Software Architecture and Product Lines for Mobile Applications

Bhagavatula, Prasanth

pbhagav1@hawk.iit.edu

1. Introduction

Nowadays, Mobile phones are not merely used as a means of communication. Advanced technologies in hardware have drastically increased the storing, computing and data transmission capabilities of Mobile devices. Increase in the hardware capabilities and decrease in the costs of hardware created a wide spread popularity for Mobile devices. This makes the mobile devices as a convenient and logical access point for Mobile Applications. Many Mobile Applications using afore mentioned hardware capabilities have been developed in different areas of interest.

Regardless of the advancements in hardware, the Mobile devices cannot provide the capabilities of computer. Because of the small size of Mobile devices, limited hardware resources can be used, thus mobile devices have device limitations such as limited storage, limited processing power and limited working memory. These problems are exacerbated by the power constraints. Another major challenge for Mobile Application developers is the heterogeneity of the devices. In 2016 Q1 of all the Smartphones sales, 84% of mobile devices run on Android, 15% run on iOS and close to 1% of phones run on Windows 10/Phone [1]. The application needs to be portable in order to run on different platforms.

Mobile Application developers can overcome the different challenges of developing efficient Mobile Applications using either a good Software Architecture or using a Product Line to reuse multiple components. Software Architecture, whether it is for computer software or mobile application, plays a key role in providing the required quality of service. Finding the right design flow is one of the major design techniques in Mobile Application development.

According to SEI, “A Software Product Line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way” [2]. Product Lines enables us to re-use a lot of developed, tested and implemented components which can be integrated to produce a different product. This will substantially reduce the development costs and time-to-market of the application.

In the following sections, we will see different Software Architectures for three different kinds of Mobile Applications. We will also look at a case-study of how Nokia developed a Product line of Mobile browsers and a Robust SPL architecture for data collection.

2. Discussion of relevant topics

In this section we will discuss about the different Software Architectures and Product Lines for Mobile Applications. The details presented below are referred from research papers published in different IEEE conferences.

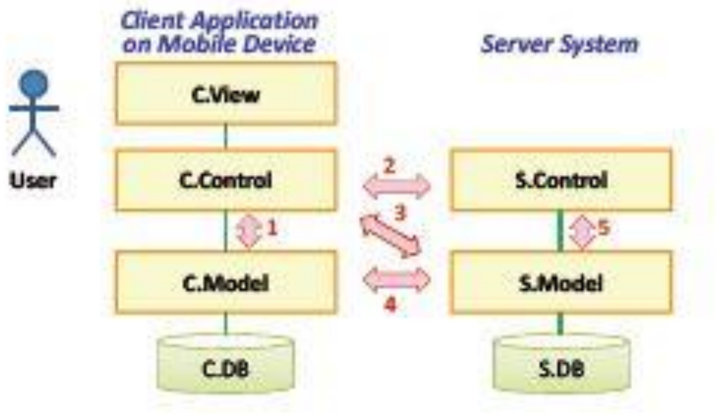
2.1.Software Architecture for Mobile Applications

Each and every Mobile Application has different set of requirements which calls for different Software Architectures. In this sub-section we will see how different Software Architectures are applied for different categories of Mobile Applications. First, we will see on how to develop a simple and efficient Mobile Application. For a data-oriented mobile application, we will discuss and compare two different software architectures. We will end this sub-section by discussing a lightweight architecture for Business applications.

2.1.1. Efficiency-centric design methodology

This paper [3] discusses a framework for designing mobile application architectures. Different variations of the conventional Model-View-Controller (MVC) architecture are presented. Design guidelines are presented which will be taken as input for choosing the most optimal architectural pattern by considering the target mobile application.

Below figure shows the conventional MVC architecture applicable for Mobile Applications.



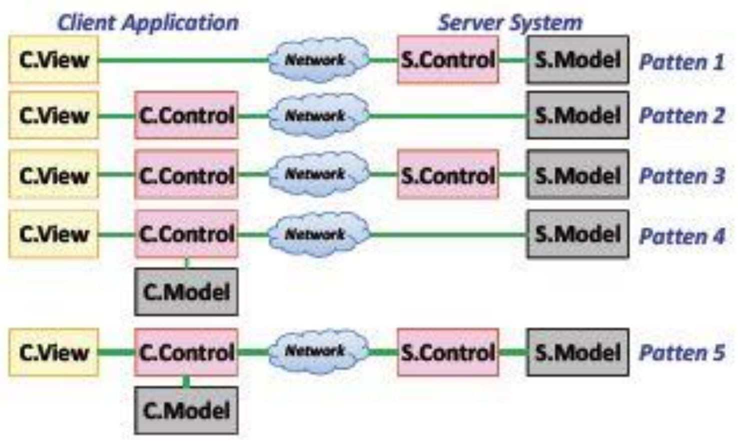
The Client side of the application will have a View layer, which represents the basic UI screens. The Control layer on the client can perform multiple tasks such as the screen touch option processing, I/O processing, data processing etc. The Client side can have a Model layer to store data locally. If the application uses basic user profile information, it can be stored in the Client device instead of storing it on the Server.

The Server side does not need a View layer since there is nothing to be displayed. The Control layer on the Server side will interact with Control layer on Client side and Model layer on the Server side and this layer majorly implements complex business logic and data processing. The Server side Model layer stores the data required for the application.

All possible interaction paths between different layers are identified and indicated in the figure. The following will provide details about the interaction paths.

- **Client Control – Client Model**: Represents the functionality that can be done locally without the need of interacting with Server.
- **Client Control – Server Control**: Represents the functionality of Client Control supported by Server Control.
- **Client Control – Server Model**: Represents the functionality that does not require interaction with Server Control. The case of updating data objects on Client side does not require interaction with Server Control.
- **Client Model – Server Model**: Represents the functionality that does not require any processing logic. In case of data synchronizations between Client and Server, no Control layer interactions are required.
- **Server Control – Server Model**: Represents the functionality that is performed by Server.

One of the challenging tasks in developing an application is the implementation of the business logic in the Control layer. Three scenarios are possible which are implementing business logic only in Client Control, only in Server Control or distributing logic across Client Control and Server Control. Also, the Client Model may be or may not be present based on the application requirements. Based on the above points, five architectural patterns have been derived from the conventional MVC architecture.



Each of the patterns is discussed below.

- **Pattern-1**: This has only Server Control which means that the entire business logic is present on the Server side. This is applicable for a thin Client application, but there will be lot of network interactions.
- **Pattern-2**: This has only Client Control which means that the entire business logic is present on the Client side. This pattern is ideal for applications where there is less business logic and less network interactions with Server Model.
- **Pattern-3**: This has both Client Control and Server Control and the business logic is distributed across the two layers. This pattern is used for such applications where the business logic can be parallelized across Client and Server.

- **Pattern-4**: This pattern has only Client Control and a Client Model. This pattern is similar to Pattern-2 and is useful for applications where the data, which causes lots of interactions between Client Control and Server Model, is replicated onto Client Model.
- **Pattern-5**: This pattern is similar to Pattern-3 and this has a Client Model layer. This design pattern is chosen when there are lots of data interactions and the business logic is parallelized.

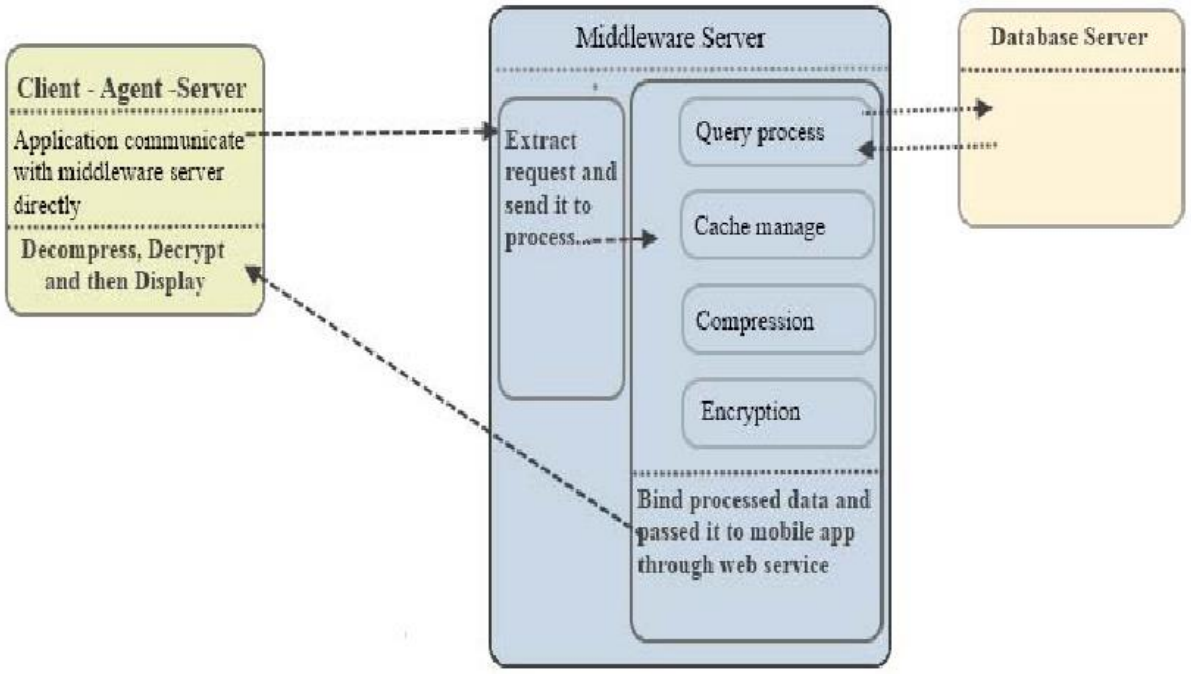
Further in the paper design criteria, design guidelines and design refinement guidelines are defined. Based on the target application requirements, applicable design pattern is chosen using the guidelines as input. Experiments are conducted for evaluating the various design patterns for different scenarios. Response time and Turnaround time are considered as factors for evaluating time efficiency and Memory consumption and Battery consumption are considered as factors for evaluating resource efficiency.

2.1.2. Software Architectures for Data-Oriented Mobile Applications

Mobile applications have developed into a non-trivial ubiquitous platform for social and commercial purposes. Even with the popularization of mobile computing, support for complex or extensive applications in mobile phones which make use of data from remote sources or need remote computing capabilities still require servers and/or middleware.

Various software models have been proposed for addressing the development of mobile applications that interact and utilize enterprise data sources; these include the client-server (C/S), client-agent-server (C/A/S), client-intercept-server (C/I/S), peer to peer, and mobile agent models. In this paper [4], client-agent-server (C/A/S) and client-intercept-server (C/I/S) architectures are discussed and compared.

Below figure presents the architecture of Client-Agent-Server model.

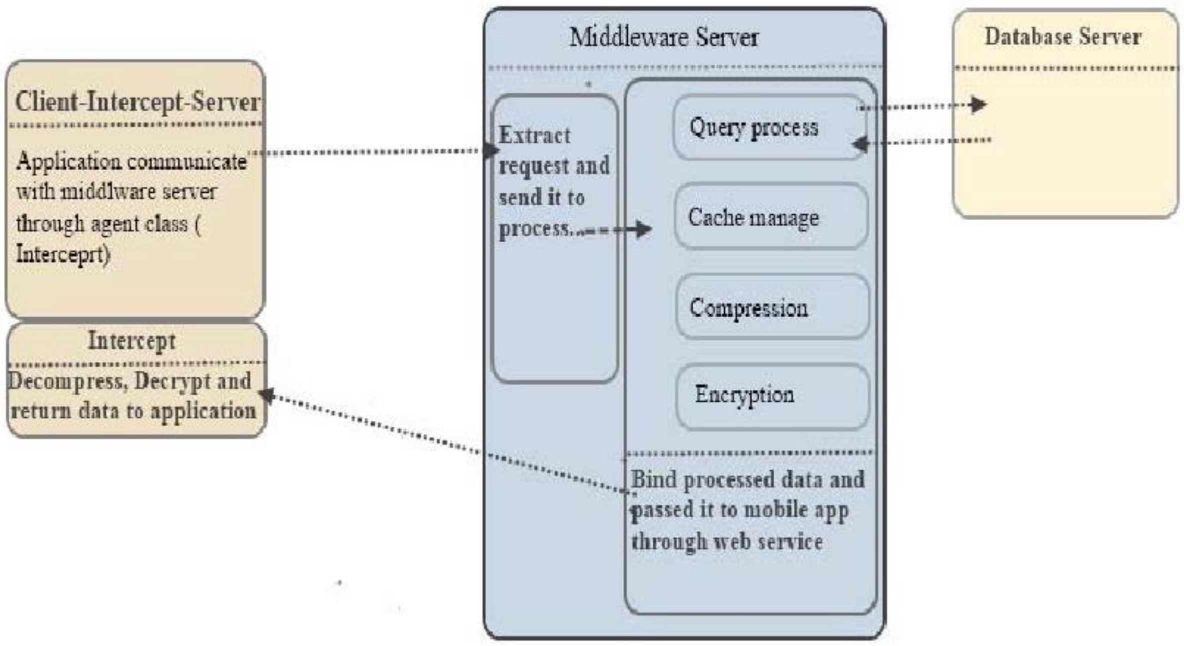

Client - Agent -Server
Application communicate with middleware server directly
Decompress, Decrypt and then Display
Middleware Server
Extract request and send it to process...
Query process
Cache manage
Compression
Encryption
Bind processed data and passed it to mobile app through web service
Database Server

The C/A/S architecture is a popular extension of client-server model and has three-tier architecture. The client is connected to agent, which acts a mobile host. The agent splits the interaction between client and server into two parts, interaction between client and agent and interaction between agent and server.

The advantages of the C/A/S architecture are:

- The client can be connected to multiple agents as required by the application.
- The agent can implement caching techniques to store the frequently accessed data in itself, thus avoiding interactions with database servers.
- The agent can also implement compression and encryption techniques to improve data transmission and data security and improving the overall Quality of Service.
- Complex business logic is present on the agent which makes the application thin on the client side.

In spite of numerous advantages, this model fails to sustain the current computation at the client during periods of disconnection. Also, the data transmission can be optimized only by agent while transmitting data from agent to client but not the other way around.


Client-Intercept-Server
Application communicate with middlware server through agent class (Interceprt)
Intercept
Decompress, Decrypt and return data to application
Middleware Server
Extract request and send it to process...
Query process
Cache manage
Compression
Encryption
Bind processed data and passed it to mobile app through web service
Database Server

Above figure represents the client-intercept-server architecture. This model proposes deploying an agent on the client side too. The client side agent will intercept the requests from client and runs in parallel with the server side agent. From a client perspective, the client side agent appears as a local proxy server.

The advantages of this model are:

- Data transmission optimization can be performed by both client side agent and server side agent.

- Similar to data transmission optimization, caching techniques can also be implemented on client side agent and thus avoiding any network interaction while accessing cached data.

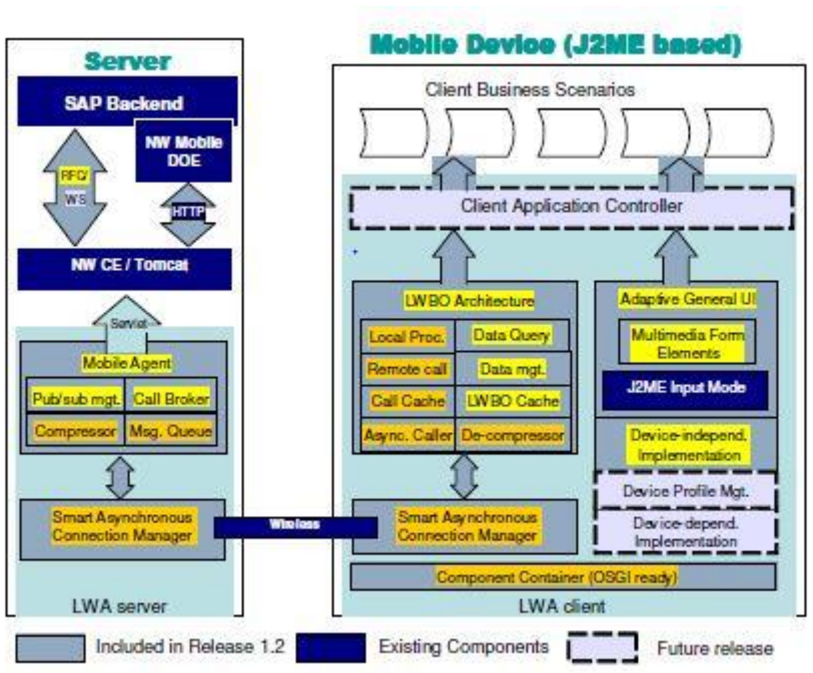
The major drawback in this model is implementing a client side agent which means running business logic on client side and thus will consume more resources than the application developed using C/A/S model.

Experiments are conducted by developing a remote database, a middle agent server and two mobile applications, one uses C/A/S architecture and the other uses C/I/S architecture, for two different domains. In summary, the application with C/A/S architecture performs better than the application with C/I/S architecture but only by a small margin.

2.1.3. Integrated Lightweight Software Architecture for Business Applications

Advancement in performance capabilities of Mobile devices calls for development of business applications. Business applications, in general, have very complex logic and mobile devices still have limitations on data storage, network access, processing power etc. This paper [5] provides a component-based framework to develop complex business applications that run on mobile devices.

Below figure shows the Lightweight architecture and its components.



The Lightweight architecture (LWA) contains the following fundamental components: container of components, registry of services and components, controller, communication manager (simple version) with HTTP and Bluetooth channels, server call cache, data manager (simple version), XML de-compressor, and a log manager (simple version).

The LWA components use XML compression techniques for storing data in XML-based format as lightweight objects. These objects will be used by the client application to provide offline support. An asynchronous remote invocation mechanism via multi-channel connection management component supports online invocations to the server side business logic.

LWA is divided into LWA server and LWA client parts. Business objects are serialized and transmitted from servers to mobile clients as XML messages. Data is obtained in the backend by RFC calls. Due to the amount of data and the complexity of business objects, data must be compressed. It is then transferred to the connection manager which is responsible for sending the compressed data to the client side.

In LWA, a component means an element that provides a well-defined function or a set of tightly correlated functions, and is able to interact with other components. From a practical perspective, a component is a set of collaborative and reusable sections of code, i.e. a set of Java Classes in our implementation. The Core components of LWA are container, registry, and controller.

- **Container**: The container is a Java-based framework to load and run components. It provides generic functionality such as component loading and activation, and resolves references and dependencies between components
- **Registry**: The registry component of LWA provides three registries: component registry, application registry, and service registration. Component registry and application registry provide the full list of all activated components and applications in a running LWA container. Although applications are also components, a separate registry is built for applications for easy management. Service registry is a list of all services, explicitly activated or implicitly loaded because of dependency links between components.
- **Controller**: LWA uses a set of controller components to separate components from events triggered by user interactions and incoming/outgoing messages. Typically, a controller dispatches events from UI to a component or from one component to another.

In the Lightweight Business Object (LWBO) model, the client side of the framework is meant to be lightweight. All heavy computation is done on the server side. Each BO instance accessed by the client application has a corresponding LWBO instance, which is a partial copy of the BO. On the client side, users access BOs through corresponding LWBOs.

In LWA, data exchange is implemented using loosely coupled process components that communicate through asynchronous message exchange. These process components are independent of other components. Their business objects are accessible via published service interfaces. Communication through dedicated interfaces simplifies process modelling and makes the process flow transparent. The connection manager provides an API to check the current status and the real bandwidth of every connection. In a business context all messages should be precise and accurate and checksum techniques can be used to validate the message.

In summary, the paper introduces a Lightweight Architecture for developing mobile business applications. The paper discusses three major components of the architecture, client model component, data management component and communication management component.

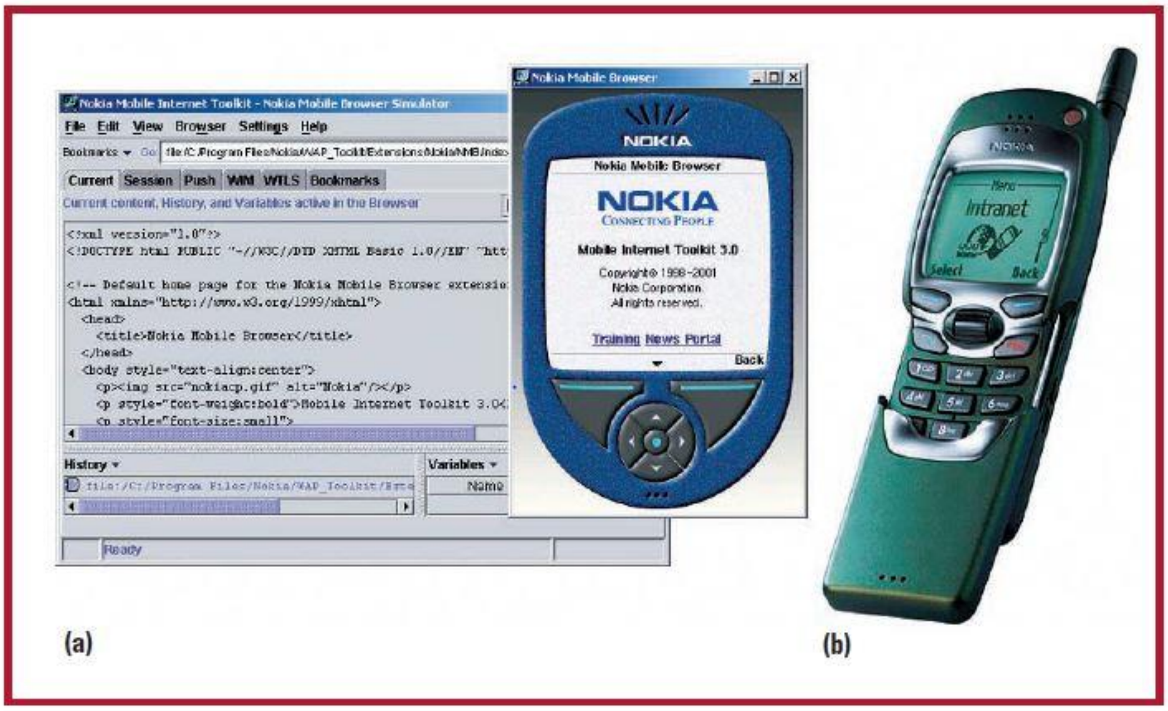
2.2. Product Lines for Mobile Applications

A Software Product Line is a set of systems that share some common set of features. Software product lines are emerging as a viable and important development paradigm allowing companies to realize order-of-magnitude improvements in time to market, cost, productivity, quality, and other business drivers. Software product line engineering can also enable rapid market entry and flexible response, and provide a capability for mass customization. In the following sub-sections, we will see how Nokia developed Mobile Browsers in a Product Line. We will also discuss how a Robust SPL architecture can be used in developing applications for data collection domain.

2.2.1. Developing Mobile Browsers in a Product Line

Nokia used a product line to develop mobile browser products [6] that let mobile phone or personal digital assistant users access services over wireless telecommunications networks. They developed the technology first for their own handsets and later distributed it as a software product. There were two reasons to initiate the product line. First one is to serve an increasingly heterogeneous customer base and secondly, they wanted to benefit from large-scale reuse.

In 1999, Nokia launched the Nokia Browsers and Tools product line. The technology was initially based on the Wireless Application Protocol specification, which specifies, among other things, mark-up and scripting languages and a communication protocol. They started by developing a WAP browser and toolkit. Then they extended the product family to include three browser products and a multimode toolkit product.

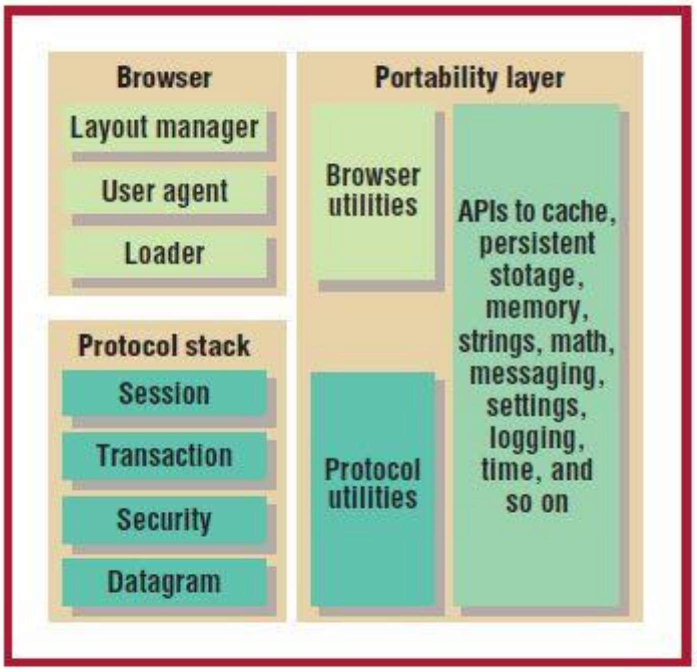

Nokia Mobile Internet Toolkit - Nokia Mobile Browser Simulator
File Edit View Browser Settings Help
Bookmarks Go file:/C:/Program Files/Nokia/WAP_Toolkit/Extensions/Nokia/NMB/index
Current Session Push WIM WTLS Bookmarks
Current content, History, and Variables active in the Browser

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN" "htt
<!-- Default home page for the Nokia Mobile Browser extensio
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Nokia Mobile Browser</title>
  </head>
  <body style="text-align:center">
    <p><img src="nokiacp.gif" alt="Nokia"/></p>
    <p style="font-weight:bold">Mobile Internet Toolkit 3.0<
    <p style="font-size:small">
```

History
file:/C:/Program Files/Nokia/WAP_Toolkit/Exte
Variables
Name
Ready
Nokia Mobile Browser
NOKIA
Nokia Mobile Browser
NOKIA
CONNECTING PEOPLE
Mobile Internet Toolkit 3.0
Copyright© 1998-2001
Nokia Corporation.
All rights reserved.
Training News Portal
Back
NOKIA
Menu
Intranet
Select
Back
(a)
(b)

The generic version of the browser is named Nokia Mobile Browser. The Nokia operating system version of the browser included Nokia-platform-specific adaptations and interfaces. They tailored the Symbian OS version of the browser for the Symbian operating system with C++ wrappers and Symbian extensions. The Nokia Mobile Internet Toolkit was a phone simulator coupled with a development environment running on Windows platforms. By the end of 2001, Nokia had several customers for all the embedded browsers, and the toolkit had approximately 500,000 registered users.

Nokia had many common requirements for all the products. These included requirements for implementing the Extensible Hypertext Markup Language (XHTML) with Cascading Style Sheets and the Wireless Markup Language 1.x browser, supporting both languages natively. All products required connectivity through the specified protocol, and Nokia tested all of them for interoperability with wireless gateways. Then Nokia decided to organize the common parts into a product line. Thus began the Nokia Browsers and Tools product line.



Nokia selected the core browser components and the test protocol stack to form the initial product line. They first enhanced these components to meet the key product requirements of full WAP implementation, portability, and product quality. Finally, they isolated platform-dependent code and APIs into a portability layer. They cleaned, tested, verified, and documented the acquired components to achieve product quality.

After eight months of clean-up, implementation, testing, and documentation, Nokia released the 1.0 version of the generic browser product from the product line. By then, they had a pool of core assets on which future products can be built. The fact that most of the core assets were already being used in existing products demonstrated that the technology worked.

2.2.2. Robust SPL Architecture for Data Collection

Software Product Line (SPL) is an approach for software development that allows variability management, by reusing a set of core assets to obtain a family of similar products. Different software engineering techniques such as Component-Based Development (CBD), Aspect-Oriented Software Development (AOSD) support SPL development coping with software complexity by supporting modularity.

Data collection is defined as the process of gathering and measuring information on variables of interest, in an established systematic fashion that enables one to answer stated research questions, test hypotheses, and evaluate outcomes. In this context, the SPL approach can support the systematic reuse of software assets to generate applications for these different sub-domains of data collection.

The paper [7] presents a robust SPL architecture for data collection called R-SPL-DC. The proposed solution involves two steps

- Generation of the feature model and the aspect-feature view
- Design of the Robust SPL architecture using the AO-FArM method

The data collection feature model is devised based on functional and non-functional requirements integrating some possible exceptions of the domain and fault tolerance techniques. After that, the Aspect-Feature model is obtained to identify crosscutting concerns in the domain using the AO-Analysis. Below figure represents the Feature Model for the Robust SPL for Data Collection.

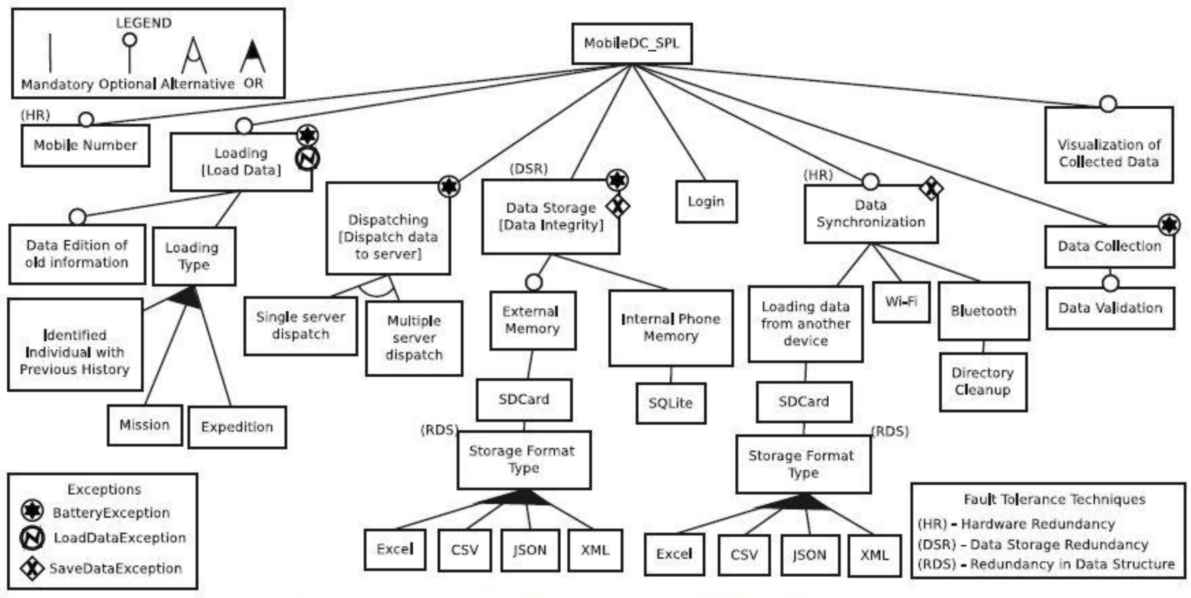


Fig. 1: Partial Feature Model for the Robust SPL for Data Collection.

Multiple features included and identified are:

- The Mobile Number represents the number of mobile devices that are taken to the field.
- The Loading feature is responsible for loading information from previous data collections.
- The Dispatching feature can be performed in two ways: using a Single Server or using Multiple Servers.
- Data Storage feature concentrates the rules to record data. It always records data in the internal memory, and optionally it does record the information in the external memory in a specific format.
- The Login feature authenticates the user.
- The Data Synchronization feature represents the data synchronization that is needed in the field,

- The feature Data Collection represents the data acquisition. It may use the optional validation feature, which checks for data consistency.
- Visualization of Collected Data is an optional feature that implements different formats to visualize the data, for example, using maps.

The next step involves identifying cross-cutting concerns using AO Analysis. AO-Analysis is a method that aims to identify the crosscutting concerns through the use of use cases and traceability matrices. Use cases that are "extended" or "included" frequently by others are candidates to crosscutting concerns. The traceability matrix looks for the impacts of crosscutting concerns over existing software features and other crosscutting concerns. The output of this method is a complementary view of the feature model called aspect-feature view.

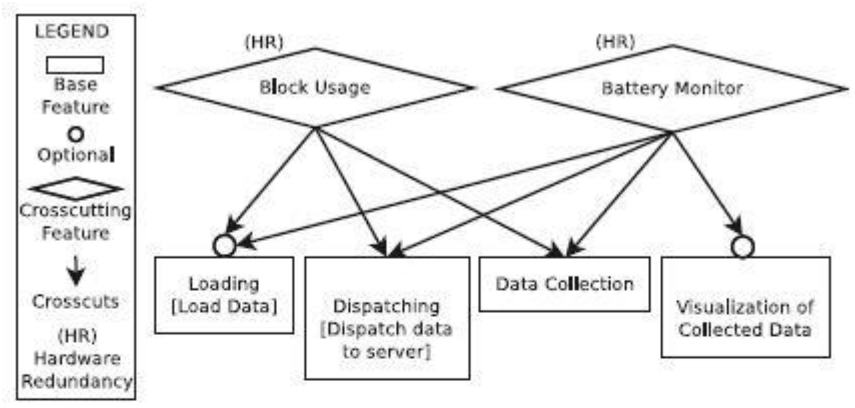


Fig. 2: Aspect-Feature View of the Robust SPL for Data Collection.

In the data collection domain, two crosscutting concerns were identified: Block Usage and Battery Monitor. Block Usage is a crosscutting feature that blocks the usage of base level some features (such as data collection and loading data from server) depending on a configurable battery level. If the battery level is lower than this value, features will not be allowed to execute and a battery exception will be thrown. Battery Monitor logs the current battery level into the internal database after each operation of "loading" or "pushing data" is sent to the server.

From this we can design a robust SPL architecture using AO-FArM method. Aspect-Oriented Feature Architecture Mapping (AO-FArM) is a method to obtain the software architecture based on the feature model and aspect-feature view. It proposes four transformations of the feature model:

- Removal of non-architecture related features and quality features resolution
- Transformations based on architectural requirements
- Transformations based on interacts relations
- Transformations based on hierarchy relations.

Below figure represents the R-SPL-DC architecture.

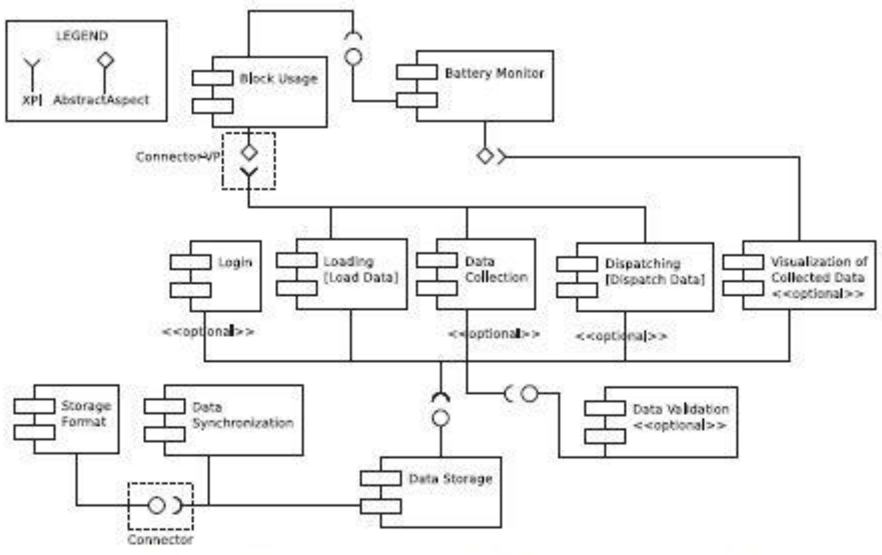


Fig. 3: Robust SPL Architecture for Data Collection Domain.

Each and every feature is mapped to a corresponding component. In the R-SPL-DC, the Data Storage component provides two APIs: one providing the Data Storage Redundancy, and another not providing Data Storage Redundancy. Therefore the variability of fault tolerance depends on the component client, i.e., it is up to the component client to decide to use or not the redundancy. For some applications, the data collection occurs in the city with high network availability; therefore the application does not need to store with redundancy in mobile. The application could store data locally only and use the Dispatching component to send data to the server in real time.

This paper presented a solution to generate a Robust SPL, a Robust SPL Architecture for Data Collection. The SPL architecture can be used to derive various data collection applications. It was created based on existing data collection applications and contains most of the basic features of the data collection domain; therefore it can generate applications using different combinations of the proposed components. Also it can be configured to allow levels of fault tolerance.

3. Conclusion

The Mobile Application Software Architecture is mainly dependent on the type and requirements of the application. For a generic mobile application itself, we have seen different patterns of the conventional MVC architecture. Different patterns have their own pros and cons and the best pattern mainly depends on the application requirements. For a data-oriented application, completely different architectures are considered and the paper discusses compares two of the architectures. Although the performance of C/A/S architecture is better than the C/I/S architecture, if the application uses caching it would be useful to use C/I/S architecture. On the other hand, Business applications have a completely different set of requirements and level of complexity which requires a new architecture. Because of the high level of complexity, we don’t want to run the business logic or data processing on the client side and the paper proposes a lightweight architecture for developing mobile business applications.

Although Product Lines provides less time to market, increase in productivity, decrease in cost of development, the actual benefits can be seen in the long run. We have seen two different ways of creating Product lines. One paper talks about the Product line involving Nokia, which developed different individual Mobile browser products initially. Then started identifying the core components which can be reused, cleaned and re-developed the components and then built multiple products on top the core components. The other paper provides a framework for creating a Product Line through a sequence of steps. Identifying the functional and non-functional requirements of the application is a key part for creating a Product Line out of it.

4. References

[1] https://en.wikipedia.org/wiki/Mobile_operating_system#World-Wide_Share_or_Shipments

[2] http://sei.cmu.edu/productlines/

[3] An efficiency-centric design methodology for mobile application architectures - Hyun Jung La; Ho Joong Lee; Soo Dong Kim

[4] A comparison of software architectures for data-oriented mobile applications - Md. Ashrafur Rahaman; Michael Bauer

[5] An Integrated Lightweight Software Architecture for Mobile Business Applications - Serhan Dagtas; Yuri Natchetoi; Huaigu Wu; Louenas Hamdi

[6] Developing mobile browsers in a product line - A. Jaaksi

[7] A Robust Software Product Line Architecture for Data Collection in Android Platform - Gustavo M. Waku; Edson R. Bollis; Cecilia M. F. Rubira; Ricardo da S. Torres