UNIVERSITY OF GOTHENBURG

Towards a Generic Reference Architecture for Mobile Applications

Bachelor of Science Thesis in Software Engineering and Management

SAMANEH TORK ABADI

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
Göteborg, Sweden, May 2011

Towards a Generic Reference Architecture for Mobile Applications

SAMANEH TORK ABADI

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

Department of Computer Science and Engineering
Göteborg, Sweden

Towards a Generic Reference Architecture for Mobile Applications

Samaneh Tork Abadi
gustorksa@student.gu.se
IT University of Gothenburg
Software Engineering and Management
Gothenburg, Sweden

Abstract

Mobile devices come with a number of challenges that affect the design of the software architecture for mobile applications. Some of these challenges can be resolved by reusable architectural solutions. In this study, we investigate the feasibility of a generic reference architecture for mobile devices by mapping reference model and architectural patterns. Volvo IT has an existing reference model used for construction of reference architecture for Java EE and .NET development tracks; we explore the possibility of utilizing the outcome of this study at this company by considering their reference model. Since the proposed solution does not focus on any mobile technology, this study can be continued in further design details by considering different mobile technologies.

Categories and Subject Descriptors D.2.11 [*Software Engineering*]: Software Architectures

Keywords Software Architecture, Reference Architecture, Reference Model, Architectural Pattern, Mobile Software Architecture, Mobile SOA

1 Introduction

Many companies like Volvo IT have been developing enterprise applications in different development tracks for many years. Several guidelines and standards covering different aspects of software application development have been introduced internally in many IT companies such as development guidelines, and architectural principles in order to save time and increase the consistency among all the software applications developed within a company.

In recent years, large collection of devices with an increasing number of mobile operating systems has emerged (Teng & Helps, 2010), and mobile devices have become a new popular platform for business applications (Natchetoi, Kaufman & Shapiro, 2008). As mobile applications and technologies are ubiquitous and powerful nowadays, Volvo IT like many other companies is adopting mobile technologies for numerous applications to increase its operational efficiency, improve its responsiveness and competitiveness, capitalize on the mobile revolution, and meet new customer demands (Unhelkar & Murugesan, 2010).

Mobile devices introduce several new challenges like constrained resources, e.g., limited battery power, network bandwidth, processor speed, and memory (Malek et al., 2009). Fast development of the technologies for mobile applications also causes design problems (Mazhelis et al., 2005), and flexible adaptation to new technologies is required (Ahlgren & Markkula, 2005). Companies intention is that their products should be used by as many customers as possible, but given that they can not dictate their customers' choice in mobile devices, they aim for developing applications for most of the leading platforms. Although each platform tries to dominate the market by promoting applications offered in their own application store, none of the platforms have managed to impose their standards in the market (Gasimov et al., 2010). Different solutions such as native (platform dependent), cross-platform, and web-based exist for developing mobile applications; each of these solutions is based on specific architecture which may or may not be similar to the others.

Many companies that have commenced developing mobile applications, seek the expediency of using their existing standards and guidelines for mobile application development as well. Volvo IT, that is the industrial partner of this study, wants to investigate if the existing reference model and architectural principles which have been used to create the reference architecture for Java EE and .NET develop-

ment tracks inside the company, can be reused for mobile application development.

This study contributes in two ways:

- In order to address the described mobile challenges by an architectural solution and help a software architect during the design of a software architecture for a mobile application, we explore the feasibility of a generic reference architecture for mobile applications which is independent of any mobile technologies and can be used for any type of mobile application.
  This investigation reveals that it is not possible to construct a generic reference architecture that suits all mobile applications' needs. We conduct the study by exemplifying four mobile application scenarios and reasoning how to construct a reference architecture for each of them. These reference architectures and the rationale behind them lead to a generic reference architecture template for mobile applications that can be reused while designing a software architecture.
- We investigate if it is possible for Volvo IT to utilize the generic reference architecture template for mobile applications based on their existing architectural artifacts, especially their reference model. We conclude that their reference model is very similar to the one selected for this study. Hence, the outcome of this study can be used at Volvo IT without major modification. If other companies are interested in using the outcome of this study based on their existing architectural artifacts, similar investigation can be performed.

The remainder of this report is structured as follows: The concepts of software architecture, intermediate stages of software architecture design, and quality attributes as well as constraints of mobile devices, and related work to this study, are introduced in Section 2. We describe the research method that we have used to approach the problem in Section 3. In order to investigate the feasibility of a generic reference architecture for mobile applications, we provide hands on details of constructing this generic reference architecture and define four different mobile application scenarios to exemplify this construction in Section 4. We present the reference architecture template for mobile applications which is one of the outcomes of this study as well as explore the similarities with the provided reference model from Volvo IT (Section 5). Finally, we conclude in Section 6.

2 Knowledge Base

In this section, we present the knowledge base of this study that is produced through reviewing related literatures from various sources.

2.1 Software Architecture

There are many different definitions of software architecture and there is not one single universal definition that is deemed correct (Clements et al., 2002). What most of them have in common is that they include the topics of elements, structures, and relationships. However, one of the more popular definitions comes from Bass, Clements and Kazman (2003, p.21) who define software architecture as follows:

> "The software architecture of a 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. 'Externally visible' properties are those assumptions other elements can make of an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on."

Using the analogy of modern programming languages, this can be interpreted from an architectural point-of-view that the public interfaces of the elements are known but the implementation details and elements needed specifically for the implementation are not known. Thus, a software architecture is a definition of the software elements to be used, and the relationship between these elements. Buschmann et al. (1996) discuss that designing a software architecture addresses several aspects of software engineering such as technical, methodological, and process aspects. Clements et al. (2002) mention that it is always up to the assigned software architect to decide whether an element is part of the architecture or not, and he or she makes this decision based on the different goals of the architecture (e.g., behavioral or quality goals). The implementation details will be left in the hands of the system designers and developers.

2.2 Quality Attributes

A software architecture should address certain quality requirements on the software. Normally, these are non-functional requirements, even though they are affected by functional requirements. An example of this relation is given in (Bass, Clements & Kazman, 2003) where a functional requirement may demand to sort an enormous database, which means it can be impossible to achieve the required quality attribute of lightning-fast performance. This also shows that implementation, and architecture both need to address the quality attributes of an application.

A bad architecture can make certain quality attributes impossible to achieve during implementation and vice versa (Buschmann et al., 1996).

Albin (2003) states that a quality attribute is a property of a product or process that can have some quantitative or qualitative value, and that can be measured or observed. There is not one single list of quality attributes available, but rather several theories and models to compare. However, most of the models address fairly the same topics. Once again, one of the most popular quality models comes from Bass, Clements and Kazman (2003) that defines three categories of quality attributes: system, business, and architecture (Table 1). Not all of these quality attributes can be addressed by the software architecture alone and it is often up to the software architect to capture the quality requirements, since they are not always specified clearly in the system requirements (Albin, 2003).

Table 1. Classification of quality attributes by Bass, Clements and Kazman (2003)

System Quality Attributes	Availability Modifiability Performance Security Usability Testability
Business Quality Attributes	Time to market Cost and benefit Projected lifetime of the system Targeted market Rollout schedule Integration with legacy systems
Architectural Quality Attributes	Conceptual integrity Correctness and completeness Buildability

2.3 Intermediate Stages of Software Architecture Design

Bass, Clements and Kazman (2003) explain that design of a software architecture is started from bare box-and-line sketches, and is continued by binding of architectural choices during several stages. The final outcome of this process is a full-fledged architecture with all of the appropriate information about a system filled in. They describe reference model, architectural pattern, and reference architecture as three intermediate design stages that are very useful in their own right. Figure 1 illustrates the relation and order of these stages. We describe each one of these in the following subsections.

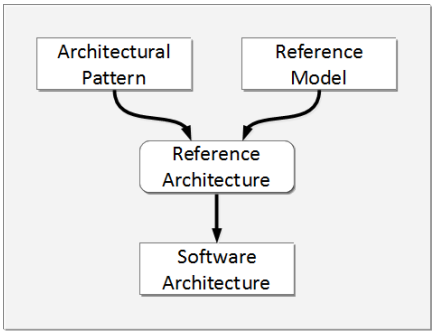

Figure 1. The relationship between reference models, architectural patterns, reference architectures, and software architectures (Adapted from Bass, Clements and Kazman (2003))

2.3.1 Architectural Pattern

Architectural pattern is a template for concrete software architecture that expresses a fundamental structural organization schema for software systems (Gamma et al., 1995). According to Microsoft Patterns & Practices Team (2009a), an architectural pattern is a number of principles that provides an abstract framework with solutions to well known and recurring problems. An architectural pattern adds certain constraints on the elements of an architecture, as well as on the relationships between these elements (Albin, 2003; Bass, Clements & Kazman, 2003). With the help of an architectural pattern, different stakeholders of a system can discuss the overall architecture using a common language without diving into technical details (Microsoft Patterns & Practices Team, 2009a).

An architectural pattern also highlights certain quality attributes such as known solutions to performance problems (Bass, Clements & Kazman, 2003). A software architect can combine several architectural patterns for a software architecture, when decomposing an architecture a number of different patterns can be applied between different elements (Buschmann et al., 1996). For instance, the relationship between a UI and a server element can be described using the client-server architectural pattern (Clements et al., 2002; Microsoft Patterns & Practices Team, 2009a), and the different server elements can be described using the layered structure architectural pattern (Clements et al., 2002; Fowler et al., 2002; Microsoft Patterns & Practices Team, 2009a). Thus, an architectural pattern can apply to a small group of elements as well as to a complete application. Other examples of architectural patterns include, but are not

limited to, Component-based Architecture, N-Tier Architecture, and Service-Oriented Architecture (SOA) (Table 2).

Table 2. List of common architectural patterns (Microsoft Patterns & Practices Team, 2009a)

Category	Architectural Pattern
Communication	Service-Oriented Architecture (SOA) Message Bus
Deployment	Client-Server N-Tier/3-Tier
Domain	Domain Driven Design
Structure	Component-Based Object-Oriented Layered Architecture

In this study, we cover some of the architectural patterns that we found during the literature review to be most common as well as applicable for mobile application architecture. The following architectural patterns are used in the construction phase, so we briefly describe them here.

Client-Server The client-server architectural pattern describes a distributed system with separate client and server systems connected by a network (Microsoft Patterns & Practices Team, 2009a). The simplest form of this pattern consists of one server with one or more clients. The most common use of this pattern is a web based application that is running in a web browser, and is connected to a web server. This type of client is referred to as a thin client as opposed to a rich client that is usually more complex, and can have for instance business and data layer on the client side (Fowler et al., 2002). van Gurp, Karhinen and Bosch (2006) illustrate the direct influence of three different architectural approaches that are client-server with native client, mobile Java client, and mobile thin client on the design, development, and deployment of an exemplified mobile application. Some of the quality attributes achieved while using the client-server pattern are centralized data access, higher security, and ease of maintenance (Microsoft Patterns & Practices Team, 2009a).

N-Tier/3-Tier Microsoft Patterns & Practices Team (2009a) describes this pattern as a separation of functionality over several tiers where each tier is deployed in separate nodes. By using this pattern to divide functions onto different tiers, resources can be used much more efficiently. In the case of a mobile application for example, a heavy calculation can be processed in one tier located on a server with the result being presented on the mobile device. By using this pattern, scalability, maintainability, flexibility, and availability quality attributes are addressed.

Layered A layered architecture groups related functions within an application into layers, where each layer consist of functions with something in common such as a common role or responsibility (Buschmann, Henney & Schmidt, 2007a; Microsoft Patterns & Practices Team, 2009a). The layered pattern promotes separation of concerns and supports flexibility, and maintainability. Each layer may be placed in a separate tier and be distributed on different physical computers as described by the N-Tier pattern. The layered architectural pattern comes with a number of quality attributes such as reusability, testability, performance, isolation, abstraction, and manageability.

SOA Microsoft Patterns & Practices Team (2009a) argues that the service-oriented architectural pattern promotes the functionality being exposed as services, which in turn can be used by applications using these services. The main principles of SOA are that services are autonomous (e.g., developed separately), distributable (i.e., located anywhere in a network or locally), loosely coupled (i.e., independent of others), and are compatible based on policy (e.g., protocol). SOA adds the quality attributes abstraction, discoverability, interoperability, domain alignment, and rationalization.

Message Bus This pattern describes software systems that can receive and send messages from/to other systems without knowing any specific details about the sender/receiver (Microsoft Patterns & Practices Team, 2009a). The pattern promotes using asynchronous message patterns like publish/subscribe for sending messages over a common bus. Benefits of using the message bus architectural pattern are flexibility, loose coupling, scalability, application simplicity, low complexity, and extensibility.

2.3.2 Reference Model

A reference model is a functional decomposition of a problem in the form of components and their connectors (Albin, 2003). It is a basic view of a problem and shows how that particular problem can be solved. It is used in conjunction with architectural patterns to shape a reference architecture. While we were reviewing the related literature, we found several reference models but in this report we present two models that we found interesting for this work.

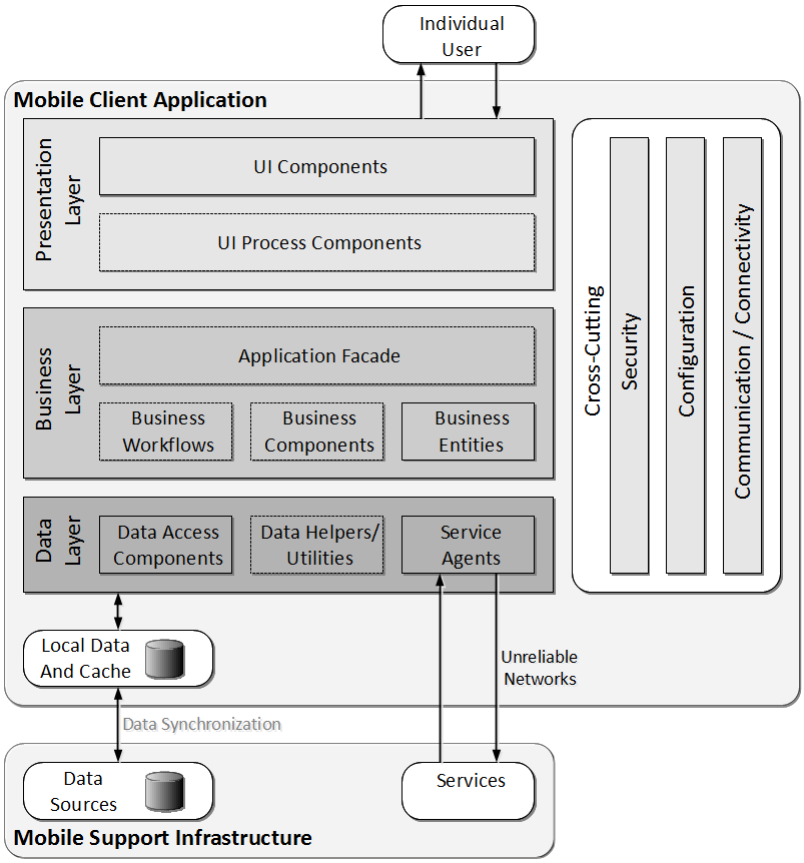

Figure 2. Common Mobile Application Architecture (Adapted from Microsoft Patterns & Practices Team (2009b))

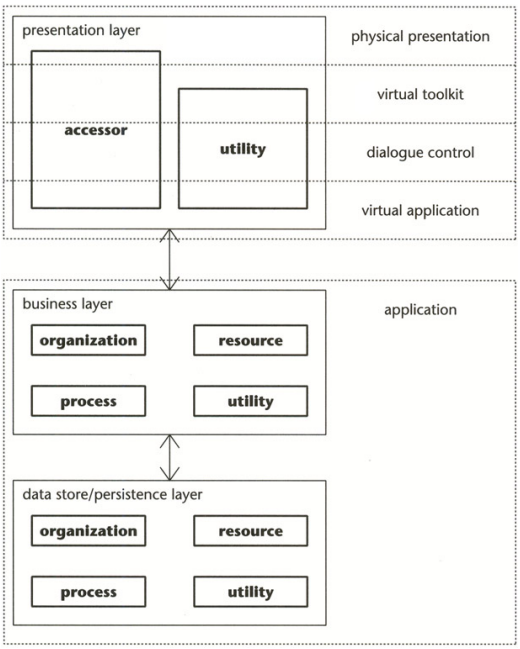

Figure 3. Enterprise Reference Model (Adapted from Albin (2003))

Figure 2 is the common mobile application architecture by Microsoft Patterns & Practices Team (2009b). This team presents a comprehensive architectural guide for designing mobile applications. It should be highlighted that this architectural guide holds a close resemblance to the Volvo Group Target Architecture (VGTA). The other example is Figure 3 from Albin (2003) which shows a set of components and the relationships between them.

2.3.3 Reference Architecture

Reference architecture is not a software architecture and is only the outcome of early design decisions. It provides a proven template solution for an architecture of a particular domain (Bass, Clements & Kazman, 2003). Reference architecture is the result of mapping reference model and architectural patterns, and is composed of software elements which jointly solves the problem defined by the reference model (Bass, Clements & Kazman, 2003). As a reference architecture can be defined at different levels of abstraction, it should be generic enough to be used in a family of systems, and not only for one specific system (Clements et al., 2002).

2.4 Design Pattern

While the architectural pattern provides a subsystem view of the application, the design pattern breaks down a subsystem into smaller architectural units, and their relationships (Buschmann et al., 1996). Design pattern provides an abstract description of a commonly recurring design problem in the form of a template that can be applied in many different situations (Gamma et al., 1995). Design patterns do not affect the fundamental structure of the software architecture, and are usually independent of programming language (Buschmann et al., 1996).

Ahlgren and Markkula (2005) address the challenges and constraints of mobile applications by utilizing design patterns. They illustrate synchronization and remote proxy patterns as two examples of many design patterns that can solve some of the constraints of mobile devices. Natchetoi, Kaufman and Shapiro (2008) also applied caching pattern in their work in order to address the limitation in availability of network connection. Furthermore, they describe two different approaches for the caching pattern, reactive and proactive. Reactive caching only caches information that has been requested by demand of the application, while proactive caching predicts what information the application will need next; the information is then pushed from the server to the local cache.

Roth (2002) presents a pattern hierarchy which is called Mobility Patterns (Figure 4). Each branch of this hierarchy addresses one mobile application development challenge, then one or several patterns for solving this problem are suggested. He elaborates more on two mobile application challenges that are mobile service usage, and mobile data. He suggests proxy and synchronization patterns for confronting these challenges. It is evident that mobility patterns are focused on a more detailed level, and closer to the mobile technology by involving design patterns.

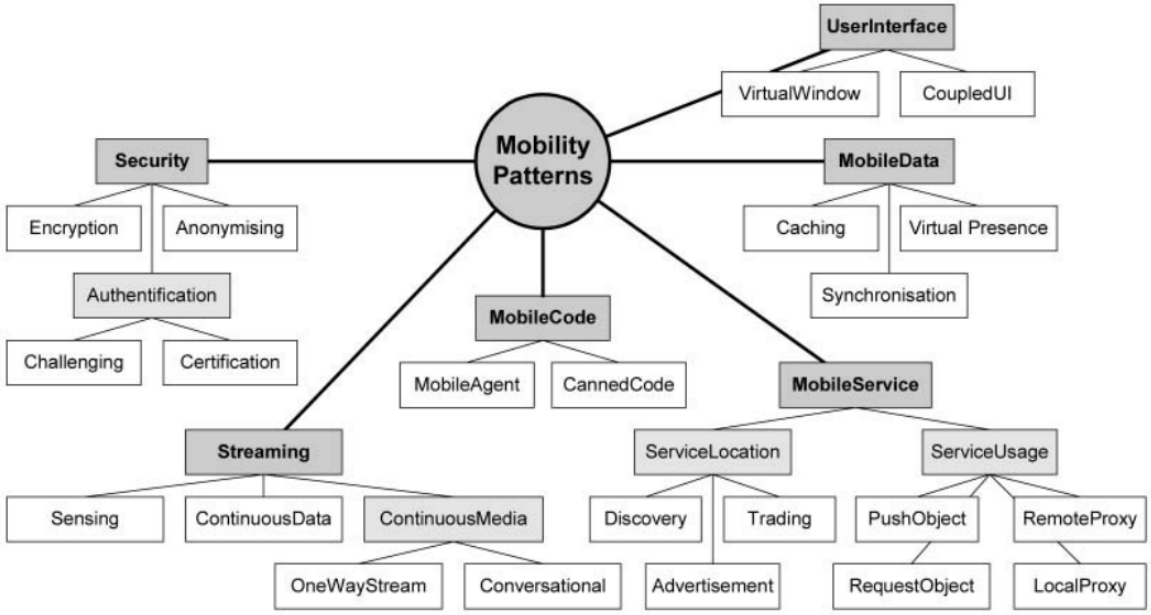

Figure 4. Mobility Patterns (Adapted from Roth (2002))

Microsoft Patterns & Practices Team (2009c) exemplifies a mobile application scenario, and deliberates on designing this example by suggesting the most suitable architectural patterns, and subsequently adding design patterns on each architectural unit. Although this work was only published on a technical web site, we decide to include this research since it is highly related, and has some unique concepts. It is worth to note that, in spite of the effort to find related work, this is the most important input for this study. This team explains the reasoning behind each architectural decision during all steps of design process for this example.

2.5 Characteristics of Mobile Devices

During the construction of an application for a mobile device, there are several challenges that are specific for these particular devices in comparison with most other target platforms. Some of these challenges can be limited battery power (Buschmann, Henney & Schmidt, 2007b), small screen size (Unhelkar & Murugesan, 2010), small and different input devices (Ahlgren & Markkula, 2005), limited memory and CPU capacity (Malek et al., 2009), security issues (Natchetoi, Kaufman & Shapiro, 2008),

platform diversity (Roth, 2002), different network protocols (Unhelkar & Murugesan, 2010), limitations of wireless network such as restricted bandwidth and availability (Mazhelis, Markkula & Jakobsson, 2005; Natchetoi, Kaufman & Shapiro, 2008), adapting to frequent disruptions in connectivity and service quality, and maintaining cache consistency across disconnected network nodes (Buschmann, Henney & Schmidt, 2007b). The mobile industry is also constantly evolving which puts higher demand on the software's ability to rapidly adapt to changes (Ahlgren & Markkula, 2005; Mazhelis, Markkula & Jakobsson, 2005; Roth, 2002). Due to these factors certain quality attributes such as modifiability, and performance (Mazhelis, Markkula & Jakobsson, 2005), may need to be carefully considered when constructing a software architecture for a mobile device.

2.6 Architectural Openness Model for Mobile Software Platforms

Anvaari (2010) introduces the architectural openness model for mobile software platforms that defines how much and under which conditions the application developers can access different layers and components of the platform and extend its functionality. This layered model consists of the following layers:

- Applications layer, this layer is divided into two sublayers which are:
  - Native Applications: applications that are developed by device manufacturer
  - Extended Applications: applications that are developed by application developers, and device users
- Middleware layer, this layer consists of main libraries and services of the platform such as data storage, and virtual machine.
- Kernel layer, this layer is the core of the platform that consists of the lower level components of the platform like device drivers, power management framework, and security framework.

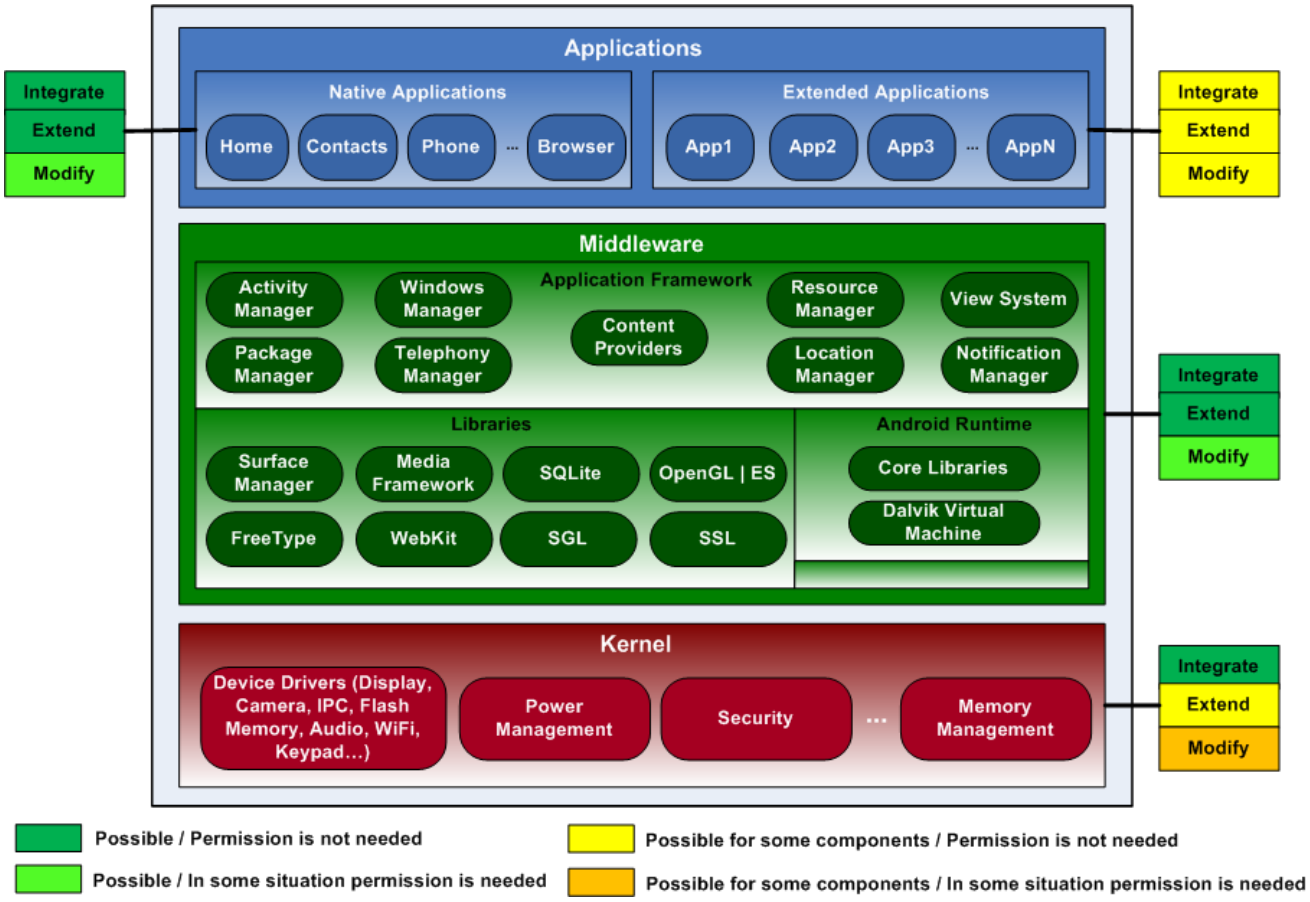

Figure 5. Architectural Openness Model for Android Platform (Adapted from Anvaari (2010))

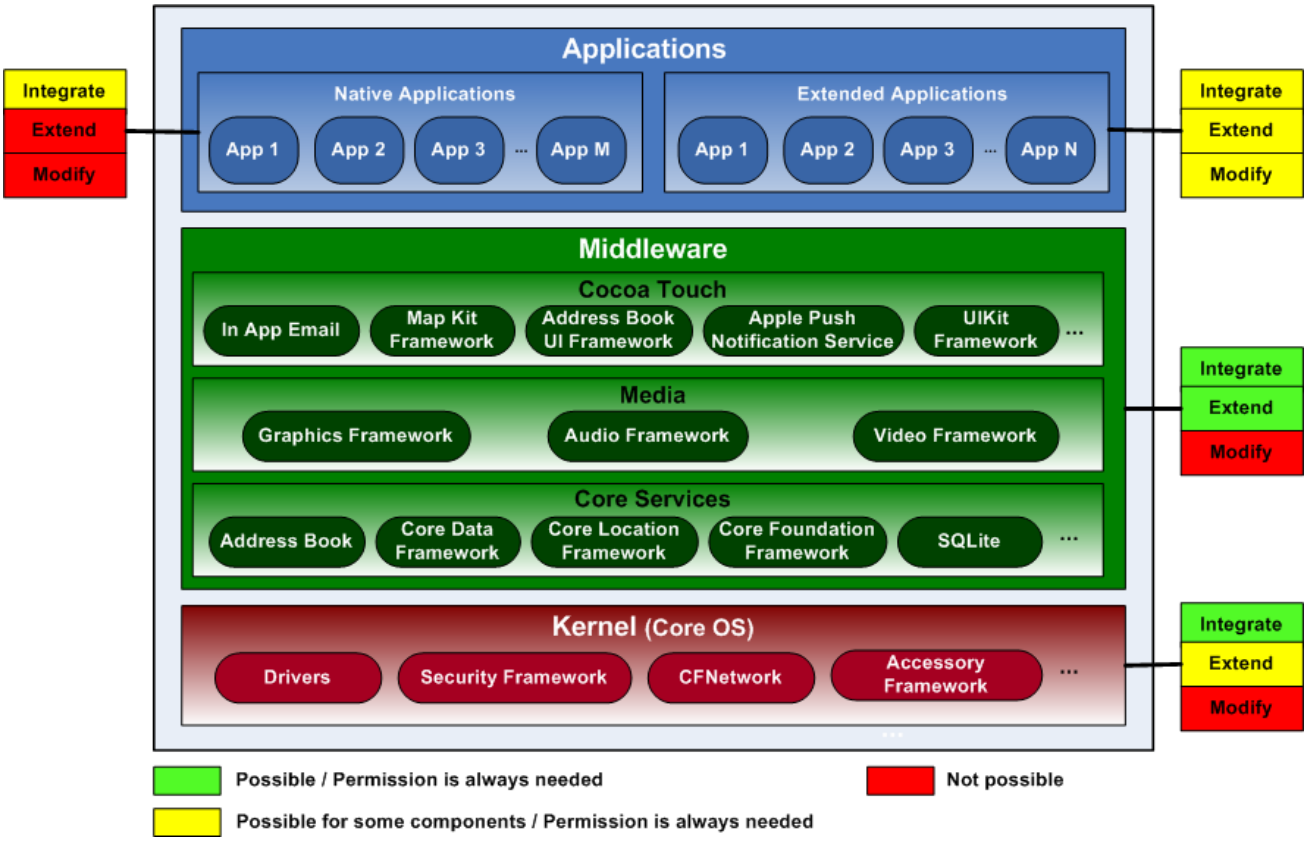

Figure 6. Architectural Openness Model for iPhone Platform (Adapted from Anvaari (2010))

In this model, Anvaari (2010) defines the concept of integrating, extending and modifying components in the mentioned layers as a means to clarify the openness notion in the architecture of the mobile platforms. By integrating a layer, he means to use existing components of a layer via e.g., API or service call; extending a layer refers to enhancing the functionality of the components of a layer and modifying a layer means to replace or change the components of a layer, for instance writing your own device driver. As the scope of this study is limited to Android and iPhone mobile devices, only the architectural openness model for Android (Figure 5), and iPhone (Figure 6) are depicted here. These two pictures show the openness of architecture for integrating, extending, and modifying in different layers of Android and iPhone platforms.

3 Research Approach

This section introduces the case for this study that is from Volvo IT (Section 3.1) and then describes all the provided input documents from this company (Section 3.2). Subsequently, we provide detailed description about the process of creating a generic reference architecture in Section 3.3.

3.1 Research Setting

This research is a result of 10 weeks qualitative study (Creswell, 2009; Seaman, 1999) and has been performed as a collaboration between IT University of Gothenburg and Volvo IT in Gothenburg. Volvo IT, which is part of the Volvo Group, is a global company that was created in 1998 but its root dates back to the 1920s with the introduction of punch card machines at Volvo (Volvo IT, 2011). Volvo IT has around 5000 employees that are located in different offices in more than 35 locations around the world. Volvo IT has active customers in approximately 60 countries. Volvo Group as one of these customers, is one of the world's largest suppliers of transport solutions and its customers are active in more than 180 countries.

Volvo Group IT Governance promotes some standards and guidelines for designing, and developing software applications. Volvo Group Target Architecture (VGTA) and Volvo Group Architectural Principles (VGAP) are two examples of these guidelines that are independent of any platform, and has been used by both Java EE and .NET development tracks in Volvo IT. Recently, Volvo IT has started to develop mobile applications for different mobile devices while still being active in developing software applications in Java EE

and .NET development tracks. Volvo IT is interested in using VGTA and VGAP as much as possible in mobile development track in order to maintain the consistency of software architectures between all three development tracks.

Volvo IT is a major stakeholder in this research since this company is interested in potential applicability of the artifacts produced during this study. The access to the intellectual property of the company in the form of the existing AVS, JVS, NVS, VGTA, and VGAP documents are all important contributions from Volvo IT to this research; although the author was required to sign a non-disclosure agreement (NDA) in order to have access to these sources. The author is required to be committed to this agreement while sharing the result of this study. However, it should be mentioned that nothing substantial relevant to the research question has been lost due to the NDA.

3.2 Volvo IT Documents

In this section, we describe the internal Volvo IT documents in general terms by considering the restriction of sharing the intellectual property of this company.

3.2.1 Volvo Group Target Architecture (VGTA)

Volvo Group Target Architecture (VGTA) is a generic architecture created within Volvo IT and promoted by the Volvo Group IT Governance. It is a high-level architecture with several elements, and connectors that shows the dependency between these elements. Using terminology presented earlier, the VGTA can best be described as a reference model. As it is mentioned in Section 3.2.3, the Architecture for Volvo Systems (AVS) was influenced by this reference model.

3.2.2 Volvo Group Architectural Principles (VGAP)

Volvo Group IT Governance has set up 10 architectural principles that should be carefully considered when designing new applications and application architectures. Some examples of these 10 principles are simplicity in solutions and work methods, and maintainable solutions. The VGAP can roughly be compared to quality attributes as described in Section 2.2.

3.2.3 Architecture for Volvo Systems (AVS)

Architecture for Volvo Systems (AVS) is a reference architecture that is influenced by both the VGTA and VGAP in pretty much the same way as any other reference architecture is influenced by reference model and architectural pattern(s) as was described in Section 2.3.3. In this case however, VGAP is not a set of architectural patterns, but rather a set of quality attributes that affects the design choices in the reference architecture. Figure 7 shows an overview of this relationship.

AVS as a reference architecture is used as base for the Java EE for Volvo Systems (JVS) and .NET for Volvo Systems (NVS) reference architectures, described in the following sections.

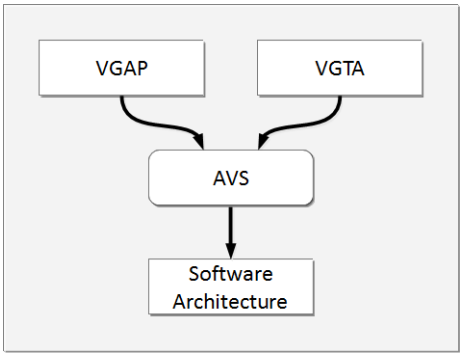

Figure 7. AVS Reference Architecture

3.2.4 Java EE for Volvo Systems (JVS)

Java EE for Volvo Systems (JVS) is the Java reference architecture implementation of the AVS. JVS further details elements and relationships defined in the AVS, and also provides solutions and documentation to common architectural issues that every software architect must deal with when designing a Java EE system. JVS provides detailed information about, for example layering, Java package names, where to use EJB's, and so on. JVS is used as a reference architecture for all Java EE systems built within Volvo IT. The relation to the AVS can be seen in Figure 8.

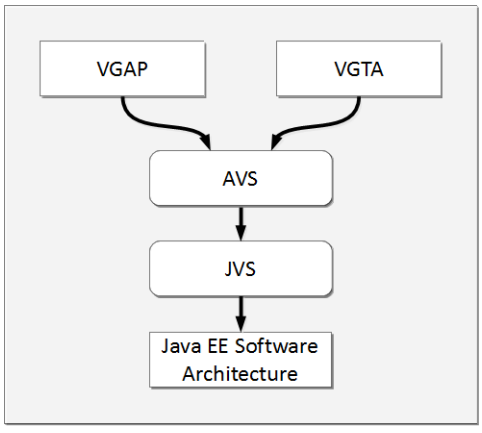

Figure 8. JVS Reference Architecture

3.2.5 .NET for Volvo Systems (NVS)

Similar to JVS, .NET for Volvo Systems (NVS) is a reference architecture for the .NET development track. NVS further details the AVS reference architecture and adds technical information applicable for .NET applications within Volvo IT. NVS relation to the AVS can be observed in Figure 9.

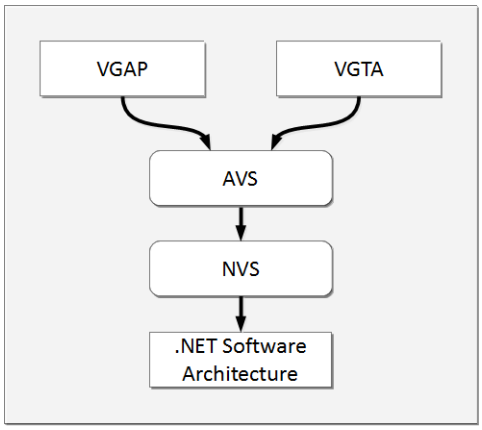

Figure 9. NVS Reference Architecture

3.3 Research Process

This study is approached as a design oriented research work which is fundamentally a problem solving paradigm. We believe that this research lies within the constructive research method (Kasanen, Lukka & Siitonen, 1993) as well as the pragmatic school of thoughts (Creswell, 2009) since this study is problem centered and aims for creating a solution for a known real world problem. Hevner et al. (2004) discusses that design research addresses unsolved problems in unique or innovative ways, and has its roots in engineering and the sciences of the artificial. This study utilizes the build and evaluate processes in several iterations in forms of assessment and refinement. Figure 10 illustrates how we adapted these processes to suit this research. The build process consists of two phases which are knowledge base production and construction of the research artifact.

Knowledge base production is divided into two steps that are data collection and data analysis. These two tightly related steps are collaborating with each other in many iterations before the construction phase begins. When the research artifact is built during the construction phase, all the collected and analyzed data from the previous phase is applied. The two phases are iterated when it is required in order to gain enough knowledge to build the research artifact. The produced research artifact is assessed by the stakeholders of this research work in the evaluation process and if any refinement is needed, the build process is revisited. The iterations between the build and evaluate processes are finally terminated due to the defined scope and time limitation of this study. If the stakeholders of this thesis work suggest any refinement which is out of the scope, it can be considered as a continuation of this work.

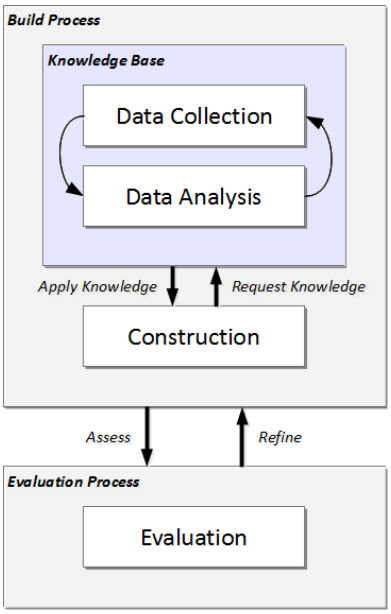

Figure 10. Research Process

3.3.1 Knowledge Base Production Approach

In this study, a set of information about software architecture in general, a collection of previous studies related to mobile application architecture as well as a list of suitable architectural patterns for mobile applications are collected by means of literature review. In order to perform a systematic and organized literature review, appropriate keywords are identified such as software architecture, reference architecture, reference model, architectural pattern, mobile software architecture, and mobile SOA.

Related research in this area has been published in different outlets such as books, journals, conferences, magazines, blogs, web sites and technical reviews. This study collects data from literatures in the form of books, journals, conferences, and a few technical web sites. It should be noted that a group of academic databases are selected for gathering related papers from journals and conferences, for instance IEEE Xplore, SpringerLink, Elsevier and ACM. As mobile's world changes rapidly, we decide to choose materials that were published after year 2001. However, there are some exceptions that are not directly related to mobile application design and development; these sources are either

related to the basic concept of software architecture or the selected research methodology for this study. Apart from the books in software architecture field that are used as the foundation of this study, we also search for relevant papers. The review process of these papers is started by reading the abstract section, and if the author find it interesting and related to the research study then it is fully reviewed. Literature review assists in discovering major sources in software architecture, mobile fields, and also defining the nature, scope and structure of the knowledge in these fields. Besides, it helps to reveal the challenges and constraints of mobile applications that should be considered in this study.

References used by many authors, including but not limited to (Bass, Clements & Kazman, 2003), (Albin, 2003), (Ahlgren & Markkula, 2005), (Roth, 2002), (Microsoft Patterns & Practices Team, 2009a), (Microsoft Patterns & Practices Team, 2009b), and (Microsoft Patterns & Practices Team, 2009c), are found to be tremendously useful for this study.

3.3.2 Construction Approach

The construction phase of this study begins during knowledge base production phase since it acts as an infrastructure for this construction. According to Figure 1, the construction of a generic reference architecture proceeds with identifying and choosing the most suitable reference model and architectural patterns for mobile applications, and mapping them onto a reference architecture. While producing the knowledge base of this study, we found out that mobile constraints and challenges also affect this construction but due to research limitation of this work, we narrowed them down to a few choices. We used all these choices and bound them together in order to assess the construction of the reference architecture.

3.3.3 Evaluation Approach

Unlike other research methods that collect empirical data, constructive research method considers the validity of the produced outcome in a practical example as its empirical data. Before evaluating the produced outcome of the study, fail and success states should be defined. According to Kasanen, Lukka and Siitonen (1993), a successfully produced outcome is the one that satisfies all stakeholders of the study as well as fulfilling the needs of a real world problem. The main stakeholder of this study is Volvo IT which verifies whether the produced outcome is satisfactory by testing this generic reference architecture in different possible scenarios that mobile applications should handle.

4 Reference Architecture Construction

In order to investigate the feasibility of a generic reference architecture for mobile applications, we thoroughly describe the procedure of constructing a generic reference architecture for mobile applications along with introducing all the influential factors on this construction. After discussing the result of this investigation, this section is conducted with defining four different mobile applications scenarios in order to construct a reference architecture for each scenario. As it is mentioned in Section 3.3.2, this construction is built upon the knowledge base of this study which is a result of data collection and analysis in several iterations.

4.1 Influential Factors in Construction

The software architecture helps both software development, and maintenance being productive, and it should be emphasized that there is a direct correlation between the quality of an application and its software architecture. Hence, the software architecture should be designed properly to fulfill all the expected requirements for an application, and address possible challenges. However, creating a software architecture from scratch for every system can be a time-consuming task. A software architect strives for reusing as much as possible from both knowledge gained from his or her earlier work and also from the experience of other people. As described in Section 2.3, the process of designing a software architecture normally begins with drawing a rough outline based on requirements from different stakeholders, and then enrich the design using different architectural choices in several intermediate stages. The software architect may choose to build his or her architecture based on a reference architecture, which in turn is influenced by reference model and architectural patterns, in order to save time and cost. Besides, using experience and lessons learned from previous similar projects can be beneficial while designing the software architecture for a new project.

In this section, we discuss the process of constructing a reference architecture for mobile applications that aims to address some of the constraints of mobile devices that challenge mobile application development. It is worth to mention that this reference architecture is intended to be built irrespective of any mobile technologies. Therefore, we aim to describe how to construct a generic reference architecture that is applicable for any mobile development solution. The desired outcome is a generic reference architecture that can be specialized for different target mobile development solutions in the next level as shown in Figure 11.

As described earlier in Section 2.3, reference model and architectural patterns are mapped onto reference architecture. Hence, the major influence of reference model and archi-

tectural patterns on reference architectures is obvious. For mobile devices, these are not the only factors that affect a reference architecture. While producing the knowledge base of this study, we realized the important role of mobile constraints and challenges during the design of a reference architecture for mobile applications.

In the rest of this section, we gradually describe the necessary steps for constructing a reference architecture for mobile applications using the described influence factors. In each step according to the knowledge base of this study, we make one or more architectural decisions and discuss the rationale behind them. During several steps, these architectural decisions are bound together and subsequently direct this investigation towards a conclusion whether it is possible to construct a generic reference architecture for mobile applications.

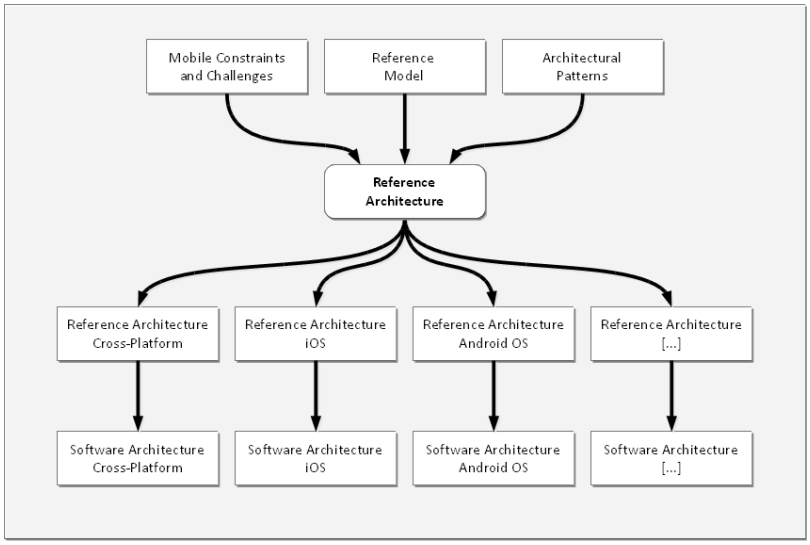

Figure 11. The relationship between mobile constraints and challenges, reference model, architectural patterns, reference architectures, and software architectures for mobile applications

4.1.1 Constraints and Challenges of Mobile Devices

In addition to the many benefits that mobile devices brought for their users, there are still several challenges and constraints that should be kept in mind while designing mobile applications. In Section 2.5 of this report, the characteristics of mobile devices are described in detail. In this study, we find network availability, limited bandwidth, limited memory, and limited battery power the most common and important challenges among all mobile devices. Estimation of battery power usage per different types of transactions is out of the scope of this study due to time limitation of this thesis work. Also battery power usage is mainly dependent on the target mobile device and technology, and as such is hard to address in a high level reference architecture. Figure 12 displays that our focus in this study is limited to network availability, limited bandwidth and limited memory.

As it is illustrated in Figure 12, mobile applications experience three states of network connection that are connected, occasionally connected, and disconnected. The term limited bandwidth deals mainly with the limitations in speed, but can also include limitations on the amount of data that may be transferred due to different cell phone operator plans. However, technologies such as 4G makes the speed limitation less of an issue. Mobile devices normally comes with a limited amount of memory, both RAM memory to use during execution but also for storage, so for example you may not be able to store a complete database on the mobile device.

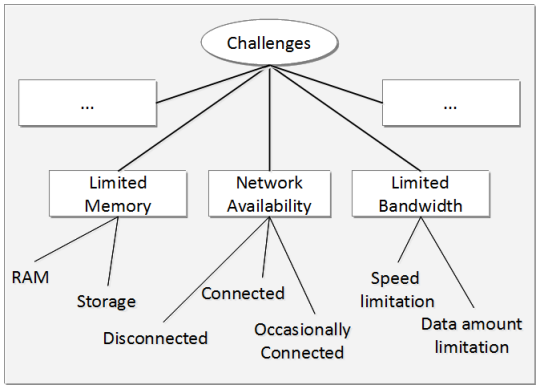

Figure 12. Constraints and Challenges of Mobile Devices

4.1.2 Reference Model

In section 2.3.2, we introduced two reference models that we found interesting. By comparing these two examples, we can see that there are similarities between them. For example, they both have presentation component, business component, and data component as well as some cross-cutting utility components. Since this study focuses on reference architecture for mobile applications, the model from Microsoft Patterns & Practices Team (2009b) addresses some issues that are extra interesting for this study. In order to continue the construction, we choose to build the possible generic reference architecture based on the common mobile application architecture from Microsoft Patterns & Practices Team (2009b).

4.1.3 Architectural Pattern

One of the major decisions that a software architect should make is about choosing the most appropriate architectural approach for the structure of an application. For a mobile application, this first decision is what type of application to build, for instance if it should be a client-server or a standalone application. If client-server is selected, the next decision is whether it should be a thin or a rich client. In order to choose a suitable architectural approach for a mobile application that fulfills all the expected requirements and functionalities on your target platforms, we suggest to explore the mobile platform accessibility and extension mechanisms as described in section 2.6. As the iPhone platform is more restricted than the Android platform, the differences between the openness of architecture of these platforms should be kept in mind while the software architect chooses an architectural approach for a mobile application. Since the reference architecture is intended to be generic, it is possible in some rare cases that a specific target platform can not use the generic solution as a result of insufficient architectural support for openness in that platform.

As we learned while producing the knowledge base of this study, the utter suitable architectural pattern can address constraints and challenges of mobile application, and enhance achieving the required quality attributes. The fact that each mobile application based on its nature introduces some of these challenges, e.g., demands on network availability, makes us incapable of choosing specific architectural pattern(s) that can be suitable for all kind of applications. For example, while one application that relies on data from a server may have requirements to be completely usable even while being disconnected from the network, another application may require a constant network connection in order to function. These two application examples will need to be approached with different architectural patterns. We aimed to construct a reference architecture that can be applicable and practical for designing any sort of mobile application. When we started this construction, we made an assumption that this is a linear process which is about choosing reference model and architectural pattern(s) by considering the challenges and constraints of mobile applications that affect on these decisions. We also assumed this reference architecture would suit entire needs for all sorts of mobile applications (Figure 11). As it turns out, it seems not possible to find one generic reference architecture that can be applied for all types of mobile applications. We realize that we need one reference architecture per type of mobile application, so the linear construction process we started with must be divided into different branches depending on the type of mobile application. Then similar types of mobile applications should follow the relevant branch (Figure 13). In order to proceed this construction process, we exemplify four of these branches by defining four mobile application scenarios, and illustrate how to reason when constructing a reference architecture for each of them.

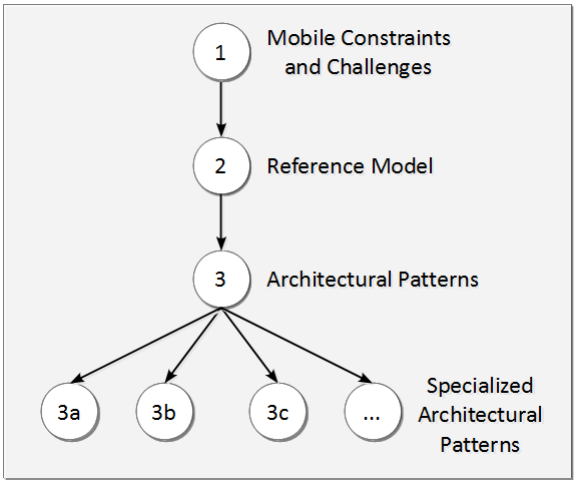

Figure 13. Construction Process

4.2 Exemplified Scenarios

In this part, we introduce some common mobile application scenarios that are defined based on the knowledge base of this study; they exemplify some possible mobile applications by considering mobile constraints and challenges (Figure 12), required quality attributes, and suitable architectural patterns. It is worth to note that this is just a few of the many possible scenarios and should only be treated as examples. Each scenario is interpreted as a branch in the construction process as depicted in Figure 13. The construction process is continued and leads to a specific reference architecture per branch. The scenarios address different quality attributes, however some quality attributes are common

for all scenarios. For example maintainability and reusability are common quality attributes among all scenarios, and can be partly addressed by using layered architectural pattern.

4.2.1 Scenario 1

Requirement: A mobile application that does not have a reliable network connection but still needs to interact with an application server. The requirements state that this application should be functional regardless of network connectivity state, i.e., even when being disconnected from the network.

Suggested architectural patterns: Since it is given in the requirements of this scenario that the mobile application needs to communicate with a server, we suggest to apply the client-server architectural pattern to describe the relationship between the mobile application and the server. We suggest using caching and synchronization, either reactive or proactive, which can be accomplished by having a local cache on the mobile device that is synchronized if needed, as soon as the network is available. In order to use caching, the application needs to have local storage which can only be accomplished by a rich client mobile application. Besides, we suggest using the message bus architectural pattern to describe the asynchronous connection to the application server.

Solution: To realize the client-server architectural pattern, we need to have at least 2 tiers, one client and one server. The server part may have several tiers, such as a database tier on a separate physical server. This application should be functional even when being disconnected from the network, hence we need to have a local storage mechanism.

Figure 14 illustrates a reference architecture for this scenario. As described in the scenario description, we use the layered architectural pattern. Since this is a rich client application the presentation layer is located on the mobile device, it is responsible for accepting user input and rendering the user interface. The client will also implement a business layer in order to duplicate some of the business rules from the application server for better performance and to support being disconnected from the network. The third layer is the data access layer, that provides access to data located either on the mobile device or on the application server. The local cache is accessed through the data access layer and is synchronized with the application server, in some way, when the network is available.

Conclusion: The reasoning behind this reference architecture provides the base for a group of applications that share similar characteristics such as usage of storage on the mobile device, demands on high uptime and connection to an application server. If an application only shares some of the same characteristics, the reference architecture may need to be adjusted. For instance, if another application does not have any demand for a local cache that part of the reference architecture could be removed, but that may also lead to other architectural decisions as will be shown in Scenario 4.2.2.

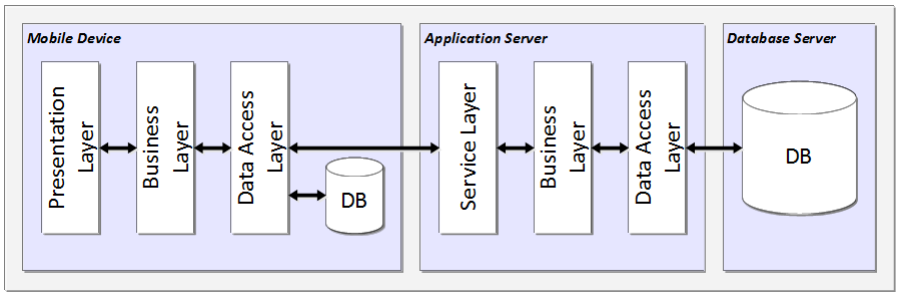

Figure 14. Reference Architecture for Scenario 1

4.2.2 Scenario 2

Requirement: A mobile application that has a reliable network connection and retrieves information located on an application server.

Suggested architectural patterns: The requirements imply that information is retrieved from an application server, thus we again suggest using the client-server architectural pattern. However, in this scenario there are no requirements that make it necessary to build a rich client since for example no local services are needed and no data needs to be stored on the client. The option that we suggest is to build a thin client in the form of a web based application. All processing is performed on the application server, and the result is presented in the web browser. By using the web browser to run the application, it can be run on most mobile platforms as long as they have a web browser installed.

Solution: Similar to scenario 1, this scenario uses the layered architectural pattern on the server side, but in this case with a presentation layer producing the output that is dis-

played in the web browser on the client. Figure 15 shows a proposed reference architecture for this scenario.

Conclusion: This scenario highlights a group of applications that are highly dependent on a network connection. If the network is not available, the application will not work. Compared to the reference architecture defined in Scenario 4.2.1, the proposed reference architecture for this scenario is quite different even though the major change between the applications is only the demand of local storage. There may be other demands on the client that makes an adjusted version of the reference architecture in Scenario 4.2.1 without local storage a better option, e.g., if there are requirements that the application should use the GPS or accelerometer built into the mobile device a web based client is probably not applicable.

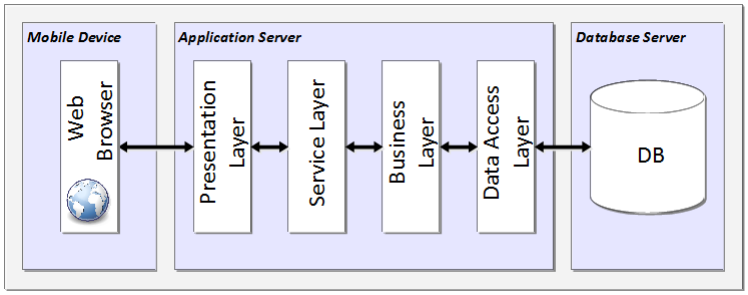

Figure 15. Reference Architecture for Scenario 2

4.2.3 Scenario 3

Requirement: A mobile application that does not require any network connection.

Suggested architectural patterns: All services related to the application is available on the mobile devices, thus a stand-alone application should be designed. Apart from the layered architectural pattern, we do not introduce any other architectural patterns.

Solution: Since the application does not utilize the network in any way; all data needed during the lifetime of the application resides on the mobile device. Depending on requirements of the application, the different layers and database may or may not be needed in order to fulfill the specified requirements. The reference architecture for this scenario (Figure 16) uses local storage and thus contains all mobile device layers and components as is described in Scenario 4.2.1.

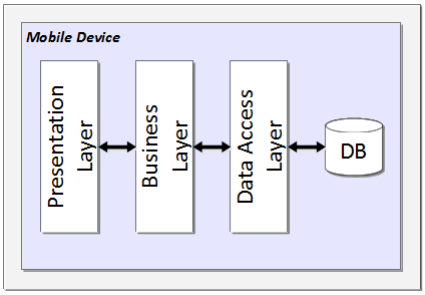

Figure 16. Reference Architecture for Scenario 3

Conclusion: This group of applications is completely self-contained and even though they may not require access to the network, there may be a need to connect to local services on the mobile device. This group of applications is very similar to the one discussed in Scenario 4.2.1 and uses the same architectural patterns to describe the reference architecture on the client side, excluding the network connectivity.

4.2.4 Scenario 4

Requirement: A mobile application that has a reliable network connection and requests information from several network resources as well as accessing local services like the phone book on the mobile device.

Suggested architectural patterns: Since the requirements include access to local services, the application must be a rich client. The difference compared to Scenario 1 is that the network connection is reliable, thus no data has to be stored locally on the mobile client. In order to retrieve information from different sources, both network and local, we suggest using the SOA architectural pattern.

Solution: Figure 17 depicts a reference architecture that fulfills this scenario. Just like the reference architecture seen in scenario 1, the proposed reference architecture for this scenario consists of a layered architecture on the mobile device using three layers. In scenario 1 the application had to store data on the mobile device, which is not a requirement for this scenario thus the local database is removed from the architecture. The access to the different service providers is done through the data access layer using predefined interfaces.

Conclusion: A family of applications that needs data from one or more servers that is not dedicated to the mobile application and thus not under the software architects control. This type of application can be combined with for instance, the application family described in Scenario 4.2.1 but the server may then be considered as orchestrator of calling the different services required in order to reduce the complexity on the mobile client.

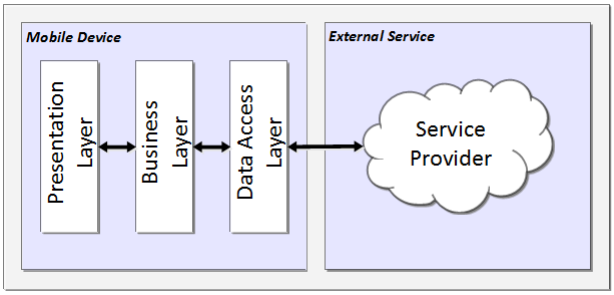

Figure 17. Reference Architecture for Scenario 4

5 Generic Reference Architecture for Mobile Applications

This study set out to explore the feasibility of a generic reference architecture for mobile applications which saves time, solves similar problems, and increases consistency among different applications. While this study has been progressing, it has become clear that there can not be one single reference architecture, which can satisfy all requirements for all types of mobile applications. As each application is unique and have different requirements, the reference architecture must address these differences even though some parts of the architecture may be similar for a family of applications that share the same characteristics. In Section 4.2, we constructed the reference architecture for four example scenarios. As it can be observed, there are similar components among them, and we also concluded that the reference architecture for each scenario can be adjusted in order to suit different needs for slightly different application scenarios. By changing the requirements of the exemplified application scenarios even just a little bit, we argued how to reason when choosing between the scenarios and how to adapt the reference architectures.

In Section 5.1, we want to use the components defined by the application scenarios and generalize them in order to build a reference architecture template that can be used for designing any type of mobile application. Subsequently, we discuss the similarities between our findings and Volvo Group Target Architecture (VGTA) as well as Volvo Group Architectural Principles (VGAP) in Section 5.2.

5.1 Reference Architecture Template

As it is discussed in Section 4.2, the similarities between the example scenarios are apparent and the same functions are reused in different tiers of the application depending on each scenario. By presenting these four scenarios, we conveyed the rationale behind mapping the reference architecture according to the set of architectural decisions for each of these scenarios. The reasoning behind the architectural solution for each of these scenarios can be applied for constructing a reference architecture for any type of mobile application. We suggest a template that presents a generic scheme for its solutions and describes one or several solutions for each particular recurring design challenge in mobile application design context. We believe that this template can help a software architect to commence the design of an appropriate reference architecture. As it is depicted in Figure 18, this template consists of several building blocks where each of them represents a group of functionalities.

For any new family of mobile applications, the whole process of constructing a reference architecture which is illustrated in Figure 13 should be followed, and the most suitable architectural pattern(s) should be chosen at the third level. The reference architecture template should be adjusted based on the selected architectural patterns. After the template has been adjusted, the next step can be to specialize a reference architecture per target platform and mobile solution as described in Figure 11. This can be accomplished by further enhancing the reference architecture through involving mobile technologies and solutions, and introducing design patterns in for example the form of mobility patterns to this reference architecture.

We mentioned that the goal of this work was to find out if it is possible to have a generic reference architecture for all families of mobile applications. The outcome of this exploration showed that it is not possible to have a generic reference architecture and instead we propose using a reference architecture template. It might be controversial that the proposed reference architecture template have been constructed based on a limited collection of architectural choices. We are well aware of the influence of restricting the architectural decision regarding reference model into just two instances. In addition, we accept the fact that not all of the existing architectural patterns have been included in this construction process. We think that the outcome of this work may differ if another reference model, more architectural patterns and mobile constraints and challenges were considered in this construction. However, it is worth to mention that we chose the most common and appropriate architectural decisions, as well as the most common mobile constraints and challenges considering the time limitation of this study. We believe that this thesis work should be

utilized as a starting point for constructing a comprehensive reference architecture template, which addresses all possible mobile constraints and challenges by including design patterns.

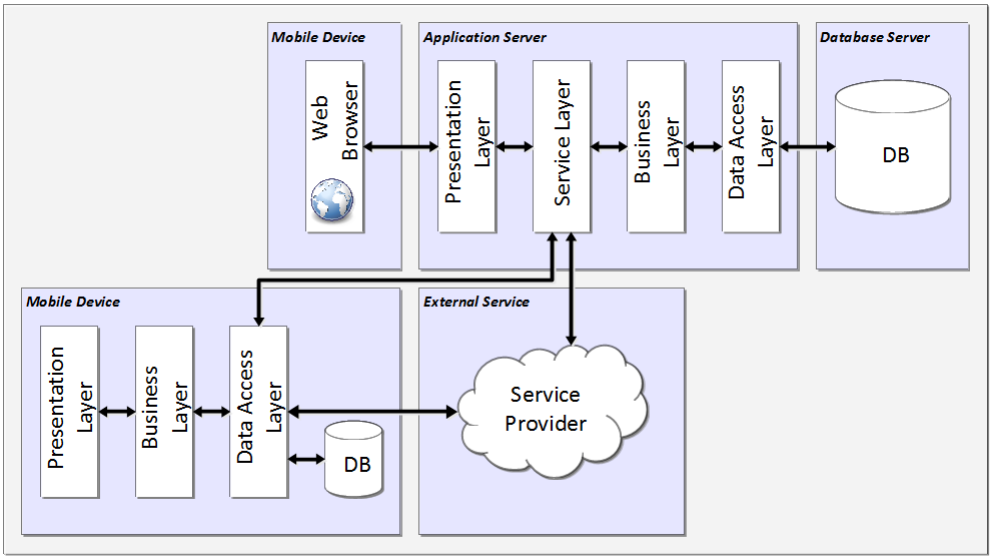

Figure 18. Generic Reference Architecture Template

5.2 VGTA and VGAP

In this part of the study, we describe briefly how similar the reference model used as input for constructing the reference architecture for mobile applications is with the existing reference model, VGTA, used at Volvo IT. As it is mentioned in Section 3.1, the author has signed a non-disclosure agreement and therefore can not share the detailed information about the structure of the Volvo IT reference model and related documents. However, the author can share the result of this comparison without going into details.

We compared the building blocks of VGTA, with the reference model described in Figure 2. The reference models are similar as each of them consists of several components and relations between them; it is also worth to note that they have several of these components and relations in common. Due to these similarities we argue that the VGTA can be used in the second step of the construction process (Figure 13) in order to build a generic reference architecture as part of the mobile solution for Volvo IT.

As we can not go into details about VGAP in this report, we can only mention that these architectural principles are valuable input while enriching the findings of this study. We believe that these architectural principles can be roughly interpreted as quality attributes, and all of them can be considered for mobile application development as well. However, since VGAP had been defined before Volvo IT started mobile application development, some of these architectural principles can not be fully applied. Mobile development track needs to grow, gain experience, and possibly introduce new services and guidelines in order to fulfill all of these principles.

6 Conclusion

The purpose of this thesis work was to explore the feasibility of a reference architecture to be utilized for designing all types of mobile applications, in order to help a software architect with the work of designing a software architecture for one or several mobile applications. In addition, we investigate the possibility of using the findings of this thesis work in Volvo IT considering their existing architectural artifacts.

During the construction process, we came to the conclusion that there can not be one reference architecture which fulfills the needs of all types of mobile applications, due to the challenges and the very nature of different mobile application devices. For example, if a mobile application requires data from a server to function but may not always have a network connection; local data storage may be needed which in turn puts certain requirements on the architecture of the application.

In order to address the problem that there can not be one single reference architecture to cover all possible mobile applications, we presented a reference architecture template in Section 5.1 as a mean to create a reference architecture for each individual mobile application. The template shows

a layered structure and the relationship between different types of clients, for example rich or thin, and the relationship with a dedicated application server or other online services. We used four different mobile application scenarios in Section 4.2 in order to reason and explain how to create a reference architecture for a family of mobile applications. We reasoned how a minor change to the requirements would impact the proposed reference architecture, and also explained how close to each other the different scenarios are from a requirements point of view even though the reference architecture may differ quite a lot.

In conclusion, the deliverable of this thesis is both the reference architecture template, and a way of thinking and reasoning behind each architectural decision that should be made at different stages of software architecture design. We believe that the outcome of this study can, to some extent, ease the process of creating a software architecture for a mobile application. We also believe that the template can be further enriched by adding design patterns and considering other architectural artifacts as input to the described process of creating a reference architecture for an application.

The study investigated architectural artifacts used at Volvo IT for developing software applications in different development tracks, in order to find out if the generic reference architecture template for mobile applications can be utilized for the mobile development track. The result of the study proved the potential of using the reference architecture template as well as the existing reference model and architectural artifacts at Volvo IT as inputs for this template. Since Volvo IT has started to incorporate mobile application development as a new development track into the company, we suggest to continue this study and enrich the proposed reference architecture template by involving mobility patterns, and considering other architectural choices compared to the ones made in this study. Furthermore, as the main focus of Volvo IT is on Android and iPhone platforms, both the mobility patterns and the proposed reference architecture template should be specialized for each of these platforms.

Other companies that have similar architectural artifacts as has been used as input for the work described in this report, can compare for example their existing reference model with the one used in this thesis work to see if there are enough similarities for it to be considered as it is, or adapt either their existing reference model or the template provided in this report accordingly.

Acknowledgment The author would especially like to thank Micael Andersson and Peter Nordlander at Volvo IT for their valuable contribution to this research, Gerardo Schneider, Helena Holmström Olsson, Carl Magnus Olsson, and Agneta Nilsson at IT University of Gothenburg for their thoughtful guiding of this thesis work, and Henrik Utter for his technical insights and continuous constructive discussions during this study.

References

[Ahlgren & Markkula, 2005] Ahlgren, R., & Markkula, J. (2005). Design Patterns and Organisational Memory in Mobile Application Development. In Bomarius, F., & Komi-Sirvi, S. (Eds.), *Proceedings of the 6th International Conference on Product Focused Software Process Improvement (Profes2005), LNCS 3547*, (pp. 143-156). Berlin, Germany: Springer-Verlag.

[Albin, 2003] Albin, S. T. (2003). *The Art of Software Architecture: Design Methods and Techniques*. Indianapolis, IN, USA: Wiley Publishing, Inc.

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

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

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

[Buschmann, Henney & Schmidt, 2007a] Buschmann, F., Henney, K. & Schmidt, D. C. (2007a). *Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, Volume 4*. Chichester, UK: Wiley.

[Buschmann, Henney & Schmidt, 2007b] Buschmann, F., Henney, K. & Schmidt, D. C. (2007b). Past, Present and Future Trends in Software Patterns. *Software, IEEE, 24*(4), 31-37.

[Clements et al., 2002] 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.

[Creswell, 2009] Creswell, J. W. (2009). *Research Design: Qualitative, Quantitative and Mixed Methods Approaches* (3rd ed.). Thousand Oaks, CA, USA: Sage.

[Fowler et al., 2002] Fowler, M., Rice, D., Foemmel, M., Hieatt, E., Mee, R., & Stafford, R. (2002). *Patterns of Enterprise Application Architecture*. Boston, MA, USA: Addison-Wesley Professional.

[Gamma et al., 1995] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston, MA, USA: Addison-Wesley.

[Gasimov et al., 2010] 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.

[Hevner et al., 2004] Hevner, A., March, S., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly, 28*(1), 75-105.

[Kasanen, Lukka & Siitonen, 1993] Kasanen, E., Lukka, K., & Siitonen, A. (1993). The constructive approach in management accounting research. *Journal of Management Accounting Research, 5*(1), 243-264.

[Malek et al., 2009] Malek, S., Edwards, G., Brun, Y., Tajalli, 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.

[Mazhelis, Markkula & Jakobsson, 2005] 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.

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

[Microsoft Patterns & Practices Team, 2009b] Microsoft Patterns & Practices Team (2009b). *Mobile Application Architecture Guide*. Retrieved April 04, 2011 from http://apparch.codeplex.com/releases/view/19798

[Microsoft Patterns & Practices Team, 2009c] Microsoft Patterns & Practices Team (2009c). *App Pattern: Three-Tier Mobile Application Scenario*. Retrieved April 04, 2011 from http://apparch.codeplex.com/wikipage?title=App%20Pattern%20-%20Three-Tier%20Mobile%20Application%20Scenario&referringTitle=Application%20Patterns

[Natchetoi, Kaufman & Shapiro, 2008] 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.

[Roth, 2002] Roth, J. (2002). Patterns of Mobile Interaction. *Personal and Ubiquitous Computing, 6*(4), 282-289.

[Seaman, 1999] Seaman, C. B. (1999). Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering, 25*(4), 557-572.

[Teng & Helps, 2010] 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 & Murugesan, 2010] Unhelkar, B., & Murugesan, S. (2010). The Enterprise Mobile Applications Development Framework. *IT Professional, IEEE Computer Society, 12*(3), 33-39.

[van Gurp, Karhinen & Bosch, 2005] van Gurp, J., Karhinen, A., & Bosch, J. (2005). Mobile Service Oriented Architectures. *Proceedings of the 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems (DAIS 2006), LNCS 4025*, (pp. 1-15). Bologna, Italy: Springer.

[Volvo IT, 2011] Volvo IT (2011). *Volvo IT company presentation 2011* [PDF]. Retrieved April 04, 2011 from http://www.volvoit.com/SiteCollectionDocuments/Volvo%20IT/documents/other/Volvo%20IT %20company%20presentation.pdf