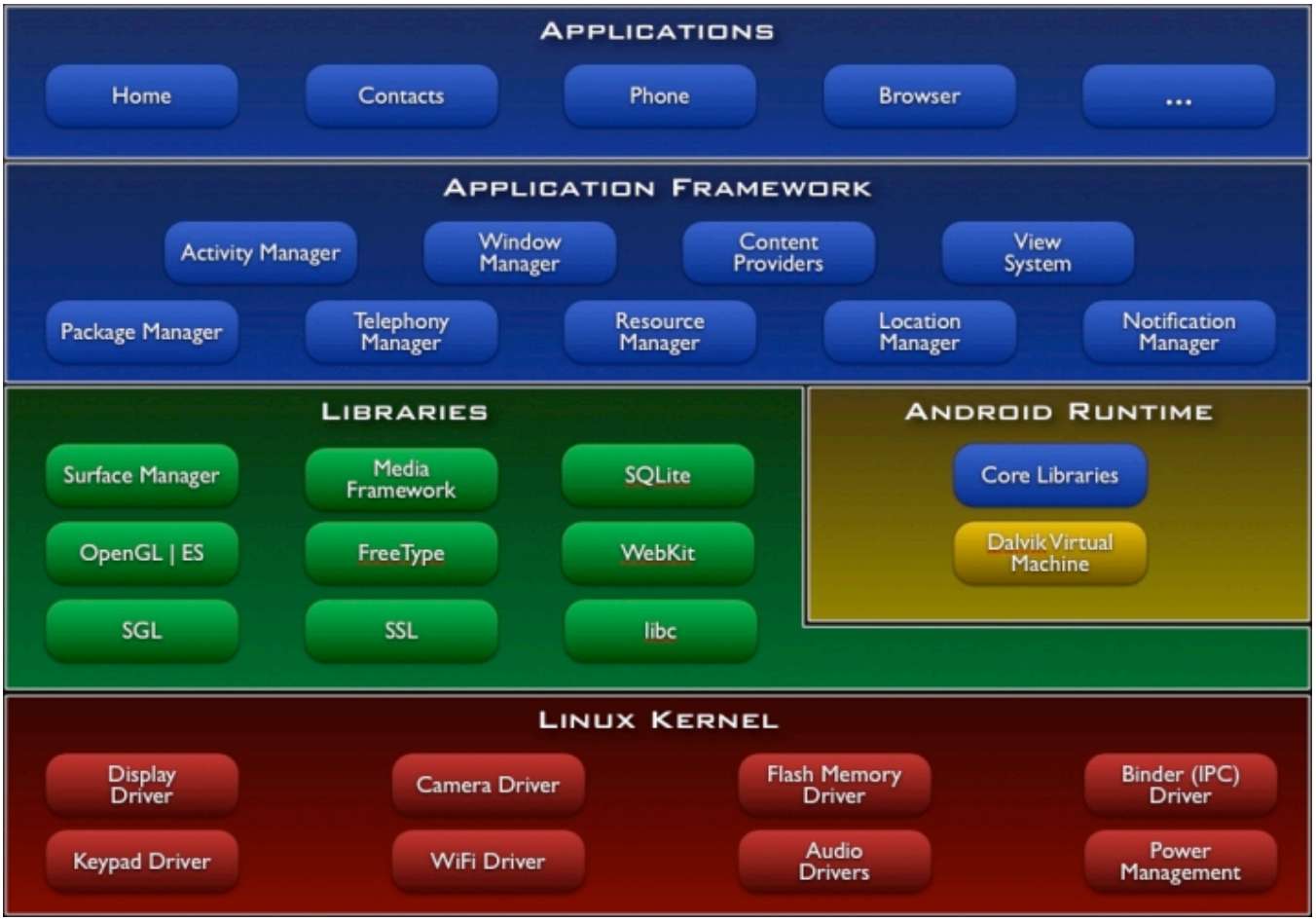

Figure 2. Android Architecture. (Retrieved April 12, 2012, from: http://developer.android.com/guide/basics/what-is-android.html).

mail. It should be mentioned that each activity usually has a user-interface schema, called a layout. Developers can define the layout structure and hold all the elements of the user interface in an XML file to appear to user[3].

- **Intent Receiver:** An intent receiver is a way, which applications register some code that will not be running until it is triggered by some external event. Therefore, the triggered code in the application always executes in reaction to an external event. Consequently, the developer can write some codes through XML and register it to be woken up and run when something happens or even works at the certain time, and so on.
- **Service:** As Shu et al. (2009) declared, service is a task that does not have any UI. It is long lived, and running in the background. For example, a music player; a user may start playing music from an activity, piece of UI, but once music is played, the user can navigate to other parts of user experience. For that reason, the code, which is actually running through the playlist, and playing song, would be a service running in the background. So, the user can connect to it later if they want to move from an activity to another by binding to the service and sending and getting messages like skip to the next song, and so on.
- **Content Provider:** Applications keep their data in files, SQLite database, preferences, or any other tool that makes sense (Shu et al., 2009). It is a component that allows developer to share data with other processes or other applications. Any application can store data, but if they wanted to have that data available as part of the platform in order to let the other applications make use of it, the content provider is a solution for that. As Shu et al. (2009) stated, Android was designed at the fundamental level to encourage reusing and replacing components.

Eclipse: Eclipse is known as an integrated development environment (IDE) for Java. Eclipse is produced by an Open Source community at 2001 and is used in several different

3 Google. (2012). XML Layouts. Retrieved April 10, 2012, from: http://developer.android.com/guide/topics/ui/declaring-layout.html

areas such as a development environment for Java or Android applications[4].

E. iPhone OS Platform

Apple iPhone OS (iOS) is the operating system created by Apple for all their mobile devices (iPhone, iPad, and iPod Touch)[5]. However, applications made for the iPad cannot be used on the iPhone and iPod Touch. IOS is first released to market in June 2007 (Hoog & Strzempka, 2011).

The SDK of iOS "enables you to create applications that run on specific versions of iOS or Mac OS X including versions different from the one you are developing on. An iOS SDK consists of frameworks, libraries, header files, and system tools"[6]. SDK of iOS contains native applications, which no longer needs a remote server like web-based applications and can run on an iPhone such as any built-in applications (Yan et al., 2011).

1) iOS Architecture

As Wentk (2011) mentioned, iOS architecture is based on MVC architectural pattern. Based on MVC style, an application is divided into three components: a data stock called a model, an interface called a view, and a controller object that passes data between the model and the view.

Iulia-Maria and Ciocarlie (2011) stated that iOS MVC suggests rather than make principles compulsory. The proposal is that the model data should be preserved in separate objects and possibly in separate custom classes. However, it still looks unreasonably complex in a simple application. Wentk (2010) cited that if your data is in a single array, it is not necessary to take out a data object of the appDelegate and put it into a separate wrapper object. It might be more useful in a larger application to keep data in a separate data-handling class. Nevertheless, the significant feature of iOS MVC is how the data is processed and made ready for demonstration by the controller not how the data is stored.

2) Developing an iOS Application

To develop an iOS application developers need to use delegation pattern, cocoa framework, and Xcode editor.

Delegation: Yan et al. (2011) mentioned delegation pattern is implemented using protocol and is widely used in iPhone application development. For example, the default delegate class in all iPhone applications conforms to the UIApplicationDelegate protocol. This class can be customized for application cycle events by implementing the various methods that UIApplication will call in prescribed order.

Cocoa: According to Wentk (2011) cocoa is a set of object-oriented frameworks that provides a runtime environment for applications running in Mac OS X and iOS. Cocoa is the famous application environment for Mac OS X and the only application environment for iOS.

Xcode: According to Piper (2009), "Xcode is Apple's powerful integrated development environment for creating great apps for Mac, iPhone, and iPad. Xcode includes the instruments analysis tool, iOS Simulator, and the latest Mac OS X and iOS SDKs". Xcode follows a guideline put down by Apple and it push limitation on designing the multiple applications. As "an application can generally only access files created by that application", thus "an applications can access certain other files such as address book data and photos, but only through APIs specifically designed for that purpose"[7].

F. Volvo IT Documents

In this section, by effort to maintain the Volvo IT's intellectual properties, we will describe two internal documents of Volvo IT.

1) Volvo Group Target Architecture

Volvo IT has created Volvo Group Target Architecture (VGTA), which is a generic architecture. VGTA also has been supported by the Volvo Group's IT Governance. It is an advanced architecture model including a number of components, and connectors that funding the components reliance. It is used as target architecture independent of platforms. It should be mentioned that VGTA is not following any specific architectural styles. All of the software architects at Volvo IT who design the systems in JavaEE or .Net deal with VGTA. By having VGTA the system's maintenance is easier, since this architectural pattern is universal within a company and it put a perfect separation between components. Thus the components can be reused by the other systems.

VGTA Concepts:

An Application is the top-level concept defined by the reference architecture. The concept of an application is typically defined by components belonging together and having the same life cycle. The lifecycle and ownership is also what defines the border of an application.

An application in turn consists of the following components, which serve a specific purpose:

4 Eclipse Foundation. (2012). About the Eclipse Foundation. Retrieved May 5, 2012, from: http://www.eclipse.org/org/

5 Apple Inc. (2012). IPhone. Retrieved April 22, 2012, from: http://www.apple.com/iphone/ios/

6 Apple Inc. (2010). Cocoa Fundamental Guide. Retrieved April 2, 2012,from:https://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html

7http://developer.apple.com/library/ios/documentation/FileManagement/Conceptual/FileSystemProgrammingGUide/FileSystemProgrammingGuide.pdf

- Domain Components: domain components contain the business rules and the information model associated with a particular business domain.
- User Interface (UI) Components: UI Components are responsible for accomplishing a user interface. This means UI presentation and UI workflow support functionality.
- Workflow Components: workflow components are only to be used when a workflow, which spans over multiple Domain/Proxy/Gateway Components, needs to be orchestrated. In most cases the workflow can be implemented in the UI- and Domain Components and be left out.
- Gateway Components: gateway components expose services provided by the application externally. It also transforms the information model from external to internal format. This component must not contain business logic.
- Proxy Components: proxy components provide external services to the application. They also transform information model from internal to external format. This component must not contain business logic.
- Utility Components: utility components hold generic functionality that may be used by all components of the application. Utility components do not depend on any other components of the application. Constants, common base classes, and common exception types may reside in this component.

2) *Volvo Group Architecture Principle*

Volvo Group IT Governance has established 10 architectural principles based on quality attributes, which are more important for Volvo IT. These principles should be wisely applied during the design of new applications. Simplicity in solutions and work methods, and maintainable solutions are a number of examples of these values. In designing the

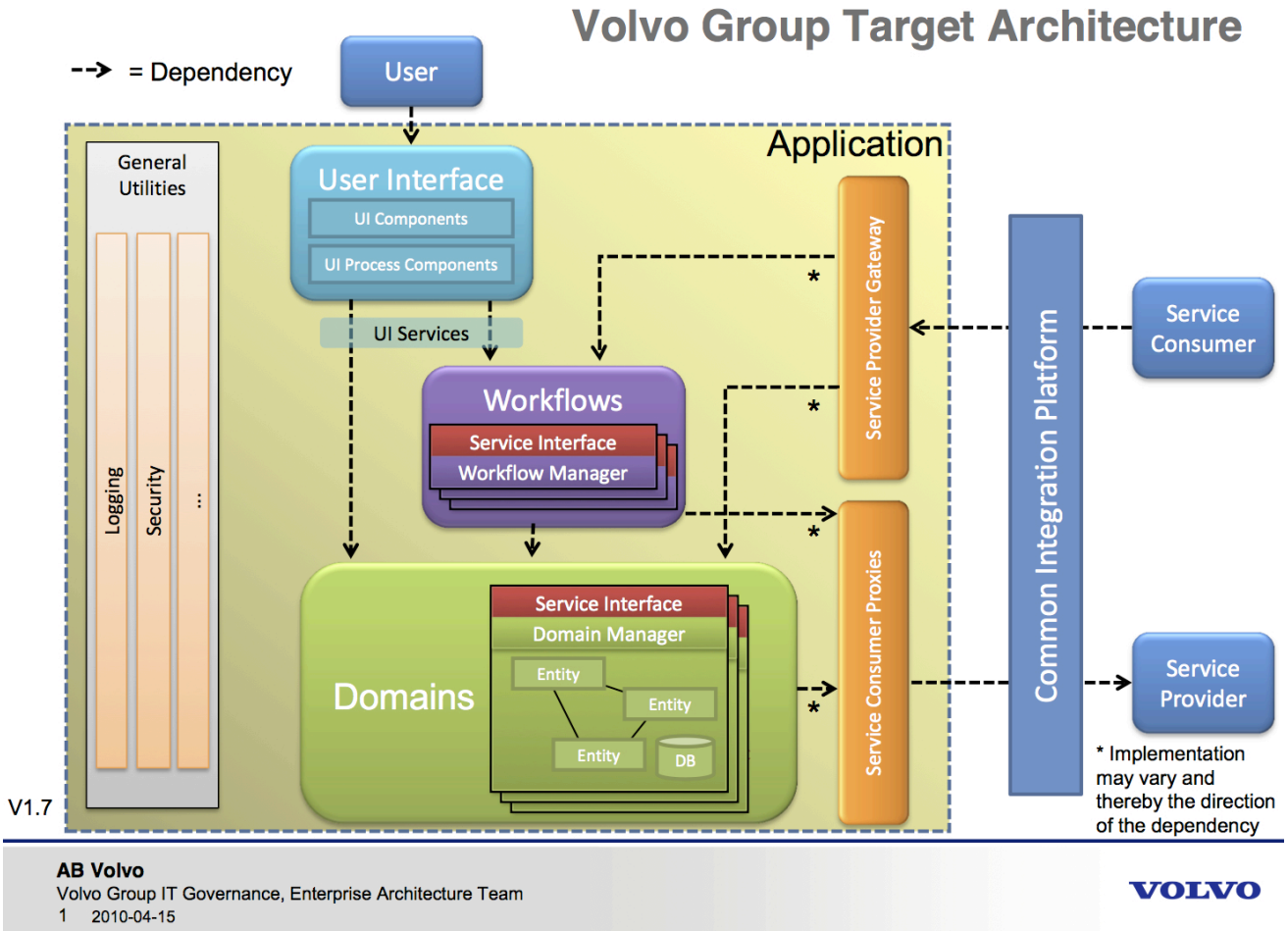

Figure 3. Volvo Reference Architecture

architecture and making decisions, architects should consider if the decision is well suited to the principles or breaks the principles. It should be mentioned some of these principles have contradiction therefore, based on the certain aspect of the system; these guidelines can support the architect.

III. RESEARCH APPROACH

This section presents the case for this research, which is from Volvo IT and provides a clear and complete picture of how this study was done and what steps we were taking in this study.

A. *Research Setting*

Two students from IT University of Gothenburg, in collaboration with Volvo IT, carried out this case study. The case study is "an empirical method aimed at investigating contemporary phenomena in their context" (Runeson & Höst 2009, p.12). This research is based on the case study methodology that focuses on the exploratory type. An exploratory case study centers on understanding the situation and finding out the insights within the contributors (Robson 2002).

For designing and developing software applications, Volvo IT, industrial partner of this study, utilizes some guidelines and criteria, for instance Volvo Group Target Architecture, VGTA and Volvo Group Architecture Principles, VGAP that have used for .NET and JavaEE development process. Volvo IT develops mobile applications for different platforms and they still want to use VGTA and VGAP in mobile platforms as well in order to keep standardization and lead time reduction within the company. Therefore, the focus of this study is to come up with specific software architecture for just two mobile platforms, Android and iOS, which standardize the designing of mobile applications in these two platforms. The reason that Android and iPhone platforms are chosen is the most of the customers of Volvo IT work with these two platforms.

The stakeholders of this research have verified all of the outcomes of this study and they have a position as IT architect, Android developer and iPhone developer experts at Volvo IT.

B. *Research Process*

The Research process of this study was split into four phases, which has been done in several iterations. As depicted in Figure 4 the research process consist of 4 significant steps, which are interviews, analysis, problems, and solutions. These are tightly related steps that collaborate with each other in numerous repetitions.

In the first iteration and at the early step, we studied the documents provided to us by the responsible person at the case study company, Volvo IT. Then, some structured and semi-structured interview was conducted to find out the stakeholders opinion. The next step was to discuss the obtained data and analyze it in order to find the problems, which was about the architectures should mapped with the same reference architecture. Thus the problems concerned the challenges on mapping Android and iOS platform's architecture to VGTA. The last step in the first iteration was to solve the problems that were mapping the architectures. In the other hand, at this iteration researchers tried to find out what is going to be problematic, what is going to be challenging or what is difficult to achieve. Authors went beyond record and analysis and draw a preliminary conclusion where they also gave their concern on the particular situation or concept been addressed. Then, the researcher came up with the solutions based on the literature review and the related works and all the collected data and analyzed data were applied. Literature review supported the researchers in finding main resources in mobile devices, software architecture of mobile devices. It also assisted the authors to disclose the constraints of the architecture of the mobile applications, which should be reflected to the research.

Accordingly, the second set of iteration was conducted; therefore the second set of interviews was conducted to see how does the solutions fit or what the stakeholders think about the solutions. At that point, the researchers came up to develop the applications for the mobile platforms. The application's requirements were the main problem at this stage. Then by developing the applications, the problems solved. Following that we verified them by having the third interviews. Consequently, the iteration is terminated due to the time limitation and getting solutions. Eventually, the stakeholders assess the outcome of this research study. If the stakeholders recommend any improvement that are out of the scope of the research study, it could be consider as a future work of this research.

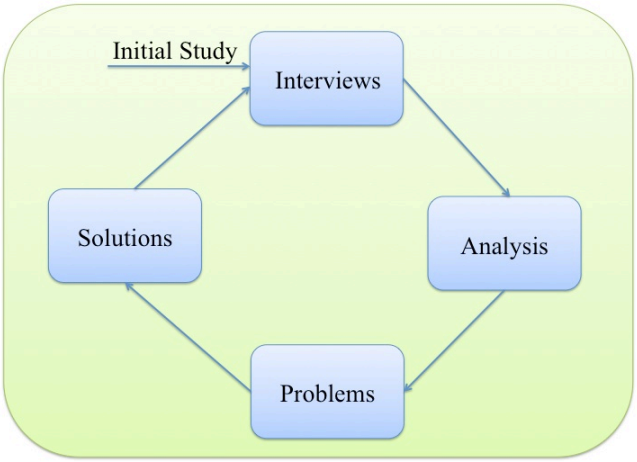

Figure 4. Research Process

C. Data Collection

The collected data was mainly gathered from structured and semi-structured interviews. First, we came up with the relevant questions and then divided into different groups according to the roles of interviewee to be interviewed. The first set of interviews was used to come up with the core of the concept and architecture, the second group to evaluate the mapped architectures and add more required thing. As a result, the second set of interviews was conducted where interviewees evaluate the architectures of two mobile platforms. The third set of the interviews was presented to see how the applications reflect the architecture and the evaluation of the implementation part.

It should be mentioned that some of these interviews were recorded and saved in audio files to be analyzed further. At the same time the researchers took notes in shared files.

D. Data Analysis Process

As Runeson & Höst (2009) pointed out because the case study research method is a flexible methodology, therefore the qualitative data analysis is generally used for this method. Keeping a clear sequence of the evidence while obtaining the outcome is the main purpose of this analysis. As Runeson and Höst (2009) argued, having analyzing data and the data collection in parallel keeps a clear chain of evidence and there is required to apply systematic analysis techniques. Hence in this study, the analysis carried out with the data collection because the approach of this research is flexible and new vision may is found within the data analysis. Collecting the new data and updating the interview questions can investigate these new vision and insights. The data analysis followed a format. The format consisted of three parts: (i) Share notes and recording, (ii) Analysis, and (iii) Summarize. Here are definitions the various parts of the data analysis done:

(i) Share notes and recording: The person who interviewed the interviewee ensured that an appropriate interpretation of the recorded interview was collected.

(ii) Analysis: After sharing the data an interpretation of the data into a meaningful content, which is then related to the objective of the project. This is rather not the same thing as record since we draw connection between ideas gathered from the interactive sections.

(iii) Summarize: Authors went beyond record and analysis and draw a preliminary conclusion where we also gave our concern on the particular situation or concept been addressed.

Since there were two persons carrying out the data collection at the same time, and anticipating the need to carry out a comparative analysis, it is important that the interpretive data follow a common format. After collecting the data, an interpretation of the data into a meaningful content, which is then related to the objective of the project, has been done. We draw connection between ideas gathered from the interactive section.

E. Proof of concept application

Part Order System (POS) is an example application within Volvo IT Company that the software architect deals with to check the validity of their designed architectures. POS application is not a system to go to the production phase. It has been used to prove the correctness of the upcoming architectures for JavaEE and .Net platforms. Therefore we developed POS application to validate the suggestion architecture in Android and iOS platforms.

In comparison with VGTA, POS included domain, which is contained part and order entities with a database of different parts and orders. It also contained graphical user interface with process component for ordering to present to the users.

F. Limitation

One of the limitations in this research was that there were not any related works regarding the mapping architectures in the academic databases for example Google scholar[8] and IEEE Xplore[9]. In order to accomplish an organized searching process, appropriate keywords were selected such as mapping reference architecture, mapping based on reference architecture, mapping mobile software architecture. There were not any related works, maybe because it is proprietary and no companies want to show their reference architectures in public. In a strategy to minimize this limitation, we had to interview different expert architects from Volvo IT staff as well as IT university of Gothenburg and got some details about architectural components in different architecture.

Another limitation to this study is that we could not work on an application, which communicate with different applications. As mentioned in section II.E.2, iOS guidelines tool pushes limitations and do not allow the developers to build mobile applications, which interconnect with each other.

In addition, due to the research limitation of this study, we just introduced two quality attributes, maintainability and loose coupling among both of mapping architectures and narrowed them down to a two choices, which is one of the limitations of this study.

IV. SOLUTIONS

As shown in Figure 5, stage (i) consists of interviewees who are experts on the software architecture. This step contains the interview process, data collected from the interviews, and

8 http://scholar.google.com/

9 http://ieeexplore.ieee.org/Xplore/home.jsp?tag=1

the construction of mapping the architectures, which is the primary objective of this study. The suggested reference architecture is affected by the both VGTA and VGAP. As mentioned in section II.F.2, the VGAP is a set of quality attributes principles that influence the building design process in the final architecture. In this study, not all of these principles are addressed, since some of them are not specified in the system requirements and also are not important for the stakeholders of this research. Eventually, the recommended architectures were proven and validated by the interviewees from Volvo IT. Step (ii) presented the relation of the suggested architectures with the developed application. In this step by considering the VGAP and the requirements of the system that were given by stakeholders, the implementations of the POS applications have been done. Therefore step (ii) is tightly related to the proposed architecture in step (i) and the final application corresponds to VGTA as well. The corresponding stakeholders who are experts in each platform prove the effectiveness of each application.

In this section the templates that present generic reference architectures are discussed. The authors believe that these templates can help the developers to develop the proper applications based on the templates. As depicted in Figures 6 and 7, these guidelines consist of several components and each of them represents the different groups of functionality. The architectures are defined by following the Android and iOS architectural models in VGTA to satisfy the need of the standardization design for various mobile applications within a company. Each expressed component in the architecture framework has its own particular transformation rules related to each mobile platform.

By building the applications, the authors achieved the confirmation of the right structure of the suggested architectures. The stakeholders of this study did the ultimate verification of this work. We developed a Part Order System (POS) mobile application in each platform to validate the related architecture. The detailed information of the application requirements cannot be shared by considering the restriction of sharing the belongings of the Volvo Company.

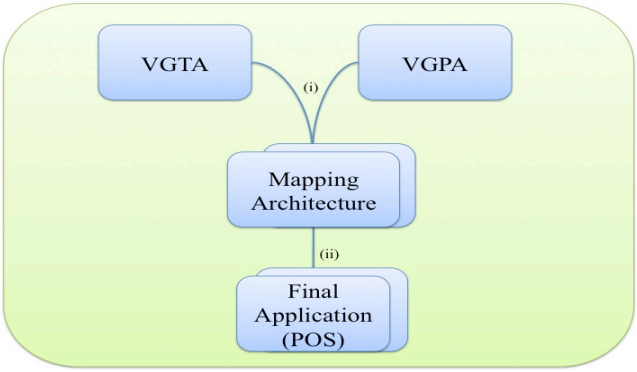

Figure 5. Building Construction

A. *Android and VGTA*

As presented in Figure 6, the building blocks of the Android application architecture are compared to VGTA components. Both architectures are similar to each other in some parts, since each of them consists of several components and relations between them. As mentioned in section II.D, Android SDK components interact with one or more components through the services that are published by the other components. Integration between components should be done in an organized way, where the types of the components and hierarchal aspects must be considered. In general Android architecture follows SOA architectural pattern, but since the stand-alone application framework is structured similar to the common Model-View-Controller architectural style. By presenting the similarities between the VGTA and Android architecture, we supply the rationale behind the mapping of these architectures.

As described in section II.C.2, the model module represents the application's data, the link's data, and also operations between them in order to provide an explicit service. Similar to this, the model in the Android application is also data or data storage. It can act like a local database in a device. In addition, as stated in section II.F.1 domain components in VGTA cover information model and rules related to that model. Therefore, the model module in the Android architecture should basically be the full domain components in VGTA and it will help make the domain model stronger, since it will decrease the need of using the different processes to deal with different objects. For example, in the POS application if the user wants to review the items in the cart, they can click on the "Edit cart" button and it triggers an event in the application. The application gets the appropriate data from the database and creates the required data to be sent back to the user.

According to definition of the view module in section II.C.2, it detects the state and accomplishes the demonstration of the information in the screen. Similarly the view module in Android, called layouts, is the part of the application responsible for rendering the screen and GUI. This contains the UI components and handles events for them. For example, the view module in the POS application might contain a component that indicates the list of user orders. User can interact with this layer and this layer activates actions that in turn are sent to the application functions. Based on the MVC style the controller is part of the application that reacts to external events. Likewise, based on the activities explanation in section II.D.2, activities cover the functionalities of the corresponding screens and take care of the logic that should be accomplished. In addition, activities prepare model that requires to be shown to view module.

There were a couple of challenges to come up with mapping the View and Controller modules to the User Interface (UI) module in VGTA. It all comes down to how we perceive

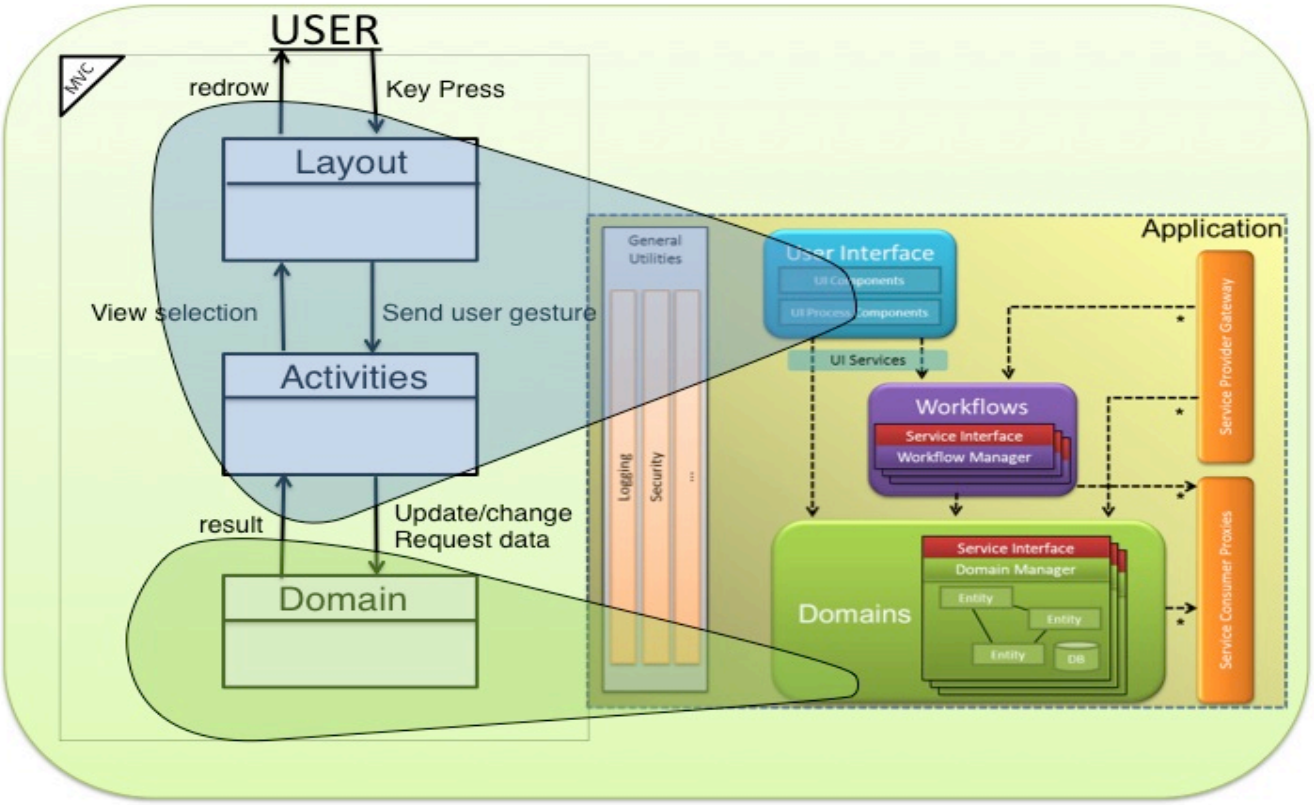

Figure 6. Android and VGTA

Android activity class. Is it a controller or is it a view? As mentioned before, the actual activity class does not extend Android's View class, but it handles displaying a window to the user and also handles the events of that window. Therefore, by using this mapping, the controller will actually be a view controller. Since it is controlling and displaying the window to the user with additional view components that developers can add to it, and it also controls events for several activity life cycle events. Because of this we have found that this mapping is a perfect fit to VGTA. By decoupling the model from the view and controller, we could support separate lifecycles between domain components and the UI. Therefore we could fulfill the loose coupling and increase the maintainability quality attributes.

B. *File structure in Android implementation*

Based on the suggested architecture and the requirements of the POS application, we have built an application for the Android platform. This construction represents the components of the suggested architecture for Android within the Android application anatomy boundary.

As mentioned in chapter II.D, Android SDK includes the main classes which developers should to manage while developing the applications. The POS application follows the mapped architecture and the architecture enables a clear division of the GUI part development from the development of the business logic (model) and distinguishes them. Building nice GUI needs different abilities than the developers have. Consequently, this suggested architecture is more suitable for GUI designers. Obviously, it would be better if a graphical designer design a GUI and a developer writes code and the logic behind the interface. But, the suggested architecture for Android overcomes this contradiction by clearly separating responsibilities. Since two persons worked on the application, so they could separate the GUI and logic part and work in separate area.

The POS application consists of the different folders and packages. As depicted in Figure 8, the most important folders in developing the Android application are the *src* and *res* folder and developers just deal with these two folders.

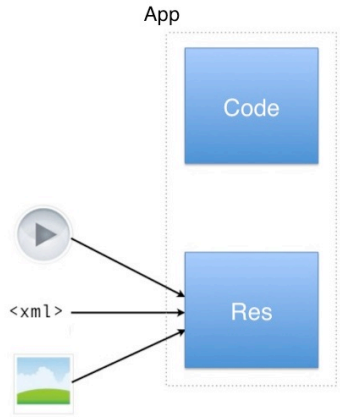

Figure 7. Android Build Design

As it shown in Figure 7, the authors considered the Android applications in two sections, Codes and Res. All off the text, pictures, and sound are broken out of the code in to the resource folder that is referenced in to the class Res. The packages consist of different classes. The Eclipse plug-in editor is used for developing the POS application for The Android platform. Since the POS application follows the mapped architecture, so some of the packages depend on each other, but each one exists on its own entities and performances a specific role. Each one is a unique building block, which helps the behavior of the entire application. Based on the mapped architecture, these different types of packages serve a distinct purpose and have a different lifecycle, which defines how theirs components is created and destroyed. The main challenges to follow the architecture was that in Eclipse Editor, developers cannot make the custom design for packaging codes and the entire written code packages place in the src folder. But, as mentioned above Android provides an alternative UI construction model: XML-based layout files which can be placed in the UIComponents section in the VGTA. As depicted in the Figure 8, the layout folder includes the entire GUI in XML files that it enables to better separate the presentation of the application from the codes, which control the behavior. So, it meets the loose-coupling requirements and simplifies maintainability. As mentioned before, the activity classes in Android handled displaying the windows to the user and the logic behind the interface. As depicted in Figure 8, all of the activity classes placed in the UI package. As adapter objects acts as a bridge between the UI and the domains, and provide access to the data, they act like a database manager in VGTA and are therefore placed in the domain component.

As depicted in Figure 8, we have not mapped anything to workflow, since we do not have an example of an application where there is a need to utilize the workflow. As mentioned in section II.F.1, there is no need to have a workflow until there is more than one domain component.

By following the suggested structure, a change from requirements will not affect complete source codes. Consequently, the suggested framework is proved and verified with the POS application.

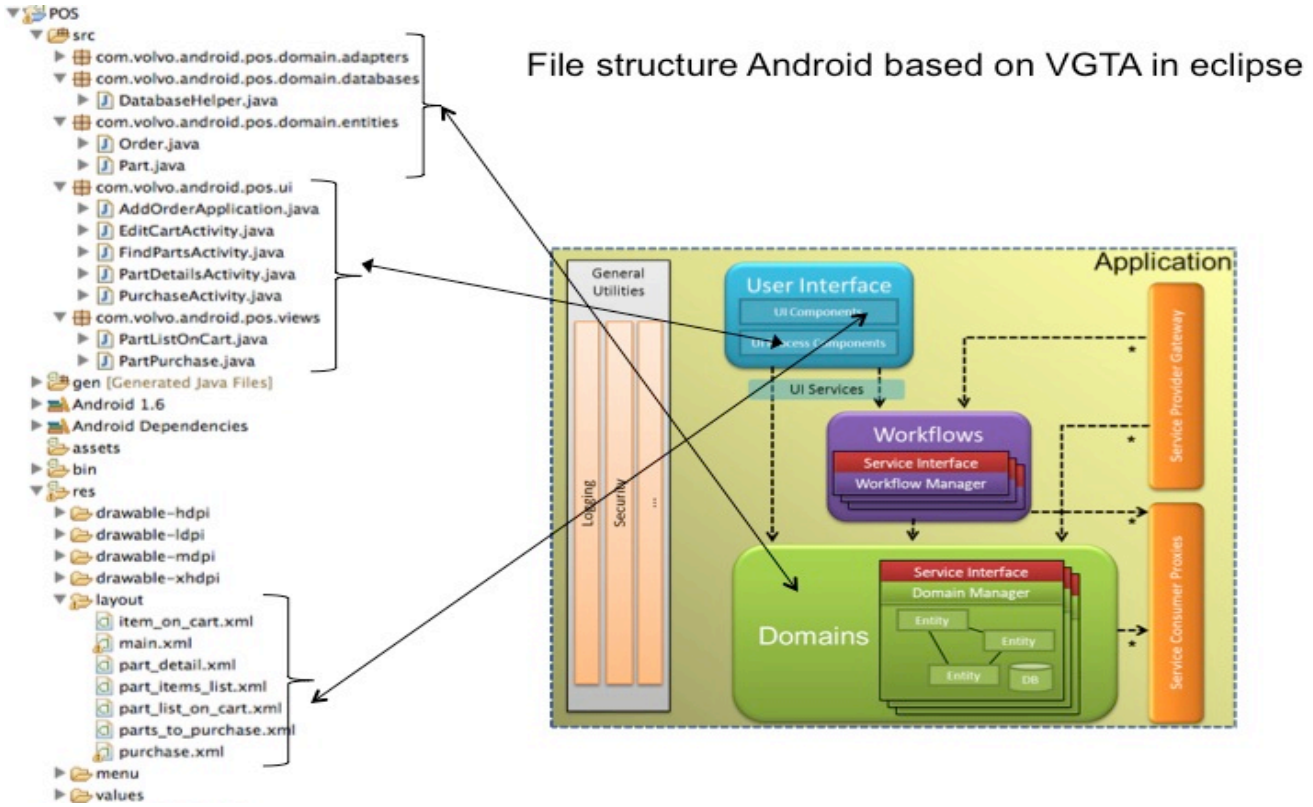

Figure 8. Android implementation Design

C. *iOS and VGTA*

Figure 9 depicted how MVC architecture in the iOS platform is mapped with VGTA. The main quality attributes, which are considered for this mapping, are the same as the introduced quality attributes for Android platform. Since maintainability and loose coupling are the important quality attributes for Volvo reference architecture.

As mentioned in section II.E.1, in iOS architecture the view module displays data from the application's model and the UIView component is the parent class for display objects on the iOS. Furthermore, as section II.F.1 shows, UI components in VGTA reference architecture are responsible for user interface demonstration. Therefore the UIView component acts like the UI component in VGTA. For example, View component in POS application contains a table with the list of parts. Then the part can be shown to the user. Accordingly, like a UI Process Component in VGTA, a UIViewController does not have a GUI; it simply coordinates the display of a UIView. Thus the actual pictures are going to need to be done in a UIView. In addition, the UIView is responsible for recognizing touches, gestures, and so on. That is where it ends though; the actual reaction of the program should be up to the UIViewController. Moreover, in section II.F.1, VGTA presents the UIProcess, which supports the functionality of components. Therefore these similarities supply the rationale behind the mapping of these architectures.

The model module in MVC architecture in iOS demonstrates the application's data including the access and operation logic of the data. As argued before, the VGTA domain contains the business rules and the information model as well, so we mapped the model module into the iOS architecture that encapsulates the specific data into an application. It also defines the logic and computation that manipulate and process the data with the model presented in VGTA. For example, there are parts and order entities in the POS application and each of them have different attributes like ID, price, and store.

It should be considered that the direction of these delegations is very important, because we wanted to be able to notify the controller about this delegation mechanism. For example, if some changes happen in the domain it can notify the controller that there has been a change and in the other direction when something changes in user interface we want to notify the controller that something has changed, so the controller is in the center and communicates in both directions. As depicted in Figure 9, the direction of the user action is down to the controller then the controller sends the update request to the Model.

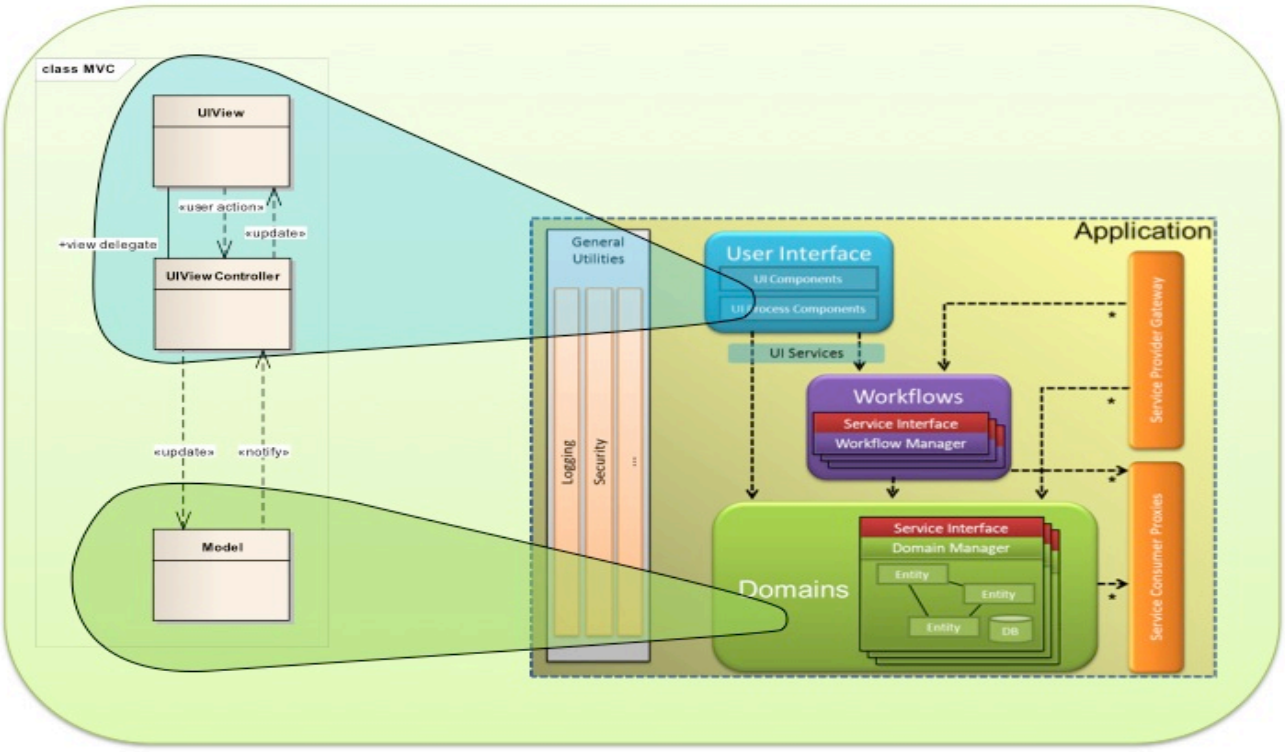

Figure 9. iOS and VGTA

D. *File structure for iOS implementation*

Figure 10 depicts the structure of POS application implementation in iOS platform based on VGTA. This construct also presents how similar the structure of VGTA and iOS reference architectures is.

As mentioned in chapter II.E.2, an Xcode developer tool provides everything that a developer requires to create application. Xcode is tightly integrated with CoCoa framework; therefore we used this environment to develop the POS application. Since this tool is flexible in a way of packaging the codes, it helped us to organize the packages based on their design schema. So we had no limitation to accomplish what we wanted to build. Therefore developers could have domain modules with all the entities, UI package, Proxy, and supporting files. These supporting files include the appDelegate that is described before. We attempted to consider appDelegate to be outside the VGTA because it is mainly class structure that Apple is promoting it to set up the application's structures in a good way and the case study company wanted to use the code in the other application and follow their own structure. But we should choose the appDelegate because it is the way to drive the framework, since the appDelegate only helped us to achieve the Apple feature around the suggested framework. As depicted in the Figure 9, the views and the view controllers extend from appDelegate, PartManager, and corresponding entities. Therefore the appDelegate is one way to transfer the data from domain to view controller.

As we did not worry about the workflow in the implantation Android application, we do not have an example on the iOS side where we actually need to use the workflow as well.

As depicted in Figure 10, the header file *.h* is where to declare different parts of your program. In this file, you will only declare the functions and global variables and *.m* (implementation file), which take care of the implementation part of all methods that are declared in the *.h* file of the program.

As mentioned in section II.E, iOS SDK is designed based on MVC design pattern. Thus, all the views always exist with a view controller object. Although the developer can mix the data model with MVC roles, the best way is to keep the division between roles. These separations increase the loose coupling of the objects and maintainability of the system.

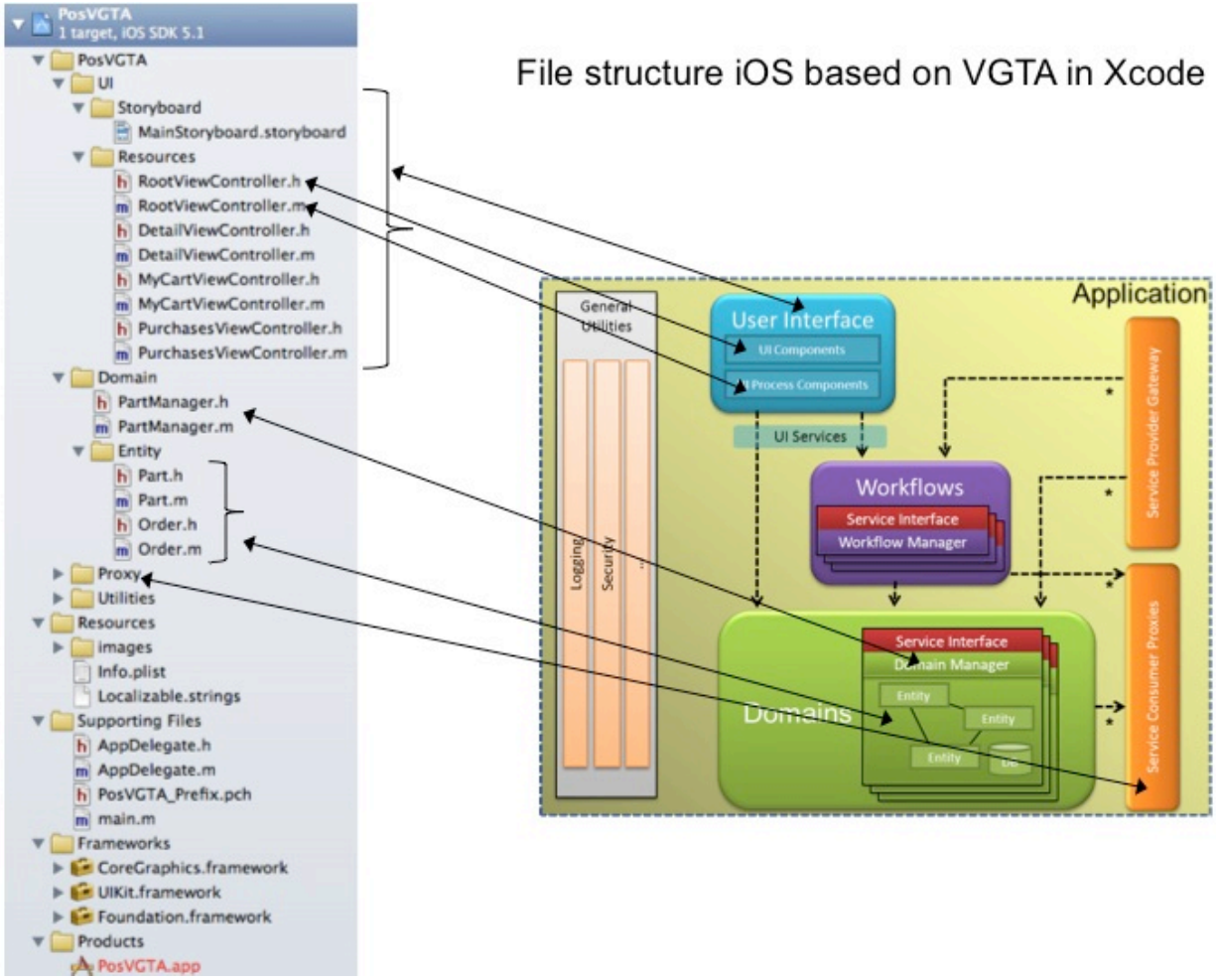

Figure 10. iOS Build Design

V. REFLECTION

This study set out to explore the challenges of mapping the existing reference architecture to the Android and iOS application architectures, to increase the consistency among distinctive applications. In this section, based on the author experience, some differences were found between the iOS and Android platforms will be described. While there are a lot of similarities between foundamental parts of Apple iOS and Android OS applications, there is a striking difference between architectural decisions on the application infrastructure layer made by the researchers of both OSs. Apple utilized Objective-C as a programming language and a runtime for iOS applications. Android applications are strikingly different in this aspect: they are written in Java, which is very different programming language than Objective-C. During the research, it was found that Apple tends to do every thing in their own special "Apple" way. The main problem that Android OS architects have to deal with is that Google does not control the hardware manufacture. Apple does not have this problem as iOS only runs on Apple-built hardware and they are in a complete control of it.

We gave a rationale concerning service-oriented approach for having different applications on both Android and iOS platforms. We obtained some key challenges of having several applications. As mentioned before, and as La and Kim (2010) argued, since mobile applications implemented on the mobile devices and the mobile devices themselves have limitations on their resources, mobile applications have inherited limitations and features, which do not appear in conventional software applications. By considering the Android OS and iOS features and the building blocks of them, we derived a conclusion that Android applications can communicate with each other with a service-oriented approach to develop mobile applications. As stated in section II.C.1, SOA allows the functionality of the applications to provide services to interact with each other. Also, as mentioned in section II.D.2, Content Provider in Android applications allows the applications to share data and connect to the other applications. By considering this feature, common and reusable functionalities can be modeled as a service and can be deployed by the content provider to the other applications. Therefore developing the complex and large applications is simpler on the Android platform. Whereas, iOS platform acts differently and reflects the different levels of openness of each application. iOS runs all applications as the same user and limits the interaction between applications. The iOS applications follow the MVC architectures and MVC architecture allows applications to interact with each other, but the developing tool, Xcode does not permit this freedom. It means that developers are not allowed to extend or modify the applications (Anvaari, 2010).

Based on section II.F.1, transforming data to the external services outside the application is done through the Proxy and Gateway components. As the scope of this study is limited to stand-alone applications, there is no need to consider Proxy and Gateway components in mapping the architectures and building the applications and it can be considered as a continuation of this work.

VI. RELATED WORK

To the best of our knowledge there is no related work on this topic. But in relation to the investigation of generic reference architectures, Torkabadi (2011) studied the feasibility of generic reference architecture for mobile devices by mapping reference models and architectural patterns. She investigated architectural artifacts, which are used at Volvo IT in different development tracks in order to find out the generic reference architecture templates for all mobile platforms. Her investigation discloses that it is not feasible to build a generic reference architecture that outfits all mobile applications' requirements. Thus, she exemplified four different mobile application scenarios and analyzed how to construct reference architecture for each of them.

Also, Autosar can be seen as one initiative to overcome the difficulties concerning diverging architectures. Within the automotive industry there are a lot of developers and subcontractors who share hardware. By having a common standard for the interfaces between different components, it is possible to ignore the different underlying architectures, since they are hidden behind the interfaces. Thus it means that a component can be replaced by another as long as they fulfill the same interface. Therefore, Autosar tried to hide differences between the architectures that agreed upon interface. The main difference is that they share the interface but VGTA does not share its interface with Android and Android does not share its interface with iOS, and there is no communication between these different arichtectures in our case. Therefore, there is a big difference between how it is done in Autosar and this research.

VII. CONCLUSION

This study set out to investigate the challenges of mapping Volvo IT's reference architecture, Volvo Group Target Architecture (VGTA), to the third party architectures, software development kits for the Android and iOS platforms.

To respond to the research question by investigating the related information and by studying the components of each architecture and the design process of them, we have found the architectural differences of Android and iPhone platforms and assessed the challenges of each platform and the similarities between them. So based on the similarities and the logic, explained in section IV, we mapped the

Android and iOS architectures to the existing reference architecture. Subsequently, based on the mapped architecture, an application for each platform was developed to evaluate the existence of the potential problems with the reference architectures.

By means of interviews, we obtained the company's point of view on how their system works, information from the literature review, and the outcomes from interviews, which were connected together based on the evaluation of the result and interviewee comments. In addition, we clarified a connection between the background and characteristics expected from each platform. For example, as mentioned in section II, we explained the components of the VGTA and how it is used within Volvo IT Company and in section IV, Android and iOS components compare to the VGTA components to get the better output.

We have focused on effective development of application processes by using the reference architecture at Volvo IT. The ability to save time, reusing the components and domains in different systems, and raising the consistency between all the software applications developed in production implies that our findings are likely important to developers and architects who are involved in building an applications, because our mapping is based on VGTA. Also, as mentioned in section II.F.1, it enables applications with reusing domains in various applications. By implementing the POS application we have found the benefit of using the reference architecture can be maintained while developing applications for third parties.

In terms of future research, we suggest a deeper investigation on the other quality attributes and research how they can fulfill all the architectural principles of Volvo IT. It may also be interesting to research the possibility of multiple applications in each platform, and investigate the challenges of communication between them. In addition, it would be perfect to consider other mobile platforms, such as Windows, Blackberry, and Symbian to find out the differences of them compared to Android and iOS, and also study challenges of mapping them to the reference architecture. Moreover, other companies that have similar architectural products to the involved reference architecture in this study can adjust their existing reference architecture with the result of this research.

ACKNOWLEDGEMENT

The authors would especially like to thank Micael Andersson and Håkan Burden for their constants supervision, advice, and feedback; Per-Anton Westbom, Stephen White and Samaneh Tork Abadi at Volvo IT for their excellent contribution to this research; Jason Brim for his valuable time and technical support; and also Helena Holmström Olsson, Carl Magnus Olsson, and Lars Pareto at IT University of Gothenburg. Finally many thanks to Solmaz Shahmehr, Asrin Javaheri and Mahdi Saadati for their very kind support throughout this study.

REFERENCES:

Anvaari, M. (2010). *Architectural Support for Openness in Mobile Software Platforms*, Master's thesis, University of Gothenburg, Gothenburg, Sweden.

Bass, L., Clements, P. & Kazman, R. (2003). *Software Architecture in Practice* (2nd ed.). Boston, MA, USA: Addison-Wesley.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M. (1996). *Pattern- Oriented Software Architecture, Volume 1, A System of Patterns.* Chichester, UK: Wiley.

Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Nord, R., & Stafford, J. (2002). *Documenting Software Architectures: Views and Beyond.* Boston, MA, USA:Addison-Wesley.

Eeles, P. (2006). *Characteristics of a software architect.* The Rational Edge, IBM Resource.

Gasimov, A., Tan, C. H., Phang, C. W., & Sutanto, J. (2010). *Visiting Mobile Application Development: What, How and Where.* Mobile Business and Global Mobility Roundtable (ICMB-GMR) (pp. 74- 81). Athens, Greece: IEEE.

Hoog, A., & Strzempka, K. (2011). *IPhone and IOS Forensics: Investigation, Analysis and Mobile Security for Apple IPhone, IPad and IOS Devices*. Elsevier.

Iulia-Maria, T., & Ciocarlie, H. (2011). *Best practices in iPhone programming: Model-view-controller architecture #x2014; Carousel component development*. EUROCON - International Conference on Computer as a Tool (EUROCON), 2011 IEEE (pp. 1 –4). doi:10.1109/EUROCON.2011.5929308

La, H. J., & Kim, S. D. (2010). *Balanced MVC Architecture for Developing Service-Based Mobile Applications. e-Business Engineering (ICEBE), 2010 IEEE 7th International Conference on* (pp. 292 –299). doi:10.1109/ICEBE.2010.70

Leff, A., & Rayfield, J. T. (2001). *Web-application development using the Model/View/Controller design pattern.* Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International (pp. 118 –127). doi:10.1109/EDOC.2001.950428

Mazhelis, O., Markkula, J., & Jakobsson, M. (2005). *Specifying Patterns for Mobile Application Domain Using General Architectural Components.* In Bomarius, F., & Komi- Sirvi, S. (Eds.), Proceedings of the 6th International Conference on Product Focused Software Process Improvement (Profes2005), LNCS 3547, (pp. 157-172). Berlin, Germany: Springer-Verlag.

Malek, S., Edwards, G., Brun, Y., Ta- jalli, H., Garcia, J., Krka, I., Medvidovic, N., Mikic- Rakic, M., & Sukhatme, G.S. (2009). *An Architecture- Driven Software Mobility Framework.* Journal of Systems and Software, 83(6), 972-989.

Microsoft Patterns & Practices Team (2009a). *Microsoft* R *Application Architecture Guide (2nd ed.).* n.p., USA:Microsoft Press.

Natchetoi, Y., Kaufman, V., & Shapiro, A. (2008). *Service-Oriented Architecture for Mobile Applications.* Proceedings of the 1st international workshop on Software architectures and mobility/ International Conference on Software Engineering (pp. 27-32). Leipzig, Germany: ACM.

Kruchten, P (2004). *Anontology of architectural design decisions in software intensive systems.* In 2nd Groningen Workshop on Software Variability, (pp. 54–61)

Robson C (2002) *Real World Research*. Blackwell, (2nd edition)

Runeson, P., & Höst, M. (2009). *Guidelines for conducting and reporting case study research in software engineering.* Empirical Software Engineering, 14(2), 131---164.

Shu, X., Du, Z., & Chen, R. (2009). *Research on Mobile Location Service Design Based on Android.* Wireless Communications, Networking and Mobile Computing, 2009. WiCom '09. 5th International Conference on (pp. 1 –4). doi:10.1109/WICOM.2009.5302615

Torkabadi S. (2011). *Towards a Generic Reference Architecture for Mobile Applications*, Bachelor's thesis for bachelor's degree, University of Gothenburg, Chalmers University of Technology, Gothenburg, Sweden.

Teng, Ch., & Helps, R. (2010). *Mobile Application Development: Essential New Directions for IT.* Information Technology: New Generations (ITNG) (pp. 471-475). Las Vegas, NV: IEEE.

Unhelkar, B., & Murugesan, S. (2010). *The Enterprise Mobile Applications De- velopment Framework.* IT Professional, IEEE Computer Society, 12(3), 33-39.

Wentk, R. (2010). Cocoa. John Wiley & Sons.

Wentk, R. (2011). *Xcode 4*. John Wiley & Sons.

Yan, B., Becker, D., & Hecker, C. (2011). *An effective way of introducing iPhone application development to undergraduate students. J. Comput. Sci. Coll., 26(5),* 166–173