International Journal of Bio-Science and Bio-Technology
Vol.5, No.5 (2013), pp. 197-210
http://dx.doi.org/10.14257/ijbsbt.2013.5.5.21

Architecture for Adaptive Mobile Applications

Haeng-Kon Kim

School of IT Engineering, Catholic University of Daegu,
Kyungsan-si, Kyungbuk, 712-702, Korea
hangkon@cu.ac.kr

Abstract

Mobile computing is a relatively new field. While the challenges arising from mobility and the limitations of the portable devices are relatively well understood, there is no consensus yet as to what should be done to address these challenges. A comprehensive solution has to address many different aspects, such as the issue of dynamically changing bandwidth, the power, computational, and other limitations of the portable devices, or the varying availability of services in different environments.

In this paper, we present our architecture for such adaptive mobile applications. We motivated the architecture by classifying likely mobile applications and identified common properties. The architecture intends to be more general than previous work with respect to adaptability, flexibility, and user mobility. We developed various pieces of the overall architecture and collected some preliminary experience with adaptive mobile applications. We give an overview of the intersection of the areas of software architecture and mobile applications. We consider the mobile applications, which represent the computing functionality designed to migrate across hardware devices at runtime and execute on mobile hardware platforms, and the mobile systems, which are computing applications that include mobile software and hardware elements.

We are developing the components of a flexible and general-purpose runtime infrastructure to facilitate the rapid development and deployment of such adaptive mobile applications. We will evaluate our infrastructure by implementing a number of wireless applications and by building simulation tools to validate the scalability of our architecture when considering metropolitan and provincial cellular systems.

1. Introduction

The mobile applications support a much wider range of activities than desktop applications and leverage information about the user's environment to provide novel capabilities. From a technology perspective, mobility shifts the global computing infrastructure from static, homogenous, powerful desktop computers to highly dynamic, heterogeneous, resource-constrained handheld and wearable computers. This new computing context demands entirely new software architectural paradigms that address the challenges of mobile software development, are specialized for the nature of mobile devices and wireless networks, and take advantage of the opportunities afforded by mobile systems. This new computing context demands entirely new software architectural paradigms that address the challenges of mobile software development, are specialized for the nature of mobile devices and wireless networks, and take advantage of the opportunities afforded by mobile systems. Recent research has rapidly advanced the state-of-the-art in architectures for mobile software and systems. Researchers have established the technical basis for mobility by creating formal models of and engineering processes for mobile software. Mobile software is computing functionality designed to migrate across hardware devices at runtime and execute on mobile hardware platforms. The principles of software architecture are intrinsic to the development environments and run-time platforms that support mobile software models and processes. We motivated the

ISSN: 2233-7849 IJBSBT

architecture by classifying likely mobile applications and identified common properties. The architecture intends to be more general than previous work with respect to adaptability, flexibility, and user mobility.

In this paper, we developed various pieces of the overall architecture and collected some preliminary experience with adaptive mobile applications we explain the core concepts and open research problems at the cross-section of the fields of software architecture and mobility.

In parallel with (and supported by) these achievements, research activity that leverages this work to build mobile systems has expanded. Mobile systems are computing applications that include mobile software and hardware elements. These applications are characterized by customized software architectures that are designed for and intrinsically facilitate mobility. We present our architecture for such adaptive mobile applications. We motivated the architecture by classifying likely mobile applications and identified common properties. The architecture intends to be more general than previous work with respect to adaptability, flexibility, and user mobility. We developed various pieces of the overall architecture and collected some preliminary experience with adaptive mobile applications. We give an overview of the intersection of the areas of software architecture and mobile applications. We consider the mobile applications, which represent the computing functionality designed to migrate across hardware devices at runtime and execute on mobile hardware platforms, and the mobile systems, which are computing applications that include mobile software and hardware elements.

2. Related Works

2.1. Mobile Applications Design

The key software architectural abstractions are components, connectors, their interfaces, configurations, and constraints on system structure, behavior, composition, and interaction. Architectural styles are essentially named sets of such constraints: client–server, peer to-peer (p2p), publish-subscribe (pub-sub), event notification, and so on. It is not readily obvious what level of support for mobility is yielded by, say, a given architectural style such as p2p or pub-sub. These traditional styles do provide many design guidelines that can prove to be useful in mobile applications. The guidelines include:

- Component decoupling, whereby the practical units of mobility are delimited – most architectural styles adhere to this guideline;
- Avoiding shared memory, whereby potential side effects of computations are limited and interaction ‘‘back doors” eliminated
- Styles such as pub-sub and event notification adhere to this guideline;
- Insulating components from execution context, whereby a single component may be effectively redeployed onto a number of sites
- Implicit invocation, whereby no component relies directly on the geographical presence or location of another
- Asynchronous interaction, whereby no component relies on the temporal presence of another
- Sateless components, whereby the migration process and system consistency assurance are greatly simplified – clients in some client–server applications are examples of stateless components
- Stateless interactions, whereby each interaction is self-contained and handled independently of the service requester’s or provider’s location

2.2. Architecture Definition

As stated earlier, the architecture is a new programming paradigm that can ensure software reuse. Architecture is a template for working programs since it is a set of classes and interactions among the classes that embodies an abstract design for solutions to a family of related problems in one domain [9, 12]. Architectures are not simply collections of classes but also wired-in interactions between the classes that provide an infrastructure and architecture for the developers. To provide an infrastructure, we define the template classes that are the common and general classes to a group of related applications, and define the hot spots to a specific application. These hot spots are redefined or extended by the specific application developer later. It can be said that applications are successful when they satisfy the requirements of the customer. However, framework can be said to be successful when it can be easily customized for several different requirements. Therefore, designing a architecture is a little different from designing an application. We need more general and stronger abstract design concepts for designing architecture.

2.2. Architecture Description

Figure 1 illustrates an overview of architecture for Mobile applications development that involves two main participants: the mobile client environment and the backend server/mobile Web service. This architecture represents a logical structure and does not necessarily reflect a physical architecture. For example, the backend server may actually be load-balanced across several machines. In a large implementation, separate machines are used to run specific components of the architecture. A description of each module of our architecture presented in Figure 1 as overview of architecture for mobile applications is provided hereafter.

A. Backend Server

The backend server consists of eight components/modules that are: the enterprise web applications, business logic, content management system, workflow and profile management, user management, document repository, Web services, and database.

•**Enterprise Web Applications:** integrate functionality from remote Servers that might host Web services, portlets, *etc*. The backend server might support the J2EE standard Management System.

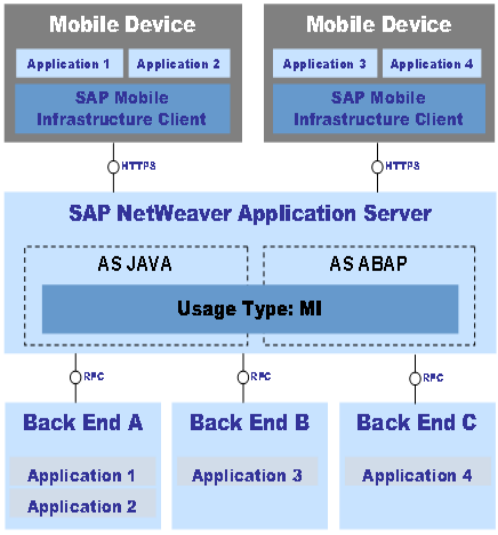

Figure 1. Overview of Architecture for Mobile Applications

•**Business Logic:** represents the core of the mobile application that exposes functionalities. The business logic can be deployed on the backend server and used remotely by the mobile application to reduce the load due to limited resource available on the mobile devices. In other situation, a subset or the whole of the business logic can also be deployed on the mobile device.
•**Content Management System:** is a system (set of tools) used to manage different contents (data, multimedia information, and forums. These include for example, importing, creating, updating, and removing content to/from the mobile application.

•**Workflow and Profile Management**: manages the flow of data between the mobile application and the backend server and adapts the mobile application features and interface to different users' profiles.

•**User Management:** allows management of different types/classes of users with different levels of services and/or permission.

•**Database:** is used to store metadata about objects and content.

•**Document Repository:** is used to store different types of documents… The repository keeps track of all the updates performed by different users and provides version control capabilities.

•**Web Services:** are web components that can be deployed on backend server and can be invoked/consumed by the mobile application.

B. Mobile Device Client Environment:

The client environment consists of six components/modules that support the mobile client environment; these are mainly the business logic, secure layer interface, mobile application API, user interface, data storage, and the browser.

•**Business Logic:** as stated above, there are situations where it is suitable to deploy some of the business logic on the mobile device itself. For example, business logic required for offline transactions.

•**Secure Layer Interfaces:** handles secure user authentication for the Mobile environment using a protocol such as SSL, and provides different interfaces for different user's profile (*e.g.* admin, normal user, *etc.*).

•**Mobile Application APIs:** is a set of libraries and application programming interfaces (API) that implement a set of utilities that might be required by the mobile application (*e.g.*).

•**User Interface:** this is the graphical user interface components that mobile clients use to access different services. User interfaces includes mainly widgets, navigation facilities, and search tools.

•**Data Storage:** this is the place where persistent data is stored. This might include transactions data, users' personal information, profiles, and preferences.

•**Browser:** these consist of *minibrowsers* that are designed for use on mobile devices and optimized to display web content most effectively on small screens. The architecture components' provide a rich, drag-and-drop development environment for development of mobile applications. This environment supports developers in providing efficient and secure mobile applications. Efficiency, in terms of used resources, is highly required for such applications due to limited available resources in mobile devices and the drawbacks of wireless networks in terms of low bandwidth and high latency. Security is of prime importance whenever wireless and open networks are being used as a medium for data exchange. Figure 2 shows the overview of mobile device client applications.

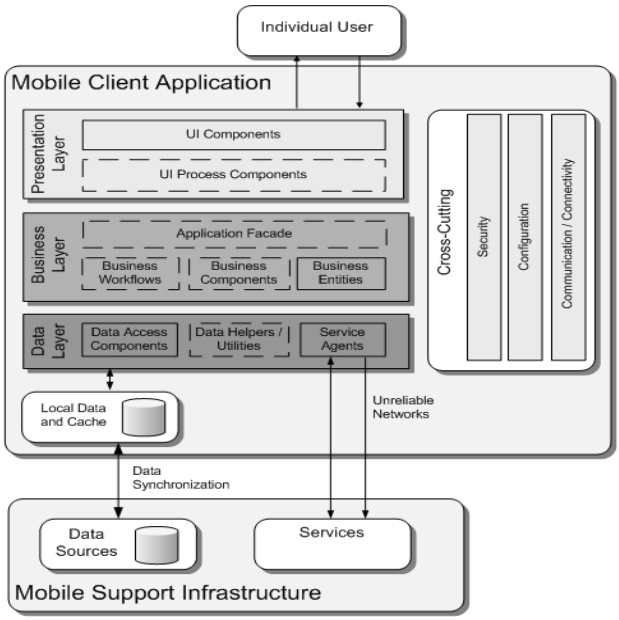

Figure 2. Overview of Mobile Device Client Applications

3. Architecture for Adaptive Mobile Applications

Current adaptive mobile applications are built using one of two approaches: either the adaptation is performed by the system which underpins the application (in an attempt to make transparent the effects of mobility) or, the application itself monitors and adapts to change. In some cases, these approaches are combined as where the middleware platform adapts the operation of the network protocol in the face of changes in QoS and, additionally, report these changes to the application to enable application level adaptation. In the general case, it has been demonstrated that maintaining transparency in the face of mobility is not practical and that it is difficult for a system to adapt without support from the application.

3.1. Components for Mobile Architecture

Most mobile systems extend an existing business system or interface with an existing system. There are typically three major components to a mobile architecture- An existing system, a middleware application, and a handheld application as in Figure 3.

Figure 3. Basic Mobile Architecture

The reason a middleware application is usually needed is to provide data transformation, apply business logic, and be a central point of communication for the devices. If a new business system is being developed or rewritten then no middleware may be necessary; the appropriate logic can be built into the system to communicate with the devices from the start. However most business systems are not rewritten very often and it is economically unfeasible to rewrite them just to 'mobilize' them. Furthermore a middleware server may also serve a configuration management server.

The architectures shown here are real-world architectures from actual projects. These mobile systems are in production in numerous locations.

Interfacing Disparate Technologies

Mobile system development often involves using different technologies due to platform restrictions. In the three architectures below, two use both Microsoft and Java technologies. In both cases applications developed in these disparate technologies must communicate seamlessly with each other. Web services, HTTP, and TCP sockets were used to bridge these gaps. A mobile system development team must have the skill and experience to determine the best data transfer design.

Device Configuration

If the application is being deployed on new handheld devices, there is a good chance that some configuration will be required. After a device is cold booted, the deployed application must be reloaded and the 802.11 wireless configurations must be restored. Different manufacturers use proprietary methods for loading applications and configuration settings. If the users should not have access to the OS (*e.g.* to play solitaire) then a top-level menu application may be needed to run at system startup. Device scanners must be configured with the correct barcode symbolizes and symbology options. Configuration options may need to be remotely managed as well.

Software Deployment and Upgrades

Beyond device configuration is software deployment. The application or suite of applications must initially be loaded or provisioned on the device. If there are many devices, this may be a formidable challenge. There are software packages that manage device software and configurations. These packages rely on a software client on the device. Proprietary packages must typically be written for the management applications that specify the software and configuration files to load. If no management package is used, the application should be self-updating. Having the users send in their devices to have software reloaded is usually unrealistic.

User Interface Design

Designing the graphical user interface (GUI) on a mobile device can be challenging because of the small screen and difficult data entry. If the application or data is complex, the user will need to interact with many screen objects such as entry fields, lists, and radio buttons. Complex screens will need to be divided into separate screens or tabbed interfaces. A wizard-like interface may be appropriate for some applications. Some applications on pen-based devices may require that a stylus is not required and the device's physical keys must be used instead. If a lot of free-form data entry is required then a tablet or notebook PC should be considered.

Performance

Servers and desktop computers have progressed significantly and performance is typically not an issue anymore. Handheld computing devices are another story however. Many are very slow by comparison. Complex user interfaces, CPU intensive algorithms, and data processing can easily make an application user-hostile. Care must be taken during design to avoid performance pitfalls. One pitfall in Compact Framework development is using ADO.Net Datasets. They are very slow on most handheld devices.

Memory Management

Although memory is cheaper than ever, most mobile devices come with a set amount of memory and cannot be upgraded. If systems analysis shows that data requirements include having large amounts of data on the handheld, this may limit your hardware choices. Efficient data storage is necessary, and low-level interfaces may be required to make the most of the memory available. Because cold boots typically erase all non-volatile memory in the device, design must ensure that critical data is stored in non-volatile memory.

Security

Security is a concern in many systems and mobile systems are no different. Mobile systems introduce a few new issues however. What if the mobile device is lost? A stranger cannot be allowed access to your sensitive business data. Some hardware includes thumbprint scanners to authenticate users. A user login may also be implemented so that users must present a set of credentials before application use.

Data Transfer

Data transfer is a significant part of mobile application architecture because of the number of 'hops' the data must make. The methods and protocols should be carefully considered during system design.
The typical tradeoffs are-

- Security
- Ease of Implementation
- Reliability
- Cost of Ownership

If the application is run on a secured LAN, then security considerations may be low. If cost and ease of implementation are factors, then a less reliable method may be used. If reliability must be guaranteed, then a more expensive communication mechanism should be considered.

Several choices for network communication are listed here-

Web Services

Web services are relatively easy to implement and existing web services in service oriented architecture (SOA) or an enterprise service bus (ESB) may potentially be used. Web services are technology agnostic and may be used between most platforms and technologies. They are not well suited for large, binary data files. Security is available, but may be more difficult to implement between heterogeneous technologies.

Queuing Services

Queuing services guarantee data delivery. They may also be configured for store-and-forward capability. There are few queuing packages available for mobile devices and may be cost prohibitive. Queuing technologies for servers are well established and several good choices exist.

TCP Sockets

Transferring data over a raw network socket offers the greatest speed of all methods. If security is required it will need to be implemented in code, usually with an encryption library. Implementation can be quick if only raw file transfer is required. Multithreading will be required on the server so that multiple devices can connect simultaneously. If

something more complicated than file transfer is required then another method (such as web services) should be considered so that a complex custom protocol does not have to be developed.

HTTP

HTTP can provide an easy way for an application to send messages and receive data from another application running a web application. SSL can easily be used to provide security. Implementation is not complicated.

FTP

Although usually not preferred, FTP still has a place in system integration. This may be the only way to transfer data to older business systems. If FTP is chosen as a data transfer method, consideration must be given to an appropriate threading architecture with retry capability. Most FTP servers are not configured with encryption, but may be set up to provide it.

Disconnected Architecture

This system is deployed in more than twelve manufacturing plants and eliminates clipboards and manual data entry. The biggest installation has over sixty users. At the time of this writing, this system has been in production for more than four years. It will continue to be improved and modified over the coming years.
This architecture addresses these high-level requirements-

- Business data (*e.g.* customers, materials, etc.) is available on the device.
- Business data may be large.
- The device may be used in a disconnected mode, e.g. out of the office.
- Users enter data, return to the office, and send the data to the business system.
- The only interface available to the business system is FTP.

Business System

The business system is a SCO Unix platform running a manufacturing shop floor management system. The only interface is via FTP and the import/export interface already exists on the SCO server. Data from the handheld devices must be manually imported in the business system by a menu choice, but this was later eliminated.

Handheld Application

The handheld application is comprised of more than fifteen screens. Users must log into the application with a username and password. The data validation logic is part of the business data from the SCO system. The hardware used to run the application are ruggedized Pocket PCs that contain non-volatile memory and optional Bluetooth and wireless networking. If Bluetooth is available, the application can print reports to a mobile Bluetooth printer. The application is self-updating; it downloads new software via the socket server from the middleware application.

Cellular Phone Architecture

This system was developed for a large logistics company. Truck drivers carry cellular phones running the mobile application. The application informs the drivers of new jobs, allows entry of pickups and deliveries, and general messaging.

This architecture addresses these high-level requirements-

- The system must guarantee message delivery.
- The system must store messages if a mobile device is out of the cellular data network.
- The mobile application must still allow data entry when out of network.

3.2. Mobile Applications Development using Architecture

Once a software system's architectural design is reasonably complete, it is supposed to be realized faithfully in the system's implementation. The objective of implementing architecture is simple: realize the principal design decisions and achieve the key intended non-functional system qualities as a result. However,

The system's implementation frequently departs from the intended architecture and many, if not most, existing software systems suffer from architectural drift.

Figure 4 shows the component architectures for mobile applications that consist of business system, middleware applications and handheld applications.

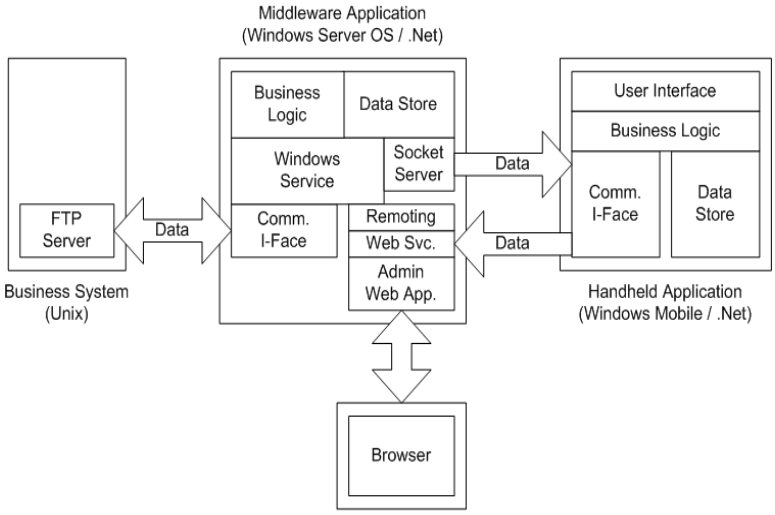

Figure 4. Components for Mobile Architecture

As with any large, distributed, long-lived systems, mobile systems are most effectively developed with the help of middleware facilities. These facilities rest on top of operating systems and programming language runtimes, and provide the needed implementation-level abstractions and execution support for the desired services: distribution, resource management, service discovery, security, run-time adaptation, and so on. Many different middleware solutions have emerged over the past couple of decades with the proliferation of distributed computing. An increasing number of middleware solutions have been specifically geared to mobile systems.

Traditional middleware – These are typically object-oriented middleware platforms adapted to mobile settings, in order to render mobile devices interoperable with fixed networks.

Context-aware middleware – These middleware platforms specifically target the heterogeneity of hosts and networks, limited resources, and variations in the user environments in order to improve an application's behavior.

Data sharing middleware – These middleware platforms in particular target disconnected operation, where the availability of data is ensured regardless of the network's performance and reliability.

Tuple spaces middleware – Certain middleware platforms allow system components to

communicate in a manner that is decoupled in both time and space. This is achieved via tuple spaces, which are globally shared memory spaces in which data structures are vectors of typed values.

While these platforms support the construction of different classes of and provide support for different aspects of mobile systems, the question of the extent to which any of them supports the architectural integrity of those systems remains. In fact, it can be argued that they do not help to capture architectural notions or enforce architectural design decisions pertaining to system computation, data, interaction, and styles. Thus, for example, - Components can be realized in a number of different ways, including object classes, packages, modules;

- Connectors may not exist explicitly in system implementations (other than as procedure calls), or they may be embodied in facilities such as software buses and messaging services;

- Interfaces are typically captured as the system modules' API signatures; however, other aspects of interfaces, such as constraints on their use, are not readily transferred to the implementation;

- Architectural configurations are typically hidden in system implementations due to the typical realization of interfaces (in particular, the required interfaces, which are distributed throughout the code in the form of method calls), use of function pointers, dynamic class loading and binding, and reflection;

- Design rationale is not captured in the system explicitly, but rather is implicit in the comments embedded in the code as well as system documentation;

- Behavior is embodied throughout a system in the form of algorithms, but a critical question is how an engineer is to translate formal notations used at the level of architecture.

- Non-functional properties are a direct function of the implementation, but are not in any way visible in the implementation; rather, they are assessed indirectly and often partially via design rationale, code inspections, testing, user studies, and so on.

In order to address this discrepancy between the architectural intent and design on the one hand and system implementation on the other hand, another category of middleware platform has been recently proposed. This category has grown out of early work on architectural implementation architectures

Architectural middleware – A class of mobile middleware platforms focuses primarily on providing implementation-level constructs for the realization of architectural concepts and abstractions. In these platforms, developers explicitly declare software components, connectors, communication ports and events, and may even be able to specify named sets of implementation constraints that correspond to specific architectural styles.

All aspects of security, including user authentication and encryption, are supported through a well-defined and efficient set of as software systems have become increasingly mobile, different solutions for supporting that mobility have emerged before. They divide characteristics of mobile systems along three dimensions: device network connection, and execution context. An execution device can be fixed or mobile. A network connection can be permanent or intermittent. The execution context can be static or dynamic. A mobile system can have different combinations of these characteristics, but typically in this setting we are interested in software systems executing on mobile devices, with intermittent network connections, and with dynamic execution contexts. In other words, we are primarily focusing on a specific subset of the problem space, comprising

1. Systems that are deployed on novel computing platforms that have emerged during the past decade – smart phones, mobile robots, motes, sensors, and so on;

2. Novel usage scenarios frequently demanding computation anytime, anywhere – search-and-rescue, environment exploration, traffic management, medicine, assisted living, sensor networks, and so on, and thus

3. Amplified software engineering challenges – distribution, decentralization, heterogeneity, resource constraints, context-awareness, real-time requirements, and so on.

Software architecture is defined as the set of principal design decisions P about a software system. This means that a number of other system facets (*e.g.*, low-level implementation support). Certain design-level solutions are better suited to mobile systems than others. Such solutions are usually embodied in architectural patterns and styles. A software architectural style is a named collection of architectural design decisions that (1) are applicable in a given development context, (2) constrain architectural design decisions that are specific to a particular system within that context, and (3) elicit beneficial qualities in each resulting system.

An issue relevant in the context of architecture-based software development that is particularly amplified within mobile systems is the relationship between a system's ''as-designed" and ''as-implemented" architectures.

An as designed architecture, also referred to as the system's prescriptive architecture, is the set of architectural design decisions P made at time t that reflect the architects' intent. The system's as-implemented architecture, also referred to as its descriptive architecture, is the set of artifacts a realize the design decisions P.

The artifact set A could take the form of refinements of design decisions in a given modeling notation, models of employed architectural styles and patterns, existing off-the-shelf components, implementation frameworks and middleware, and so on.

4. Mobile Applications Design and Development

4.1. Mobile Application Development

Mobile application design and development is a tricky balancing act. High levels of application performance and usability must be achieved while working with many device-related constraints. The Figure 5 shows the mobile applications development process with our suggested architecture.

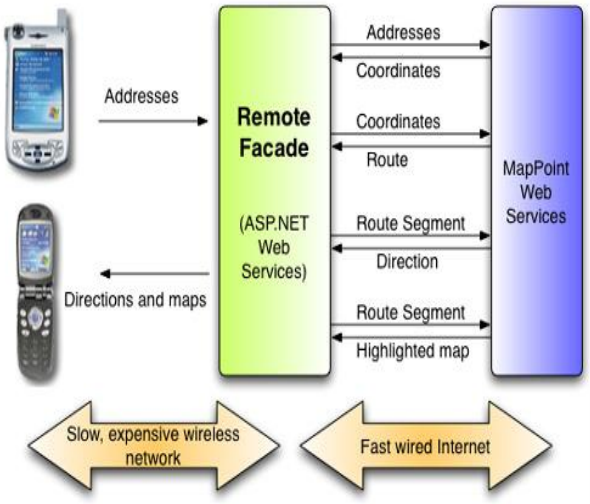

Figure 5. Mobile Applications Development using Architecture

The following are some of the important application architecture and design considerations that will be useful:

Development Methodology:

Use an iterative development methodology that includes rapid iterations of design, prototyping, development and continuous testing.

Architecture & UI Design:

Design a layered architecture appropriate for mobile devices that improves reuse and maintainability. Depending on the application type, multiple layers may be located on the device itself. Use the concept of layers to maximize separation of concerns, and to improve reuse and maintainability for the mobile application. Mobile devices require a simpler architecture, UI, and other specific design decisions in order to work within the constraints imposed by the device hardware. Keep these constraints in mind and design specifically for the device instead of trying to reuse the architecture or UI from a desktop or Web application.

Target Devices:

Consider the types of devices that will be supported and its constraints. Design decisions are highly influenced by target device's screen size, resolution, orientations, memory, CPU performance characteristics, Operating Systems capabilities & limitations etc. This is the definitive guide to building successful mobile applications. It covers every facet of development and deployment, including business issues, architectural design, integration with existing web and legacy applications, and the management of mobile application development projects. It also presents three application case studies that demonstrate best practices at work in real projects. Coverage includes:

(1)Requirements, design, development, integration, testing, release, and maintenance
(2)Mobilizing existing application architectures
(3)Building effective user interfaces for mobile applications
(4)Fat-client and thin-client scenarios
(5)Managing client-server data transfer
(6)Securing mobile applications: authentication, encryption, and data self-destruction
(7)Full Microsoft .NET code examples for cell phones, Pocket PCs, and Tablet PCs

Mobile Applications is indispensable for everyone who needs to deliver robust, high-value mobile solutions: project managers, technical leaders, architects, and experienced developers alike.

Mobile application design and development is a tricky balancing act. High levels of application performance and usability must be achieved while working with many device-related constraints. The following are some of the important mobile application architecture and design considerations:

4.2. Case Study

4.2.1. Description

To experiment with some of the features of my architectures, we design and implement a mobile application name as “Body Clinic”. Body clinic is a dietary guide application that integrates data from different users through different devices that engage them in truly measuring devices. The mobile application enables users to access the systems and use all its features. The client exchanges SOAP request/responses with the application servers.

4.2.1. Static Structure and Behavior

We have designed and implemented the applications using UML and proposed architecture that solved most of the mobile applications development challenges. Figure 6 shows the static structures and behaviors with class and sequence diagram in our architectures. Figure 7 shows the User Interface for the applications.

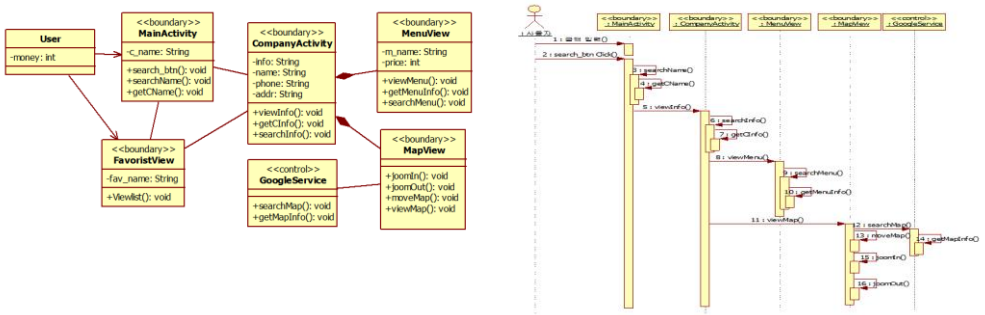

Figure 6. Body Clinic Class and Sequence diagram using Architecture

Figure 7. UI and Front Contents of Body Clinic

5. Conclusion and Future Works

The purpose of this work is to explore the feasibility of reference architecture to be utilized for designing all types of mobile applications, in order to help a software architect with the work of designing software architecture for one or several mobile applications.

During the construction process, we came to the conclusion that cannot be 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. 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 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 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 designed process of creating reference architecture for an application.

Acknowledgements

This work (Grants No. C0124408) was supported by Business for Cooperative R&D between Industry, Academy, and Research Institute funded Korea Small and Medium Business Administration in 2013.

References

[1] R. Ahlgren and J. Markkula, "Design Patterns and Organizational Memory in Mobile Application Development", Bomarius, F., & Komi-Sirvi, S. (Eds.), Proceedings of the 6th International Conference on Product Focused Software Process Improvement, Berlin, Germany: Springer-Verlag, LNCS 3547, **(2005)**, pp. 143-156.
[2] S. T. Albin, "The Art of Software Architecture: Design Methods and Techniques", Indianapolis, IN, USA: Wiley Publishing, Inc., **(2003)**.
[3] L. Bass, P. Clements and R. Kazman, "Software Architecture in Practice", (2nd ed.). Boston, MA, USA: Addison-Wesley.
[4] F. Buschmann, K. Henney and D. C. Schmidt, "Past, Present and Future Trends in Software Patterns", Software, IEEE, vol. 24, no. 4, pp. 31-37.
[5] S. Malek, G. Edwards, Y. Brun, H. Tajalli, J. Garcia, I. Krka, N. Medvidovic, M. MikicRakic and G. S. Sukhatme, "An Architecture-Driven Software Mobility Framework", Journal of Systems and Software, vol. 83, no. 6, **(2007)**, pp. 972-989.
[6] O. Mazhelis, J. Markkula and M. Jakobsson, "Specifying Patterns for Mobile Application Domain Using General Architectural Components", Bomarius, F., & Komi-Sirvi, S. (Eds), Proceedings fo the 6th International Conference on Product Focused Software Process Improvement, Berlin, Germany: Springer-Verlag, LNCS 3547, **(2005)**, pp. 157-172.
[7] B. Unhelkar and S. Murugesan, "The Enterprise Mobile Applications Development Framework", IT Professional, IEEE Computer Society, vol. 12, no. 3, **(2010)**, pp. 33-39.
[8] C. Mascolo, L. Capra, S. Zachariadis and W. Emmerich, "XMIDDLE: a data-sharing middleware for mobile computing", International Journal on Personal and Wireless Communications, **(2011)**.
[9] N. Medvidovic and G. Edwards, "Software architecture and mobility: A roadmap", The Journal of Systems and Software, vol. 83, **(2010)**, pp. 885-898.

