

April 2012

Performance of Data Transmission for mobile applications

Md Ashrafur Rahaman
The University of Western Ontario

Supervisor
Dr. Michael Bauer
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

© Md Ashrafur Rahaman 2012

Follow this and additional works at: <https://ir.lib.uwo.ca/etd>

Recommended Citation

Rahaman, Md Ashrafur, "Performance of Data Transmission for mobile applications" (2012). *Electronic Thesis and Dissertation Repository*. 408.
<https://ir.lib.uwo.ca/etd/408>

This Dissertation/Thesis is brought to you for free and open access by Scholarship@Western. It has been accepted for inclusion in Electronic Thesis and Dissertation Repository by an authorized administrator of Scholarship@Western. For more information, please contact tadam@uwo.ca.

PERFORMANCE OF DATA TRANSMISSION FOR MOBILE APPLICATIONS
(Spine title: Data transmission methods and techniques for mobile application)

(Thesis format: Monograph)

by

Md. Ashrafur Rahaman

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

© Ashrafur Rahaman 2012

CERTIFICATE OF EXAMINATION

Supervisor

Examiners

Dr. Michael Bauer

Dr. Hanan Lutfiyya

Supervisory Committee

Dr. Mike Katchabaw

Dr. Xianbin Wang

The thesis by

Md. Ashrafur Rahaman

entitled:

**Performance of Data Transmission
for mobile application**

is accepted in partial fulfillment of the
requirements for the degree of
Master of Science

Date

Chair of the Thesis Examination Board

Abstract

Mobile applications have empowered and extended the usability of mobile devices far beyond merely supporting voice communication. The development of mobile applications, however, must deal with a variety of unique problems: limited working memory, limited storage, limited processing power, and small screen size. Mobile applications which rely on remote data sources and databases are particularly challenging given the need to transmit data through wireless media and often involve complex business logic. Our main goal is to improve the performance of mobile applications which rely on remote data sources and databases. In this research work, we compare different data transmission optimization techniques, different middleware approaches and identify combinations of approaches for improving performance of data transmission over wireless network. The results of this research provide useful guidelines for the development of mobile applications needing to connect to remote databases or data sources.

Keywords

QoS of data transmission, Mobile application, Mobile data transmission models, mobile data transmission techniques, mobile computing, middleware, mobile application API.

For my parents Mrs. Ashrafunnessa and Mr. Md. Fazlur Rahaman.

Acknowledgments

First of all I would like to thank my supervisor, Prof. Dr. Michael Bauer, for his guidance and assistance. He was very helpful and encouraging in choosing the topic of this thesis and guiding me in my research. Thanks to him for his help and inspiration. I would like to give a nod to my fellow researchers for being an excellent sounding board for both ideas and the venting of frustrations. I would also like to thank faculty members and staff of the Department of Computer Science for making my stay here highly enjoyable. Many thanks go to my beloved family and friends for their understanding and endless support.

Table of Contents

CERTIFICATE OF EXAMINATION	ii
Abstract	iii
Acknowledgments.....	v
Table of Contents	vi
List of Figures	ix
List of Tables	xi
Chapter 1	1
1 Introduction	1
Chapter 2.....	5
2 Related Work on Middleware Models and Techniques for Mobile Applications	5
Chapter 3.....	11
3 Software Architectures for Mobile Application.....	11
3.1 Introduction.....	11
3.2 Client-Server (C/S) Model.....	12
3.3 Client-Agent-Server (C/A/S) Model.....	13
3.4 Client-Intercept-Server (C/I/S) Model.....	15
3.5 Peer-to-Peer Model	16
3.6 Mobile-Agent Model	16
3.7 API.....	17
Chapter 4.....	19
4 Implementation	19

4.1 Introduction.....	19
4.2 Databases	21
4.3 Middleware and Middleware API.....	22
4.3.1 Caching.....	22
4.3.2 Compression	24
4.3.3 Encryption.....	25
4.3.4 Overview of the Middleware	26
4.4 Mobile Agent API.....	27
4.4.1 Implementation of C/A/S.....	28
4.4.2 Implementation of C/I/S	30
4.5 Summary.....	32
Chapter 5.....	33
5 Experiments and Results.....	33
5.1 Experimental Setup.....	34
5.2 Experiments with Inventory System.....	35
5.3 Experiments with a Medical Information System	37
5.3.1 Experiment 1: Basic Experiment	40
5.3.2 Experiment 2: Use of caching.....	41
5.3.3 Experiment 3: Use of compression.....	43
5.3.4 Experiment 4: Experiments with encryption	45
5.3.5 4-Factor Analysis	47
5.4 Summary.....	50
Chapter 6.....	51
6 Discussion of Results.....	51
6.1 Realizing Data Transmission Techniques.....	51

6.1.1	Caching	51
6.1.2	Compression	52
6.1.3	Encryption.....	53
6.1.4	Combination of techniques	53
6.1.5	Mobile software architectures.....	54
Chapter 7	55
7	Conclusion	55

List of Figures

Figure 3-1: Common rich client mobile application architecture.....	12
Figure 3-2: The Client-Agent-Server (C/A/S) model	13
Figure 3-3: Client-Agent-Server (C/A/S) model with remote database	14
Figure 3-4: The Client-Intercept-Server (C/I/S) model	15
Figure 3-5: Client-Intercept-Server (C/I/S) model with remote database	16
Figure 3-6: The Mobile Agent model	17
Figure 4-1: High level overview of the system.....	19
Figure 4-2: Overview of Client-Agent-Server request and response.....	20
Figure 4-3: Overview of Client-Intercept-Server request and response	21
Figure 4-4: Pseudo code to retrieve data from remote data sources	22
Figure 4-5: Middleware pseudo-code to retrieve or create cache data	23
Figure 4-6: Middleware pseudo-code to compress retrieved data	25
Figure 4-7: Middleware pseudo-code for encrypting retrieved data	26
Figure 4-8: Pseudo-code of service to retrieve data in middleware.....	27
Figure 4-9: J2ME libraries and third party libraries for C/A/S application.....	29
Figure 4-10: Command actions to generate mobile data	29
Figure 4-11: Data process and display in Client-Agent-Server	30

Figure 4-12: Client-Intercept-Server mobile application..... 31

Figure 4-13: Intercept API..... 31

List of Tables

Table 5-1: List of development tools for the experimental environment	34
Table 5-2: Communications between the mobile application, middleware server, and remote database server.....	35
Table 5-3: Steps of the inventory system mobile application.....	36
Table 5-4: Steps in different scenarios of inventory system.....	37
Table 5-5: Inventory system experiment result for scenario 1 and scenario 2	37
Table 5-6: Medical Information System step by step	39
Table 5-7: Experiment 1 scenarios and data sizes	40
Table 5-8: Regular experiment result and analysis.....	41
Table 5-9: Results of the analysis of variance (ANOVA) for Experiment 1	41
Table 5-10: Experiment with caching result and analysis	42
Table 5-11: Three-factor analysis on software model, scenario, and caching.....	42
Table 5-12: Results of the analysis of variance (ANOVA) of experiment with caching .	43
Table 5-13: Change of data size after data compression	43
Table 5-14: Experiment with compression results and analysis.....	44
Table 5-15: Experimental results for software models, scenarios, and compression	44
Table 5-16: Results of the analysis of variance (ANOVA) of experiments with compression.	45
Table 5-17: Change of data size after encryption	45

Table 5-18: Experiment with encryption result and analysis.....	46
Table 5-19: Three factor analysis on software models, scenarios, and encryption	46
Table 5-20: Results of the analysis of variance (ANOVA) of experiment with encryption.	47
Table 5-21: Experiment design of 4-factor analysis	48
Table 5-22: The results of the analysis of variance (ANOVA) on the large data scenario.	49
Table 5-23: The results of the analysis of variance (ANOVA) on small data scenario of the four-factor analysis	49

Chapter 1

Introduction

From its original inception as an accessory for mobile phones, mobile applications have developed into a non-trivial ubiquitous platform for social and commercial purposes [1], [2], [3], [4]. Mobile applications have empowered and extended the usability mobile devices far beyond merely supporting voice communications.

Apple has proven that the mobile application market should not be underestimated and can represent an important revenue stream. The revolutionary App Store offers more than 500,000 apps to iPhone, iPod and iPad users in 90 countries around the world; more than 18 billion applications downloaded by October 2011 from the Apple Store [5]. Following Apple's lead, Palm Inc. opened an application store for Palm devices in June, 2009, Nokia has opened the "Ovi Store", Samsung has created Samsung Apps, and Research in Motion (RIM) also launched its application store, BlackBerry App World. Microsoft also launched its own version of the Application Store called SkyMarket with Windows Mobile (WM7). The Android Marketplace developed by Google also became well known in a short time; they started on March 2009 with 2300 applications and by January 2012 the total number of application reached to around 400,000 [6] with the record of more than 10 billion [7] downloads.

The current mobile development market is dominated by: Google (50.9%) with Android, a Linux-based operating system for mobile devices; Apple (23.9%) with iOS; Accenture (11.7%) its Symbian OS; RIM (8.8%) with its Blackberry OS; Microsoft (1.9%) with its Windows Phone Operating System; and Samsung Electronics (2.1%) with Bada Mobile operating system [8].

Each mobile OS offers a software development kit (SDK) which is generally composed of an integrated development environment (IDE), an emulator, specific libraries, and other tools. Although the development environment and programming language are similar to that in the PC environment, the program structures are very specific for each mobile OS.

With the popularization of mobile computing, many developers have faced problems due to great heterogeneity of devices [9]. In a mobile environment, the wireless communication is still intermittent, mobile devices have hardware restrictions such as: limited working memory, limited storage, limited processing power, and small screen; energy in mobile devices are very limited too. This means that developing large scale mobile applications which can connect to remote data sources or databases through wireless connections with high computational business logic must take into account these limitations. 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.

Popular mobile relational database systems like IBM's DB2 Everywhere 1.0, Oracle Lite, Sybase's SQL etc. work on Palm top and hand held devices (Windows CE devices) and provide local data storage for relational data acquired from enterprise relational databases. The main constraints for such databases relate to the size of the devices' memory and size of the program, as handheld devices have memory constraints [10]. Moreover, enterprise databases cannot be replaced by these mobile relational databases.

To support extensive applications in mobile phones that require retrieval of data from remote data sources, middleware is needed which has the capacity to deal with mobile agents and remote database servers, and can improve the transmission of data by implementing caching, pre-fetching, data prioritizing and data compression techniques. Mobile applications or agents can access the middleware through an API to make the communication between mobile agent and middleware transparent.

In our research, we investigate difference in middleware approaches to retrieve data from remote databases or data sources and transmit data to handheld devices. To analyze the performance of data transmission, different data transmission techniques has been used with different middleware models. The study includes performance analysis of data transmission in two different domains with different scenarios. Through experiments, we identify approaches or combinations of techniques that perform best for mobile application. We also conduct experiments on data security and large volume data transfer techniques over wireless network.

We compare the performance of two common software architectures (Client-Agent-Server and Client-Intercept-Server) used for the development of mobile applications requiring access to middleware. We also examine the performance impact, in terms of transmission time, of using caching, compression, encryption, and/or combination of these techniques in mobile applications relying access to remote data. Our quantitative analyses examine a mobile application running on a mobile phone, connecting to a middleware server hosted remotely with a database server located on another remote host. Thus, our experiments are performed in real life scenarios instead of a laboratory setup. The experiment results provide guidelines to the mobile application developers about the software models and techniques or combination of technologies to use for QoS of data transmission for mobile applications.

As part of our work, we also developed a middleware API and a mobile application API which can be used by others to help simplify the development of mobile applications that need to access data from remote data sources. By using the API, software designers can use the functionalities provided by the middleware and applications already developed.

In Chapter 2, we review the existing literature on some of the architectural approaches for supporting mobile applications and then look at the middleware with data transmission techniques. As we develop an API to simplify development of high computing mobile application, we review API proposed for middleware and literature on qualitative and quantitative analysis of mobile application data transmission for remote data sources.

In Chapter 3, we explain different mobile application software architectures has been proposed last two decade, middleware techniques and importance of API for mobile application development.

In Chapter 4, we present the system we have developed to analyze data transmission for mobile application. We explain the pseudo codes, algorithms, tools, and software we use to develop the middleware API and mobile application API. Also, we describe our system architecture in this chapter.

In Chapter 6 we discuss the experiment we have conducted and the results of the experiments. Also, we represent our findings from the experiments.

In Chapter 7 we represent the conclusion and future works of this research work.

Chapter 2

Related Work on Middleware Models and Techniques for Mobile Applications

During the last decade, several research efforts have focused on middleware and data transmission techniques for mobile applications. We first review some of the architectural approaches for supporting mobile applications and then look at the middleware with other data transmission techniques.

To accommodate the new computing paradigm introduced by mobile wireless computing, various software models have been proposed including 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 Chapter 3 we discuss these models in detail.

In [29], Spyrou et al. qualitatively and quantitatively analyzed a set of software models built on the client/server model or mobile agents for accessing a Web server and proposed new techniques mixing the client-server models and mobile agent techniques. They compared the C/A/S and C/I/S models in the context of browsers and web servers for wired network. According to their experimental results, the C/A/S model requires considerably less time than any other client/server model. Though C/I/S model lacks in performance due to the additional creation of the client-side agent, it supports compatibility, since it can be built on top of existing applications. According to the researchers, the C/I/S model provides more flexibility than C/A/S and that should have been translated to better performance; C/I/S model performs better for heavy-weight clients with large computational power. In our research work, we look at both the C/A/S

and C/I/S models in the context of mobile applications where there is a need to access data sources or databases are in remote locations. In particular, we compare the performance of the C/A/S and C/I/S models in dealing with remote data sources and databases for mobile applications considering different data transmission techniques (caching, encryption, and compression) in two domains (medical and general inventory) with different scenarios (large data set and small data set).

Transmission of a large amount of data through a wireless network to a memory limited mobile application is a real challenge. Researchers have proposed different middleware approaches to meet these challenges. While there has been previous work on mobile applications and supporting middleware, there has been little quantitative work comparing different approaches and use of different techniques on the performance of data transmission; the current research begins to fill this gap. Some of the previous research related to middleware is described in the following.

Capra and Mascolo [11], [12] identify requirements for middleware that supports mobility. They discuss the main characteristics and differences between wired and wireless environments that impact mobility, and categorize and study some possible middleware solutions. They propose the use of reflection capabilities and meta-data to pave the way for a new generation of middleware platforms designed to support mobility. They suggest the use of a reflexive middleware [13] in a mobile computing environment to solve the problem of losing network connectivity during the movement of the mobile devices. In [14], they describe a reflexive middleware for ad hoc networks named XMIDDLE that allows data sharing and uses XML to encapsulate information but it does not clearly distinguish the client and server applications. In [15], they present another middleware named CARISMA which allows changes in the context based on a set of rules. Both CARISMA and XMIDDLE middleware are based on client-server model.

Campbell et al. [16] propose a middleware, named Mobeware, to support multimedia applications, considering QoS requirements, in a mobile environment. The platform is built on a distributed system and Java technology and uses adaptive algorithms to support QoS controlled mobility. The goal of the Mobeware adaptive algorithms is to provide

scalable transport flows, reduce handoff dropping and improve wireless resource utilization. Mobiware provides an adaptive transport API for the middleware which follows client-server approach to select, dispatch, bootstrap, configure and tune multimedia objects.

Bellavista et al. [17] present a middleware for mobile users, also based on mobile agents, to keep services running regardless of a user's mobility. The middleware provides services to support a virtual environment, virtual terminal, and resource manager.

In [18], Chan and Chuang propose MobiPADS, a middleware that uses information from the physical environment to perform self-configuration. It allows dynamic adaptation in both the application and the middleware itself. MobiPADS has two parts: a client at a mobile device attached to the Internet through wireless or cellular networks and a server at the wired network. This extended server-client application is designed to support multiple MobiPADS clients and is responsible for most of the optimization computation. MobiPADS collects metrics about the environment such as bandwidth, latency, and processor usage, and notifies the applications that use those data.

Middleware for mobile computing environments is discussed in [19], where the authors addressed the problems of the heterogeneity of devices in such environments. To this end, they present a middleware for multi-client and multi-server mobile applications, taking into account the resource restrictions of mobile devices. At the client side, application development is done using a lightweight, robust and portable platform. At the server side, the middleware allows high scalability and the full use of resources of the machines it is installed. The proposed middleware provides a transparent communication API to develop applications for mobile environments and allows applications to divide their work amongst server and client sides. The middleware simplifies the development of applications for mobile computing. By using the API, software designers can use the functionalities provided by the middleware or applications already developed. The middleware is based on the extended C/S model (C/A/S model) and provides communication primitives for applications to exchange messages transparently between the application in mobile device and the middleware, and independently of a specific

communication protocol. The proposed middleware simplifies the development of applications for mobile computing, and, through the API, software designers can use the functionalities provided by the middleware or application already developed. The researchers, however, did not quantitatively compare the performance of the C/A/S and C/I/S models for their applications.

Gehlen and Mavronsmatis [20] proposed a mobile web service middleware, which supports the publishing and discovery of web services and enables the efficient usage of a network resource through context monitoring. Here, the context awareness is realized by a rule based data monitoring in order to minimize the communication frequency over the mobile links and, thus, to minimize the runtime costs of the mobile application. However, it does not support the real-time change of context.

In [21], researchers analyze the performance of the Wireless Application Protocol (WAP) based access compared to HTTP in terms of criteria such as: latency, data transfer, memory footprint and CPU power requirements for accessing XML Web services from mobile clients. The analysis showed that the performance in term of latency can be increased using a WAP binding to SOAP. With WAP it is possible to access all existing HTTP web services. The SOAP messages are reduced by a third using Wireless Binary XML (WBXML) encoding [22]. WBXML was developed mainly for low bandwidth networks and restricted Central Processing Unit (CPU) power, and seems appropriate for mobile XML messaging. The connectionless Wireless Session Protocol (WSP) avoids TCP's three way handshaking, header volume reduction and optional reduction of the payload data volume; it reduces the protocol data overhead by more than a third compared to the HTTP.

Caching and pre-fetching on the edge of the Internet has been one of the most popular techniques to improve the scalability and efficiency of delivering dynamic web content. Leveraging these techniques for mobile application to improve application's data transmission efficiency has been discussed in several research papers.

Gupta et al. [23] concentrated on the aspects of data management over mobile ad-hoc networks and proposed to estimate the global distribution and then predict and cache the

most popular data in the hope of being able to provide it to other devices when asked by them.

Along similar lines, Yin and Cao [24] propose a cooperative caching scheme for mobile ad hoc networks using caching of popular data so that the availability in mobile ad hoc networks is increased. Specifically, they propose three schemes: CachePath, CacheData and HybridCache. In CacheData, intermediate nodes cache the data to serve future requests instead of fetching data from the data center. In CachePath, mobile nodes cache the data path and use it to redirect future requests to the nearby node which has the data instead of the faraway data center. HybridCache improve the performance by taking advantage of CacheData and CachePath while avoiding their weaknesses. According to their simulation results, the proposed schemes can significantly improve the performance in terms of query delay and message complexity when compared to other caching schemes.

In [25], researchers propose anticipatory retrieval of data, and to cache data that is likely to be requested later. Here, caching is done asynchronously in the background during times of high bandwidth. They propose algorithms to assess the semantic relevance of the data using semantic distances along with user priorities and availability of bandwidth, and then prioritizes anticipatory data downloads on to the cloud's storage based on the relevance quotient. Here, data prioritization and reprioritization depends on data size, priority and availability. The model provides better performance, adapts to varying bandwidth, and pre fetches data from cloud with better accuracy and relatively little overhead.

Data transmission in the wireless network can be vulnerable to security attacks and, thus, ensuring data security is an important concern in some situations. In [26], researchers introduced a security model based on message digests, encryption and decryption technology to access remote data securely. Researchers implemented this security model in mobile-agent architecture with large number of remote data processing tasks. The experimental results show that the proposed model is secure and feasible for the mobile-agent architecture. This research did not explicitly examine the performance impact of

using encryption for delivery of data to mobile applications. Our work looks specifically at this impact.

In our research, we will focus on the *C/A/S* model and *C/I/S* model while investigating the impact of different technologies such as caching, encryption, and compression. Our main goal is to analyze the effect of data transmission from remote enterprise database using caching, encryption, and compression techniques.

Chapter 3

Software Architecture for Mobile Application

3.1 Introduction

Mobile applications are normally structured as a multi-layered application consisting of UI, business, and data layers. When developing a mobile application, developer may choose to develop a thin Web-based client or a rich client. In the case of building a rich client application, the business and data services layers are likely to be located on the device itself. On the other hand, the business and data layers will be located on the server to build a thin client. Figure 3-1 illustrates common rich client mobile application architecture with components grouped by areas of concern [30].

In our research, we focus on the client-agent-server (*C/A/S*) and client-intercept-server (*C/I/S*) models because we are retrieving data from remote enterprise databases or data sources and these two models seems to be the best fit for those tasks based on previous research. Moreover, our review of middleware based on mobile browsers says that most of the mobile browser engines were developed using either *C/A/S* or *C/I/S* model to retrieve from websites. In this chapter we will describe different software models proposed in previous researches.

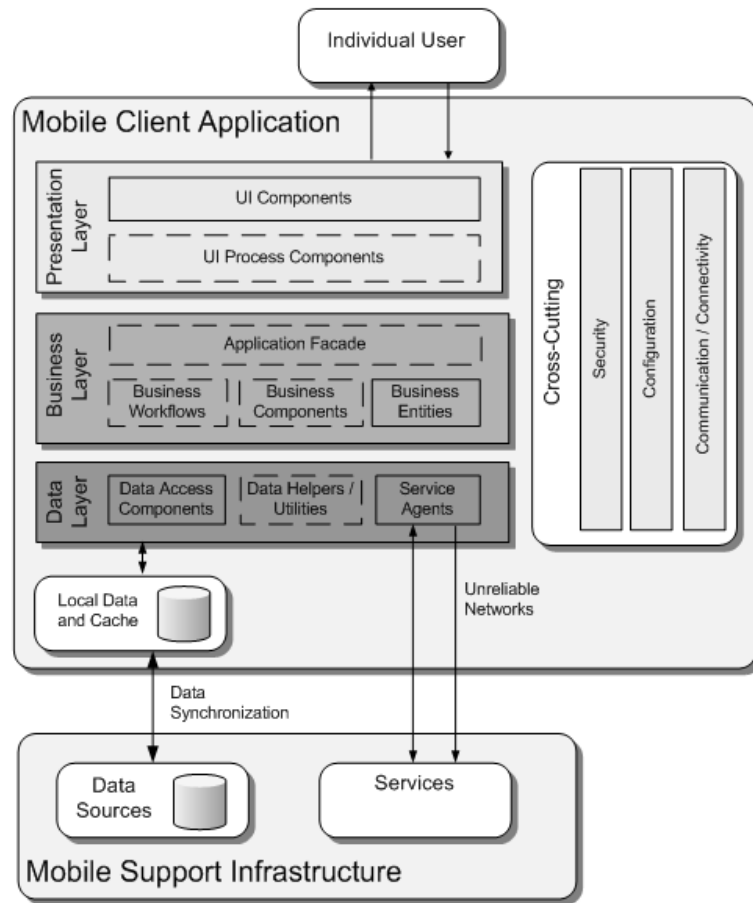


Figure 3-1: Common rich client mobile application architecture

3.2 Client-Server (C/S) Model

In the client-server model, the server component provides a function or service to one or many clients, which initiate requests for such services. The C/S model requires an application or browser to be located on the mobile client and communicate directly with the web server or database server via wireless communications. At the time of accessing a specific database, the client downloads the appropriate database driver to the mobile phone and then a connection establish between client and the database server.

In case of mobile application, the limitations of traditional C/S application are as follows:

- The main limitation of traditional C/S model is that the model suffers for performance due to download and initiation of the database driver (ranges

between 300-500 KB) on the client machine every time it connects to the database; that also wastes bandwidth.

- There is no way to optimize the data before the transmission to the mobile client, so receiving large data may not work or even if it works, it will take long time, which will affect the quality of the application data. So, a heavy loaded agent application may not work and regular applications will experience performance problems.
- Due to no data optimization before data transmission over network, data security cannot be provided to the wireless network. So, traditional C/S model based application undergoes security issue.

3.3 Client-Agent-Server (C/A/S) Model

The C/A/S architecture is a popular extension of the C/S model, containing three-tier architecture. Here, any communication goes through the mobile agent. At one extreme, agent acts as a mobile host. At another extreme, the agent is attached to a remote database or data source. Any client's request and server's response associated with this application is communicated through this service-specific agent. In this scenario, a mobile host must be associated with as many agents as the services it needs access to. Agents split the interaction between mobile clients and fixed servers into two parts, one between the client and the agent, and one between the agent and the server.

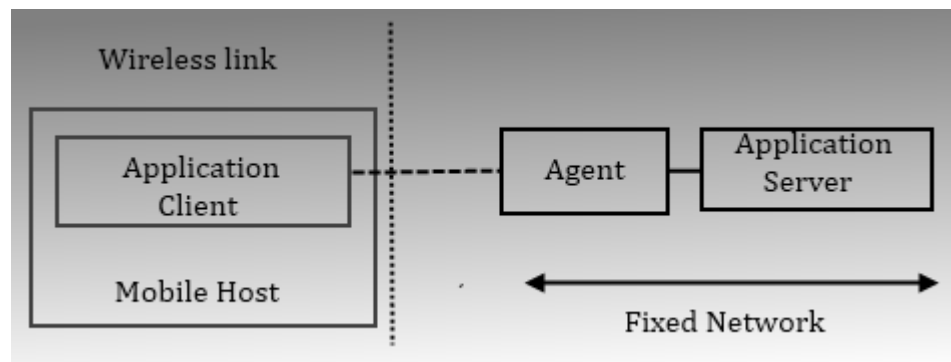


Figure 3-2: The Client-Agent-Server (C/A/S) model

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

- Solves the problem of initializing database driver for every query in C/S model.
- This model alleviates some of the impact of the limited bandwidth and poor reliability of wireless links by constantly maintaining the client's presence on the network via the agents.
- The agent splits the interaction between the mobile client and fixed servers into two parts, one between the client and the agent and one between the agent and the server.
- Data transmission can be optimized in the middleware so the QoS of data transmission improves with lower cost computation in the middleware or agent.
- A security wrapper in the middleware can provide data security over the wireless network.

Though the client-agent-server model offers number of advantages, it fails to sustain the current computation at the mobile client during periods of disconnection. In addition, the agent can directly optimize only data transmission over the wireless link from the fixed network to the mobile client but not in the opposite direction.

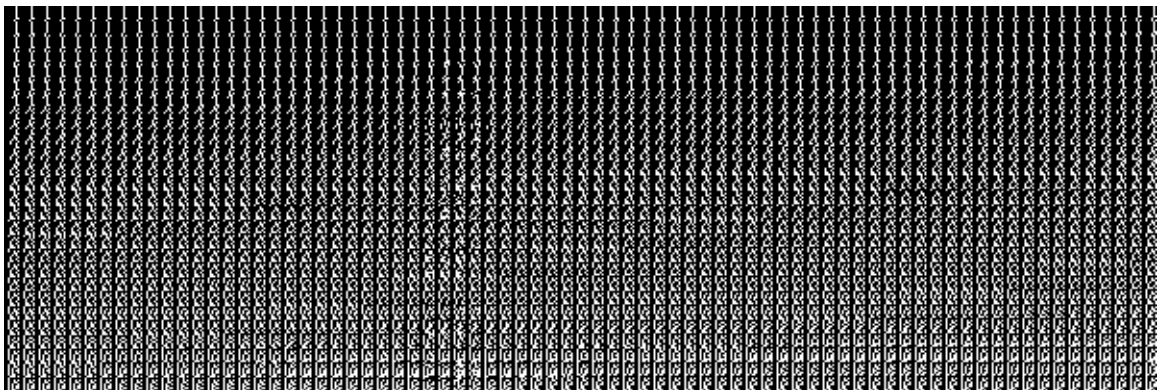


Figure 3-3: Client-Agent-Server (C/A/S) model with remote database

In our research, we focus on an enterprise database, which is located at a remote server and connected to the middleware through the Internet.

3.4 Client-Intercept-Server (C/I/S) Model

The C/I/S model proposes the deployment of an agent that will run at the mobile device along with an agent that will run in the server side or middleware. This client-side agent intercepts the client's requests and together with the server-side agent performs optimizations to reduce data transmission over the wireless link, improve data availability and sustain the mobile computation uninterrupted. From the point of view of the client, the client-side agent appears as the local server proxy that is co-resident with the client. Since the pair of agents is virtually inserted in the data path between the client and the server, the model is also called C/I/S instead of C/A/S.

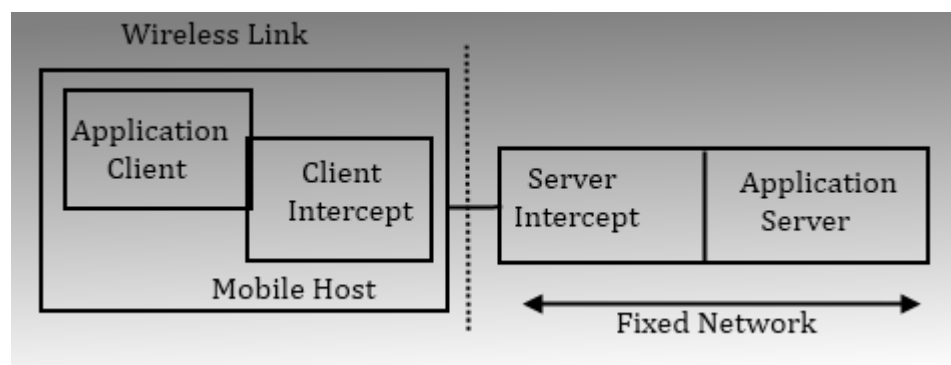


Figure 3-4: The Client-Intercept-Server (C/I/S) model

This model provides separation of responsibility between the client-side and server-side agents which facilitate highly effective data reduction and protocol optimization. In case of database applications this model consists a client-side database agent which is specific to the agent and serve only one agent; the server side database agent will serve many agents at a time. The agent pair cooperates to intercept and control communication over the wireless link for reducing network traffic and query processing. In our research, we also investigate the C/I/S model.

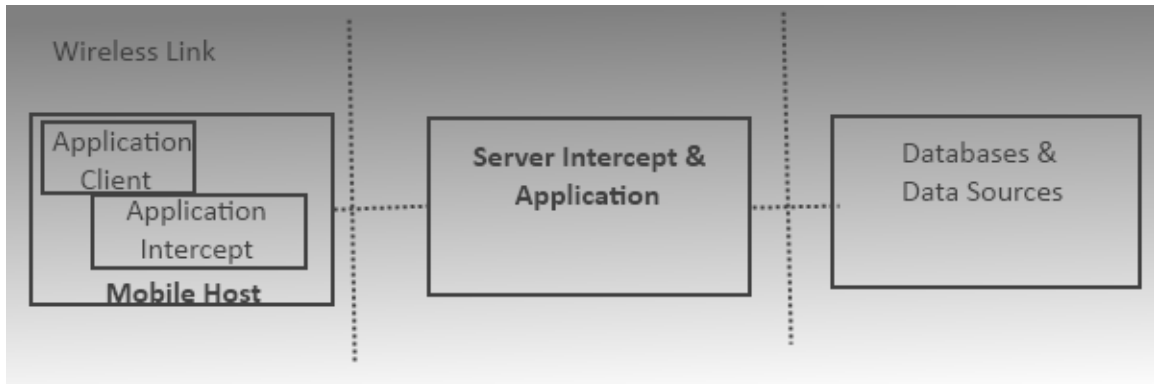


Figure 3-5: Client-Intercept-Server (C/I/S) model with remote database

3.5 Peer-to-Peer Model

In the peer-to-peer model, the server resides at the mobile host. In this case, mobile hosts are equal partners in distributed computations. This network has emerged as an efficient system, being typically used for sharing files containing audio, video, data or any digital-format files and distributing services over fixed networks.

3.6 Mobile-Agent Model

In the mobile-agent model [31], [32] an agent first lands at an object server and then is executed to manipulate objects in the object server. If the agent finishes manipulating the objects in the object server, the agent moves to another server which has data to be manipulated. Here, agents manipulate objects only in local object servers without exchanging messages in a network. In addition, an agent negotiates with the agent if some agents manipulate objects in a conflicting manner. Through the negotiation, each agent autonomously makes a decision on whether the agent continues to hold the objects or gives up to hold the objects. After manipulating all or some of the servers, an agent makes a decision on commit or abort. Here, object servers may suffer from crash faults.

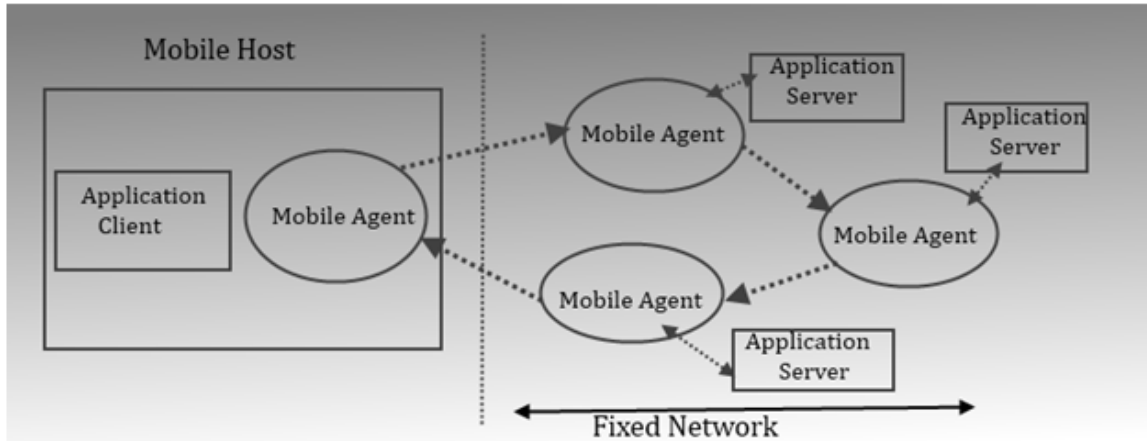


Figure 3-6: The Mobile Agent model

3.7 API

In this research we experiment with C/A/S and C/I/S; as we are retrieving data from remote data source or database, these two models are supposed to behave better than other models proposed by researches over the last two decades.

To experiment with different models and technologies for data retrieval, we developed an API for the middleware. There are several mobile phones providers available in the market with their own operating system; and every operating system provides a different SDK to develop applications. Mobile application developers are facing problems due to great heterogeneity of these devices. A common API with the flexibility to support caching, compression, and encryption will make development of mobile application easier and faster. We also developed an Agent API for C/I/S model which will compute the caching data, decompression and decryption and the mobile application end.

The two APIs developed, then, are: **1) The Middleware API** which provides a web service which return data in XML format, so any mobile application can retrieve the data and use it for the application. The API has the flexibility to retrieve data from databases through middleware and make use of caching, compression, encryption or any combination of these technologies. **2) The Mobile Agent API** which has been developed using J2ME which provides functionalities for retrieving cache data, decompressing data,

and decrypting data coming from the middleware API. Mobile application developers can use this API for their J2ME or Java based mobile application.

The middleware API is a universal API providing web services and returning standard web service data so any mobile developer can use this to retrieve data from remote databases. Also, the mobile API will help to process the remote data for the mobile application, especially developed in J2ME or Java.

Chapter 4

Implementation

4.1 Introduction

To evaluate the different models and impact of the different techniques on data transmission for mobile applications in a wireless network, we developed the following:

- Two Remote databases
- Middleware API
- Mobile Agent API and Mobile application for two different domains.

In the data transmission lifecycle, the mobile application sends a request to the middleware; the middleware retrieves data from the cache or executes the query on the remote database and processes the data; after processing the middleware returns data to the mobile application. Finally, the mobile application processes the data returned from the middleware and displays it. In Figure 4-1 we present a high level overview of the data transmission lifecycle.

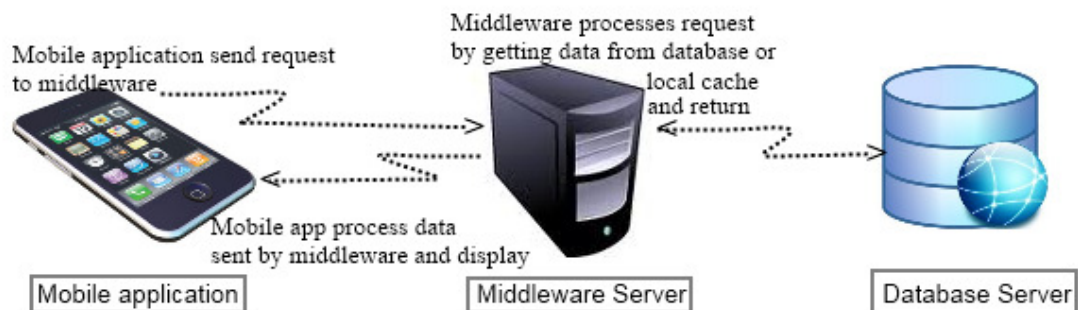


Figure 4-1: High level overview of the system

We experiment on two mobile application architectures: the C/A/S and C/I/S models. In the C/A/S model, the mobile application sends requests to the middleware and the middleware processes the data based on the requests from the mobile application. If the mobile application requests cache data, the middleware searches for the data in the cache. If the data is found, the middleware returns it back to the mobile application. If the data is not in the cache, the middleware retrieves the data from the database, caches it and sends it back to the mobile client. The middleware also compresses and/or encrypts data based on the request from the mobile client. After getting the data from the middleware, the mobile application decompresses and/or decrypts data if necessary, processes the XML formatted data from the middleware, and then displays it on the mobile screen. In Figure 4-2, we illustrate the request and response cycle for the C/A/S model.

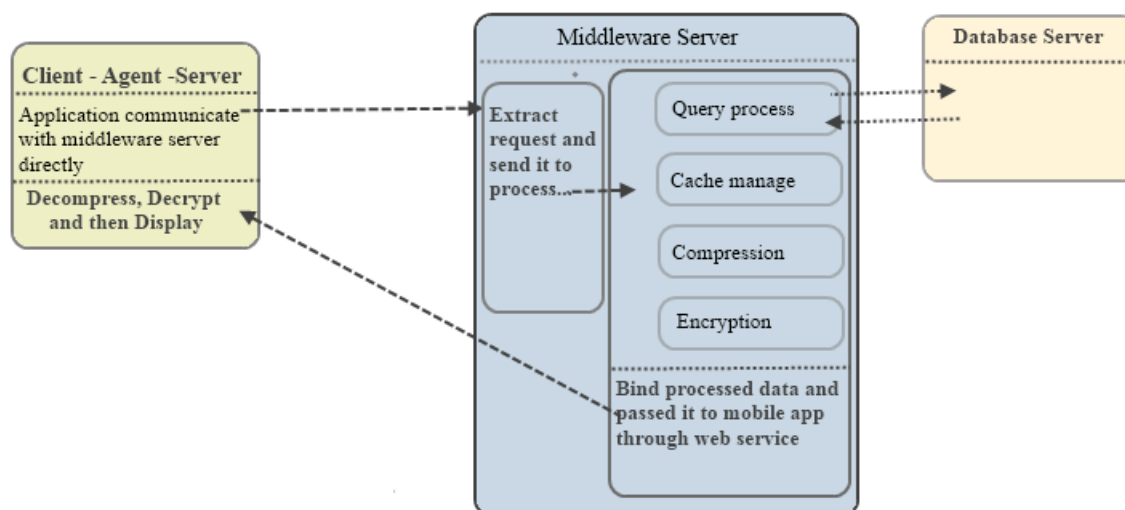


Figure 4-2: Overview of Client-Agent-Server request and response

In case of C/I/S, the middleware and the database are same as in the C/A/S model. However, the mobile application does not directly communicate with the middleware or processes returning data from the middleware; rather, it calls the Intercept. Instead of calling web service, processing returned data from the web service (decompress, decrypt, or extract from XML), the mobile application calls methods of mobile API which compute these functions and return data to the application to display it. In Figure 4-3, we illustrate the request and response interaction of the C/I/S model.

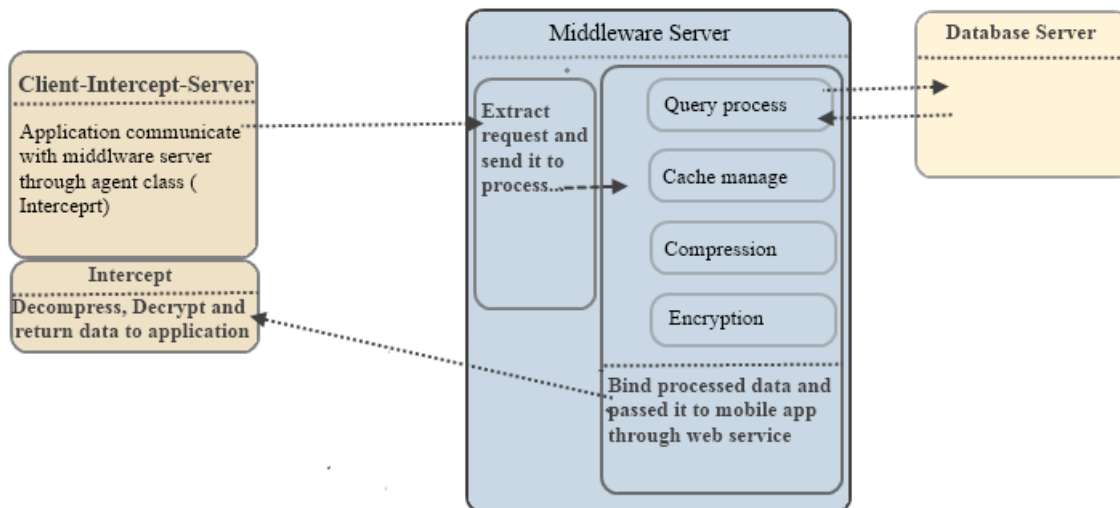


Figure 4-3: Overview of Client-Intercept-Server request and response

4.2 Databases

We experimented with two different databases. One database contains medical information of around 10,000 patients (all fictitious) and another database is the common test database named AdventureWorks [42], which contains inventory information of different products.

The medical database contains several tables of hospital information (hospital), patient information (patient), patient in hospital information (patientinhospital), and the diagnosis results of patients (patient_data). The database contains approximately ten thousand patient's diagnosis result in the patient_data table. Based on the patient data in the in the patient_data table we can do different experiments.

The adventure works database contains many tables for a complete inventory system. In our experiments we use product, product category, product subcategory, product description, and product photo table; these all together create a small inventory system for use with a mobile application.

4.3 Middleware and Middleware API

The middleware retrieves data from the remote databases and returns it back to the mobile application. The middleware is developed using C# programming language in an ASP.NET platform. The pseudo code to retrieve data from a remote data source is presented in Figure 4-4.

Function Get Data From Remote Data Source
Input: Database Information, Query String
Output: Data from database
Query ← Encrypt Query String for Security
HttpRequest to send the request query to the database
HttpResponse to retrieve response result of the query from database
StreamReader to read the return data
Return String Format of StreamReader data

Figure 4-4: Pseudo code to retrieve data from remote data sources

We experiment with caching, compression and encryption within the middleware to understand their impact on data transmission in a wireless network. The mobile software developer can use the API to specify whether data should be cached or compressed or encrypted or any combination of these techniques by the middleware, just by calling the web services provided by the API. The API provides a generic interface to a middleware platform and the mobile application developers can customize the use of these techniques to meet the need of their application to retrieve data.

4.3.1 Caching

In most web based application or wireless applications, caching significantly improves the performance of applications. Instead of repeatedly fetching the same data, caching can be used to store data in a temporary memory. Every time the application fetches data, the data can come from the cache instead of recalculating it or fetching it from a remote location.

In our middleware, we use the ASP.NET application cache to cache data. The application cache provides a programmatic way to store arbitrary data in memory using key/value

pairs. Using the application cache is like using application state. However, unlike application state, the data in the application cache is volatile. This means it is not stored in memory for the life of the application. The advantage of using the application cache is that ASP.NET manages the cache and removes items when they expire or become invalidated, or when memory runs low.

Using a cache key the application cache determines whether an item exists in the cache or not, and if it does, to use it. If the item does not exist, the application automatically recreates the item and then places it back in the cache. The pattern of key/value pairs ensures that the cache contains latest data from data source.

To retrieve data faster we pre-fetch data in the middleware. Every time the application creates cache data we keep a record of the cache key and cache query. After a certain time, the application automatically checks the data and pre-fetches it in files if the data changes. We use the ASP.NET custom cache dependency feature to validate and expire cache data. Custom cache dependency represents a logical dependency between a cached item and a file(s), folder(s), or another cached item(s). When the dependency changes (i.e., the file/folder/item changes), the bound cached item is removed from the cache so the cache expires. In our middleware we create files based on cache dependency, every time a new cache item is created, the system creates a file with the same name as the cache key. At the time of pre-fetching, the system updates these cache dependency files to keep the cache data update. The pseudo-code of the middleware caching and pre-fetching process is presented in Figure 4-5.

Function Get Cached Data
Input: Query to execute, Cache Key
Output: Cached data from memory or retrieve data from database and cache for future
IF Cache Key is already available
Return Cache Data based on cache key
ELSE
Data ← Get Posted Data from Database
Create cache dependency file for cache validation
Save Cache Key and Query for pre-fetching
Return Data

Figure 4-5: Middleware pseudo-code to retrieve or create cache data

This algorithm can be extended by using any other suitable caching technique instead of using the built-in ASP.NET application caching. In the future, different caching techniques could be implemented in order to analyze their performance for data transmission for mobile applications.

4.3.2 Compression

Data compression is a common technology to represent information in a compact format; data compression involves encoding information using fewer bits than the original representation. Compression is useful because it helps reduce the consumption of resources such as data space or transmission capacity. As the bandwidth of wireless network is scarce, it may be advantageous to compress data to get the maximum out of the bandwidth. Mobile applications suffer from limited memory where data compression may help save memory and at the same time improve the speed of data transfer.

Mainly there are two types of compression techniques are available: Lossless Compression and Lossy Compression. Lossless compression is mainly used for spreadsheets, text and executable program compression; on the other hand Lossy compression is mainly used for image, video, and audio compression. Lossless data compression is used in many applications such as the ZIP file format and in the UNIX tool gzip. Lossless compression is used in cases where it is important that the original and the decompressed data be identical, or where deviations from the original data could be deleterious. Typical examples are executable programs, text documents and source code.

In the middleware, we used gzip compression; it is based on the DEFLATE algorithm, which is a combination of Lempel-Ziv (LZ77) and Huffman coding. Gzip contains:

- 10-byte header, containing a magic number, a version number and a time stamp
- optional extra headers, such as the original file name
- a body, containing a DEFLATE-compressed payload

- 8-byte footer, containing a CRC-32 checksum and the length of the original uncompressed data.

To implement gzip compression in the middleware, we used SharpZipLib [33], an ASP.NET compression library that supports Zip files using both stored and deflate compression methods. The pseudo-code for the middleware data compression is presented in Figure 4-6.

Function Get Compressed Data
Input: Data from database or cache
Output: Compressed data of the original data
Data ← Get Data from Database of Cache
CompressData ← Compress Data in Gzip format using SharpZipLib library
CompressData ← Encode the CompressData
Return CompressData

Figure 4-6: Middleware pseudo-code to compress retrieved data

Instead of SharpZipLib, we could use any other compression algorithm in the middleware just by modifying the algorithm.

4.3.3 Encryption

Encryption is the process of transforming information using an algorithm to make it unreadable to anyone except those possessing special knowledge, usually referred to as a key. The result of the process is encrypted information. The reverse process (to make the encrypted information readable again) is referred to as decryption.

Encryption is mainly used to protect data transferred via networks, mobile telephones, wireless microphones, wireless intercom systems, Bluetooth devices and bank automatic teller machines. Encrypting data in transit helps to secure it as it is often difficult to physically secure all access to networks.

In the case of enterprise databases, it is often necessary to move data over networks – to other sites or to remote desktop and mobile applications. Data transmission across networks, particularly public networks, creates potential security problems [34]. Given

the importance of data security, we have implemented encryption in the middleware so data transmission in the wireless network can become secured.

In our research work, we encrypt the data to Base64 data format before transmitting it to wireless network. Instead of using a real encryption algorithm for our experiments, we used Base64 encoding to emulate encryption processing; in reality this encoding scheme is much less complex than a real encryption scheme. Base64 is a group of encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation. Base64 encoding schemes are commonly used when there is a need to encode binary data that needs be stored and transferred over media that are designed to deal with textual data. This is to ensure that the data remains intact without modification during transport. Base64 is commonly used in a number of applications including email via MIME, and storing complex data in XML. The pseudo-code for encryption in the middleware is presented in Figure 4-7.

Function Get Encrypted Data
Input: Data from Database or Cache
Output: Encrypted data
Data ← Data from Database or Cache
EncryptedData ← Encrypt the Data using ASP.NET text encoding library
Return EncryptedData

Figure 4-7: Middleware pseudo-code for encrypting retrieved data

4.3.4 Overview of the Middleware

The middleware API is called by the mobile application through an XML based web service. The web service call is a method named GetServiceData with parameters specifying the technologies to be used to retrieve data.

The parameters are:

1. Database or data source;
2. Query to execute in the database;

3. Caching information;
4. Compression Information;
5. Encryption Information;

The service method retrieves data from the database using the technologies as specified in the parameters of the method and returns it to the mobile application. The pseudo-code of this method is depicted in Figure 4-8.

Function Get Service Data
Input: Data source, QueryString, IsCaching, IsCompression, IsEncryption Output: Data from database with optimization Query \leftarrow QueryString IF IsCaching \neq FALSE Data \leftarrow GetCachedData ELSE Data \leftarrow GetDataFromDatabase IF IsCompression Equals TRUE Data \leftarrow GetCompressedData of the CacheData or Data from Database IF IsEncryption Equals TRUE Data \leftarrow GetEncryptedData of the CachedData or Data from Database and/or ComrepsedData

Figure 4-8: Pseudo-code of service to retrieve data in middleware

4.4 Mobile Agent API

With the growing need to access remote web services from mobile applications, most of the mobile SDKs provide a remote web services library. Even if any SDK does not provide such a library, the developer can use third party open source software such as kSOAP [35]. In our experiments, we use the J2ME Web Services API (WSA) [36]. The API's two optional packages standardize two areas of functionality that are crucial to clients of web services: remote service invocation and XML parsing.

To retrieve data from the web service data we used kXML [37], a lightweight Java-based XML parser designed to run on limited, embedded systems such as personal mobile devices. It is a pull parser which means it reads a little bit of the document at once. Then

the application drives the parser through the document by repeatedly requesting the next piece [38].

If the web service data is compressed, we use Jazilib [39]; a java based open source compression decompression library, it supports compression and decompression for J2ME application too. Jazilib supports encryption and decryption for different data formats such as ZIP, GZip, and Deflector. In our experiments, the middleware provide data in GZip compressed format so we used GZIPInputStream of Jazilib to decompress the middleware data.

If the web service data is encrypted, we use Base64 decryption provided by bouncycastle [40]. Bouncy Castle provides a collection of cryptography techniques for both the Java and the C# programming languages. They also provide a cryptography library for Android. So, all Java based mobile applications (J2ME, Blackberry java application, Android java application) can use this library to encrypt/decrypt data in different format.

4.4.1 Implementation of C/A/S

In the case of the C/A/S model, our mobile application is implemented using J2ME. The application itself contains all the functionalities, such as remote data fetching, either from cached or remote database, decompression of the data, if the data is returned by the middleware server in compressed format and decryption of the data if the data is returned by the middleware in an encrypted format. In the case of the C/A/S architecture, the mobile application directly communicates with the middleware agent to retrieve data, process the retrieve data, and display it to the mobile screen. Here all the computing in the mobile application is in one class file.

The list of the J2ME libraries and third party libraries used to develop the mobile C/A/S application are summarized in Figure 4-9.

```

javax.microedition.midlet.*;
javax.microedition.lcdui.*;
org.netbeans.microedition.lcdui.SplashScreen;
java.util.Vector;
liveservices.Services_Stub to retrieve data from middleware web service API
org.kxml2.io and org.xmlpull.v1 to read XML formatted web service data
net.sf.jazzlib.GZIPInputStream to decompress data
base64.decode to decrypt data

```

Figure 4-9: J2ME libraries and third party libraries for C/A/S application

In our experiments we display data in J2ME controls. The mobile application calls a web service on command action, and loads the processed data into control; the pseudo-code of the command action is presented in Figure 4-10.

```

SERVICE ← Initiate web service stub.
DATABASE ← Database connection information or path
SQL ← Query to retrieve data from database
CACHE ← 'false' if not cache data 'cache_key' if data from cache
COMPRESS ← 'true' if compressed, 'false' or 'blank' if not compressed
ENCRYPT ← 'true' if encrypted, 'false' or 'blank' if not encrypted
LIST_ITEM ← Name of the list item to load data on mobile screen
CALL FUNCTION GENERATEVIEW with parameters
    SERVICE, DATABASE, CACHE, COMPRESS, ENCRYPT, LIST_ITEM

```

Figure 4-10: Command actions to generate mobile data

The mobile application does the following things to generate the list view from the web service data:

1. Retrieve data from the web service by calling the web service method.
2. Check if data is encrypted, if encrypted, then it uses the Base64.Decode() method to decrypt the data.
3. Check if the data is compressed, if compressed, then it uses GZIPInputStream to decompress data.
4. Convert the data to byte array and pass it to KXML to extract data from the XML format as well as append it in List view control to display.

Figure 4-11 provides an overview of the GENERATEVIEW function, which actually retrieves data from web service, processes the data and displays it on the mobile screen.

```

DATA ← SERVICE.GETDATA ( DATABASE, SQL, CACHE, COMPRESS,
ENCRYPT)
XMLBYTEARRAY ← Initialize byte array for temporary data store
IF ENCRYPT Equals TRUE
    XMLBYTEARRAY ← Base64.Decode(DATA)
    FLAG ← TRUE
END IF
IF COMPRESS Equals TRUE
    FLAG ← TRUE
    DATABYTEARRAY ← Initialize a byte array for temporary data store
    IF XMLBYTEARRAY Equals NULL
        DATABYTEARRAY ← Base64.Decode(DATA)
    ELSE
        DATABYTEARRAY ← Base64.Decode(XMLBYTEARRAY)
    END IF
    XMLBYTEARRAY ← Get Decompress data using GZIPInputStream
END IF
IF FLAG Equals False // IF data is not compressed or encrypted
    XMLBYTEARRAY ← DATA.GetBytes()
END IF
Read XMLBYTEARRAY using KXML and append data in LIST_ITEM

```

Figure 4-11: Data process and display in Client-Agent-Server

4.4.2 Implementation of C/I/S

In the case of C/I/S, we developed an Intercept API; the name of the API package is *Intercept*. The *Intercept* API contains methods to retrieve and read XML formatted web service data, do decompression and decryption. Here, the mobile application calls the web service, retrieves web service data and passes the data to the *Intercept* API; the *Intercept* API processes the data, saves it in temporary storage, and returns it back to the mobile application to display it.

The source code for the C/I/S is pretty simple; the pseudo-code of the mobile application following C/I/S model is presented in Figure 4-12.

```

SERVICE ← Initiate Middleware Service and Call Service Method
DABASE ← Database connection information or path
SQL ← Query to retrieve data from database
CACHE ← 'false' if not cache data 'cache_key' if data from cache
COMPRESS ← 'true' if compressed, 'false' or 'blank' if not compressed
ENCRYPT ← 'true' if encrypted, 'false' or 'blank' if not encrypted
DATA ← (DATABASE, SQL, CACHE, COMPRESS, ENCRYPT)
RESULT ← CALL INTERCEPT API ( DATA, CACHE, COMPRESS, ENCRYPT)
DISPLAY RESULT in the mobile screen.

```

Figure 4-12: Client-Intercept-Server mobile application

In the case of C/I/S, the mobile application itself becomes smaller because all the computation has been done by the Intercept API, the mobile application just needs to call the API and display the returned result in any display control structure. This *Intercept* API can be used for any mobile application developed in Java; it makes the mobile application development simple and faster. The pseudo-code of the *Intercept* API is in Figure 4-13.

```

USE org.kxml2.io and org.xmlpull.v1 to read XML data
USE net.sf.jazzlib.GZIPInputStream library to decompress data
USE base64.Decode to decrypt data
FUNCTION GET_DATA_TO_DISPLAY ( DATA, CACHE, COMPRESS, ENCRYPT)
XMLBYTEARRAY ← Initialize byte array for temporary data store
IF ENCRYPT Equals TRUE
    XMLBYTEARRAY ← Base64.Decode(DATA) ; FLAG ← TRUE
END IF
IF COMPRESS Equals TRUE
    FLAG ← TRUE
    DATABYTEARRAY ← Initialize a byte array for temporary data store
    IF XMLBYTEARRAY Equals NULL
        DATABYTEARRAY ← Base64.Decode(DATA)
    ELSE DATABYTEARRAY ← Base64.Decode(XMLBYTEARRAY)
    END IF
    XMLBYTEARRAY ← Get Decompress data using GZIPInputStream
END IF
IF FLAG Equals False // IF data is not compressed or encrypted
    XMLBYTEARRAY ← DATA.GETBYTES()
END IF
RESULT ← Read XMLBYTEARRAY using KXML and Generate a Vector
RETURN RESULT

```

Figure 4-13: Intercept API

4.5 Summary

In this Chapter we reviewed our system implementation. We developed a generic middleware API and Java-based mobile application development API. These APIs meet our requirements to enable us to evaluate different mobile application architectures and technologies. These APIs can also be extended and used for further experimentation with other mobile architectures and technologies. These APIs can be used by the mobile application developers to make development simple, easier and faster.

Chapter 5

Experiments and Results

Our research focused on two extended Client/Server architectures: the C/A/S model and the C/I/S model. Our goal was to analyze data transmission for these two mobile software architectures in different scenarios of a mobile application with some common but effective technologies integrated into the middleware of the system; such as middleware data caching, data compression, and data encryption.

We experimented with data transmission in two different domains: An inventory system and a medical information system. We first developed a mobile application for the inventory system for use with small scale data analysis. Analysis involving larger data sets was then done using the medical information system. Our experimental environment consisted of the following components:

1. Two remote database server;
2. Middleware server (providing web service);
3. Mobile application (C/A/S application and C/I/S application).

In Table 5-1 we summarize the tools we used for the experimental systems.

Component	Development Tool
Remote database Server 1	MySQL Database
Remote Database Server 2	MySQL Database
Middleware API	ASP.NET web service

Mobile Application C/A/S	J2ME
Mobile Application C/I/S	J2ME

Table 5-1: List of development tools for the experimental environment

5.1 Experimental Setup

The following provides an overview of the experimental environment in more detail.

Database Server: The databases are hosted on a remote server (www.godaddy.com), accessible through a public IP address; it has a Linux operating system and runs MySQL server, version 5.0.

Middleware Server: The middleware was developed using ASP.NET web service [41] and the C# as the programming language for the middleware. The middleware server runs Internet Information Services (IIS) 7 server. The configuration of the server is: Windows 7 32-bit operating system, Intel core duo 2.3 GHz processor and 4 GB memory. It is accessible through the Internet through a public IP address.

Mobile Phone: To experiment with mobile applications, we used a Nokia E72 smart phone; configuration of the mobile phone is: Symbian OS 9.3, 600 MHz CPU, 250 MB Internal memory, and 128 MB RAM.

In Table 5-2 we outline the communication details between the mobile application, the middleware server and the remote databases.

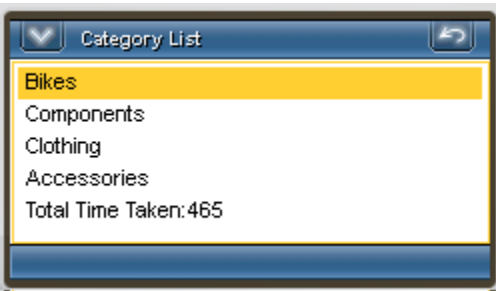
Mobile Application	Middleware	Remote Database
<p>The mobile application connects to the middleware services through wireless (WiFi).</p> <p>The mobile application is installed on a mobile phone so application can be run</p>	<p>The middleware is located in a laboratory in the Computer Science Department at the University of Western Ontario; this server has a public IP.</p>	<p>The remote database is hosted on a server at the godaddy.com hosting service. It has public IP.</p> <p>Middleware communicate with the remote database through the IP address.</p>

and accessed from anywhere. Testing was done from a home residence connection which had a 2Mbps dedicated wireless connection.	The mobile application communicates with the middleware through the middleware API using the public IP of the middleware server.	Communication to this server was from the middleware server through the Internet.
---	--	---

Table 5-2: Communications between the mobile application, middleware server, and remote database server.

5.2 Experiments with Inventory System

We developed two mobile applications for accessing information from the inventory system following the C/A/S and C/I/S models. The inventory system applications use the product information tables of the AdventureWorks database. The application displays all the product categories from the category table, and then enables the user to select a category. Selecting the category causes the application to display the subcategories from and then selecting a subcategory produces a list of the products under the category and subcategory. When the user selects any of the products from the product list, the application retrieves and displays all the information available in the database for the product along with the product image. In Appendix E, we present the query involved in each step of the application. The steps of the inventory system mobile application are presented in Table 5-3.

Step 1: Retrieve product categories from the remote database through the middleware service. The user scrolls to a desired product category and selects it (“Bikes”).	
---	--

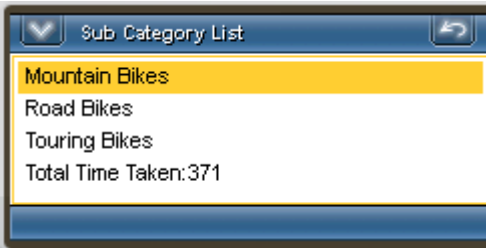
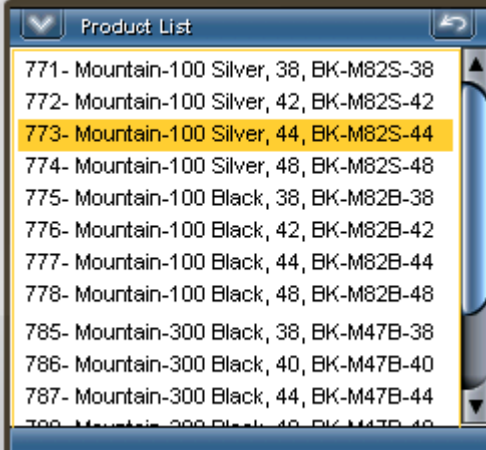

<p>Step 2: Select a product from the product category list (“Bikes”); the application retrieves the subcategories of the selected category. Selecting a subcategory (“Mountain Bikes”) retrieves the products.</p>	
<p>Step 3: Once a subcategory (“Mountain Bikes”) has been selected, the application retrieves the product list.</p>	
<p>Step 4: Selecting a specific product from the product list (see Step 3 for selected product), the application retrieves and displays information on the product and an image of the product from database.</p>	

Table 5-3: Steps of the inventory system mobile application

Our initial experiments involved two different scenarios for both of the software architectures. At every step in the application (as in Table 5-2) we measure the execution time from the time the request is sent until the information is displayed on the mobile device. We compare the two models with two different scenarios using the mobile

inventory application; the scenarios are summarized in Table 5-3. The results of the experiments are presented in Table 5-4; each experiment was repeated 3 times.

Scenario #	Step 1 (Select Category)	Step 2 (Select Sub Category)	Step 3 (Select Product)	Step 4 (Product details)
1	Bikes	Mountain Bike	Product 773	Product Details
2	Accessories	Bike Racks	Product 876	Product Details

Table 5-4: Steps in different scenarios of inventory system

	C/A/S (SD)	C/I/S (SD)	Data Size
Scenario 1	1662.00 ms (13.00)	1696.33 ms (9.82)	~7 KB
Scenario 2	1614.33 ms (12.66)	1643.00 ms (21.28)	~ 2.5 KB

Table 5-5: Inventory system experiment result for scenario 1 and scenario 2

In this experiment, Scenario 1 (~7 KB) loads with more data than Scenario 2 (~2.5 KB). The experiment result shows that in case of both of the architectures, data transmission time for Scenario 1 is larger than data transmission time of Scenario 2; standard deviations are in parentheses. The result shows that the C/A/S performs slightly better than the C/I/S, but given the size of the standard deviations, the relatively small amounts of data in both scenarios, there is really little difference in performance of the mobile software architectures.


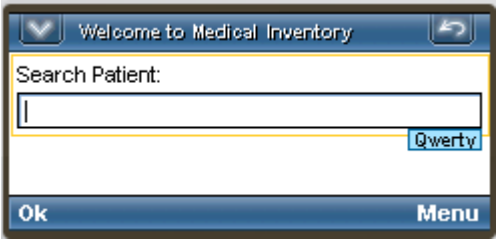
With the inventory look-up application, we were able to test the APIs and make some initial comparisons between the software models. We were able to evaluate the application with modest amounts of data. For this simple look-up application, both models functioned well.

5.3 Experiments with a Medical Information System

While the inventory application was useful, the data available was relatively limited. Given that there was more data available in the medical information system; we chose to carry out more extensive analysis with that application. In particular, we used it to assess

the performance impact of caching, compression, and encryption as well as decompression and decryption in the mobile application for both the C/A/S and C/I/S architectures. The medical application also was more “natural” in retrieving more data.

In the Medical Information Application, a user (e.g., doctor, nurse, etc.) can login and search patients by their last name or first name or any part of the name. The search result returns a list of patients along with their unique identity number, name, age, and sex. Selecting a patient from the list returned, will cause the application to search the database and return the medical history and previous diagnosis results of the patient. In Appendix F, we summarize the queries used for each step of the application. The steps of the Medical Information System are illustrated Table 5-6.

<p>Step 1: User logs in using their user name and password; saved in the database along with permissions to check a patient’s medical information.</p>	
<p>Step 2: After successful login, the user can search for a patient by first name, last name, or any part of the name.</p> <p>Example: ‘Henry’, ‘John’, or ‘Jo’</p>	

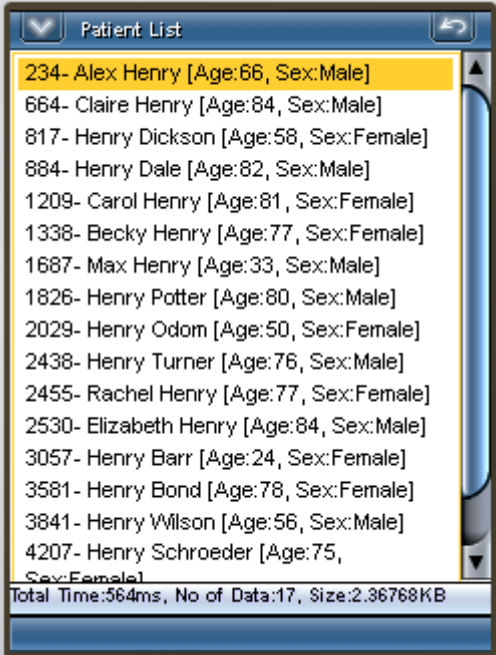
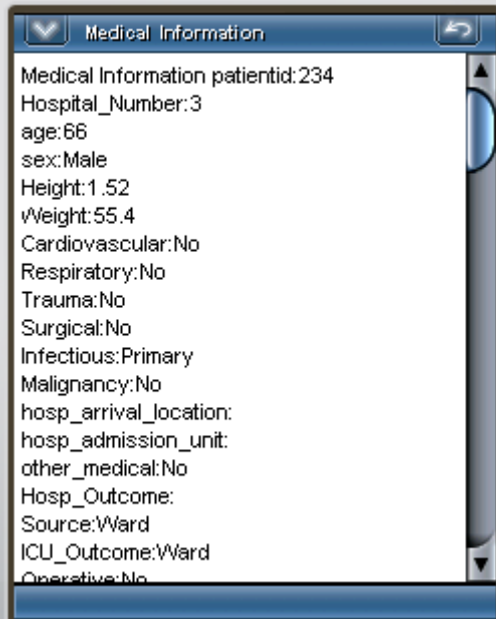
<p>Step 3: Based on the search key, data is retrieved from the remote database; basic information is displayed: Patient Identification Number, Name, Age and sex.</p> <p>At the bottom of the screen, the total time taken in milliseconds to retrieve data from database and display it is displayed, along with total number of records found and the size of the returned data in bytes.</p>	
<p>Step 4: Selecting any of the patients will display that patient's medical and diagnosis history.</p>	

Table 5-6: Medical Information System step by step

Using the Medical Information Application, we carried out a number of experiments:

- **Experiment 1: Basic experiment comparing the two architectures for two scenarios.** The remaining experiments build on this experiment by using one or more additional capabilities (e.g. caching) of the middleware.
- **Experiment 2: Experiment with caching.**
- **Experiment 3: Experiment with compression.**
- **Experiment 4: Experiment with encryption**
- **Experiment 5: Experiment varying all aspects; 4-Factor analysis on software architecture, caching, compression, and encryption**

All the experiments have been executed with three replications.

5.3.1 Experiment 1: Basic Experiment

In Experiment 1, the mobile application requests, through the middleware, data from the remote database; the middleware retrieves the data and sends it to the mobile application through the web service in XML format. The mobile application extracts the retrieved data from the XML formatted data and display it on the mobile screen. We execute the experiment for three different scenarios involving different size data results. The scenarios with the data sizes are presented in Table 5-7.

Scenario 1	Search key for patient search is 'Henry'	Total patients found: 17, Data Size: ~2KB
Scenario 2	Search key for patient search is 'Li'	Total patients found: 397, Data Size: ~45KB
Scenario 3	Search key for patient Search 'T'	Total patients found: 1850, Data Size: ~230KB

Table 5-7: Experiment 1 scenarios and data sizes

We replicate each experiment three times. The experimental results and analyses are presented in Table 5-8. The application on our test mobile phone could not handle the data of the scenario 3 (returns around 230KB data from middleware) to display; we will use scenario 3 with different data transmission techniques in our experiments.

Scenarios	C/A/S (SD ¹)	C/I/S (SD)	Analysis
Scenario 1	525.67 ms (31.39)	577.67 ms (24.22)	C/A/S little faster than C/I/S but the difference is too small to negligible
Scenario 2	856.00 ms (24.64)	1073.00 ms (37.26)	C/A/S faster than C/I/S; difference of around 200ms, not a significant difference.

Table 5-8: Regular experiment result and analysis

The results of the analysis of variance (ANOVA), presented in Table 5-8, show that the percentage of variation explained by the factors were: Software Architectures (SA) - 9%, Scenarios (S) - 84% and the interaction of these (SA+S) was 3%. The error also explained about 3% of the variation. Clearly, the different scenarios had the greatest impact on the variation in results while the variation attributable to the different architecture was minor.

Software Architecture (SA)	Scenarios (S)	SA+S	%Error
8.94%	84.18%	3.36%	3.52%

Table 5-9: Results of the analysis of variance (ANOVA) for Experiment 1

Summary of Experiment 1: In case of regular data transmission through middleware, the data transmission time increases if the data size increases. For transmission of larger data sets, C/I/S performs better than C/A/S, but the difference in transmission time between C/A/S and C/I/S is small.

5.3.2 Experiment 2: Use of caching

In Experiment 2, the scenarios are same as Experiment 1 (Table 5-7); here, we cache data in the middleware to understand its impact on the scenarios for the software architectures (C/A/S and C/I/S model). For this experiment, the C/A/S model performs better than the C/I/S model but the difference is very small; scenario 1 is faster than scenario 2 because of the data size of scenario 2 is larger than scenario 1. Same as experiment 1, we execute

¹ SD = Standard Deviation

scenario 3, but again, we are unable to display data on the mobile device. The experiment with caching results and analysis are presented in Table 5-10.

Scenarios	C/A/S (SD)	C/I/S (SD)	Analysis
Scenario 1	470.67 ms (1.15)	492.67 ms (13.65)	C/A/S little faster than C/I/S but the different is negligible
Scenario 2	544.67 ms (15.05)	622.33 ms (20.28)	C/A/S faster than C/I/S but the different is small

Table 5-10: Experiment with caching result and analysis

We perform three factor analysis of variance (ANOVA) between software architectures, scenarios, and caching using the data from Experiments 1 (without caching) and 2 (with caching); the experimental results are summarized in Table 5-10.

Software Architecture	Scenario (S) ²	Caching (Ca) ³	Result	SD
C/A/S	S1	No Ca	525.67 ms	31.39
C/A/S	S1	Ca	470.00 ms	1.55
C/A/S	S2	No Ca	1073.00 ms	37.27
C/A/S	S2	Ca	544.67 ms	15.05
C/I/S	S1	No Ca	577.67 ms	24.22
C/I/S	S1	Ca	492.67 ms	13.65
C/I/S	S2	No Ca	1073.00 ms	32.27
C/I/S	S2	Ca	622.33 ms	20.21

Table 5-11: Three-factor analysis on software model, scenario, and caching

The results of the ANOVA shows that the percentage of variation explained by the factors is: Scenarios (S) – 42.80%, Caching (Ca) – 34.50%, the interaction of these (S+Ca) – 19.40%, and the error explained about 2.5%; the full experimental results of the three-factor analysis are presented in appendix A. The different scenarios and caching have the greatest impact on the variation in performance results; also, interaction of

² S1 represents Scenario 1 and S2 represents Scenario 2.

³ Ca represents With Caching and No Ca represents Without Caching.

scenarios and caching has some impact. The variation attributable to the different architecture is minor. The results of the ANOVA for the experiment with caching are presented in Table 5-12.

Scenario (S)	Caching (CA)	S+CA	Error
42.80%	34.50%	19.40%	2.28%

Table 5-12: Results of the analysis of variance (ANOVA) of experiment with caching

Summary of Experiment 2: Data caching in the middleware significantly improves the performance of data transmission; especially in case of transmission of larger data sets.

5.3.3 Experiment 3: Use of compression

Experiments with compression in the middleware use the same scenarios as in Experiment 1. In Experiment 3, we compress data in the middleware before transmitting it to the mobile application, and decompress data in the mobile application before displaying it on the mobile screen. We analyze the effect on data transmission due to compression and decompression in middleware and mobile application respectively for the scenarios and the architectures (C/A/S and C/I/S model). In this experiment, we use gzip compression (described in 4.3.2) and found compression significantly changes the total data size that can be transmitted; the change of data sizes due to compression is presented in Table 5-13.

Scenarios	Data size without compression	Data size after compression
Scenario 1	~ 2 KB	~0.4 KB
Scenario 2	~ 45 KB	~ 10.5 KB
Scenario 3	~ 230 KB	~ 56 KB

Table 5-13: Change of data size after data compression

The compressed data is in a binary format and must be encoded in order to transmit it through our web services. This requires that the mobile application decodes the data before decompression. In Experiment 3, scenario 3 does execute, unlike in both in

Experiment 1 and Experiment 2. The analysis result shows that there is no significant difference of data transmission time for the different software models (C/A/S and C/I/S). The results with compression for the scenarios are presented in Table 5-14.

Scenarios	C/A/S (SD)	C/I/S (SD)	Analysis
Scenario 1	478.67 ms (21.08)	545.00 ms (45.93)	C/A/S better; but negligible difference.
Scenario 2	875.67 ms (41.02)	817.67 ms (39.92)	C/I/S better; but not a significant difference.
Scenario 3	2365.00 ms (128.74)	2190.67 ms (63.36)	C/I/S better; but not a significant difference.

Table 5-14: Experiment with compression results and analysis

We conduct comparative analysis in between the results after compression and the results before compression to analyze the change of data transmission time due to compression. The results show that compression improves data transmission time over the basic data transmission; the experimental results are presented in Table 5-15.

Software Architecture	Scenario (S)	Compression (Co) ⁴	Results	SD
C/A/S	S1	No Co	525.67 ms	31.39
C/A/S	S1	Co	478.67 ms	21.08
C/A/S	S2	No Co	1073.00 ms	37.27
C/A/S	S2	Co	875.67 ms	41.02
C/I/S	S1	No Co	577.67 ms	24.21
C/I/S	S1	Co	545.00 ms	45.92
C/I/S	S2	No Co	1073.00 ms	37.27
C/I/S	S2	Co	817.67 ms	39.72

Table 5-15: Experimental results for software models, scenarios, and compression

⁴ Co represents Compression and No Co represents No compression in the middleware

The results of the analysis of variance (ANOVA) (full experiment results of the three-factor analysis are presented in Appendix B shows that the percentage of variation explained by the factors is: Compression (Co) – around 8%, Scenarios (S) - around 82%; the interaction of these (Co+S) around 4% and the error also explained about 4.5% of the variation. Clearly, the different scenarios had the greatest impact on the variation in results while the variation attributable to the compression is minor. But we found that the bigger the data size, the better compression performs. So, compression has an impact in case of large data large data transmission. The results of the ANOVA for Experiment 3 are presented in Table 5-16.

Scenario (S)	Compression (Co)	S+Co	Error
82.38%	7.96%	3.91%	4.57%

Table 5-16: Results of the analysis of variance (ANOVA) of experiments with compression.

Summary of Experiment 3: Using compression we can transmit larger data sets to mobile applications and also improves data transmission time.

5.3.4 Experiment 4: Experiments with encryption

In this experiment we analyze data transmission with encryption in the middleware and decryption in the mobile application. Here, the experimental scenarios are same as in Experiment 1. We use Base64 compression (described in 4.3.3) in the middleware and Base64 decryption in mobile application. The data encryption increases the size of the data; the change of data size due to encryption is presented in Table 5-17.

Scenarios	Data size before encryption	Data size after encryption
Scenario 1	~ 2 KB	~3 KB
Scenario 2	~ 45 KB	~ 60 KB

Table 5-17: Change of data size after encryption

In this experiment, scenario 3 does not work the same as in Experiment 1 and Experiment 2 as the data size is bigger. We analyze data for the two scenarios in two different software architectures (C/A/S and C/I/S model). From the experimental results, we found

that the C/A/S model performs better than the C/I/S model in the case of encryption, but the difference is small; the experimental results are presented in Table 5-18.

Scenarios	C/A/S (SD)	C/I/S (SD)	Analysis
Scenario 1	669.33 ms (15.95)	806.00 ms (81.07)	C/A/S better but negligible difference
Scenario 2	1134.00 ms (46.81)	1236.00 ms (15.72)	C/A/S better but negligible difference

Table 5-18: Experiment with encryption result and analysis

We have done a three-factor ANOVA analysis between the results after encryption and results before encryption. From the three factor analysis results, we found that encryption increases data transmission time than regular data transmission as data size increases; the experimental results are presented in Table 5-19 and the results of the ANOVA are presented in Table 5-20; the full experiment results of the three-factor analysis are presented in appendix C.

Software Architecture	Scenario (S)	Encryption (En) ⁵	Results	SD
C/A/S	S1	No En	525.67 ms	31.39
C/A/S	S1	En	669.33 ms	15.95
C/A/S	S2	No En	1073.00 ms	37.27
C/A/S	S2	En	1134.00 ms	46.81
C/I/S	S1	No En	577.67 ms	24.21
C/I/S	S1	En	806.00 ms	81.04
C/I/S	S2	No En	1073.00 ms	37.27
C/I/S	S2	En	1236.00 ms	15.72

Table 5-19: Three factor analysis on software models, scenarios, and encryption

The results of the analysis of variance (ANOVA) shows that the percentage of variation explained by the factors was: Encryption (En) – around 8%, Scenarios (S) - around 83%.

⁵ En represents Encryption and No En represents No Encryption in the middleware

The error also explained about 5% of the variation. Clearly, the different scenarios had the greatest impact on the variation in results while the variation attributable to the encryption was a minor.

Software Architecture (SA)	Scenario (S)	Encryption (En)	Error
1.89%	83.86%	7.94%	4.87%

Table 5-20: Results of the analysis of variance (ANOVA) of experiment with encryption.

Summary of experiment 4: Encryption increases data transmission, especially if the data size is larger, but it provides security to data which is important in wireless network.

5.3.5 4-Factor Analysis

In Experiments 1, 2, 3 and 4, we analyzed the performance of different techniques and different mobile software architectures. We combine the data from these experiments with additional experiments involving combinations of these data transmission techniques (caching, compression, and encryption) and test it in the different scenarios – small data and large data. We experiment with 16 combinations of techniques and software models for each of the data scenarios. The experiments for each of the data scenarios are summarized in Table 5-21.

Identity	Software Architecture (SA)	Caching (Ca)	Compression (Co)	Encryption (En)
1	C/A/S	Ca	Co	En
2	C/A/S	Ca	Co	No En
3	C/A/S	Ca	No Co	En
4	C/A/S	Ca	No Co	No En
5	C/A/S	No Ca	Co	En
6	C/A/S	No Ca	Co	No En
7	C/A/A	No Ca	No Co	En
8	C/A/A	No Ca	No Co	No En
9	C/I/S	Ca	Co	En

10	C/I/S	Ca	Co	No En
11	C/I/S	Ca	No Co	En
12	C/I/S	Ca	No Co	No En
13	C/I/S	No Ca	Co	En
14	C/I/S	No Ca	Co	No En
15	C/I/S	No Ca	No Co	En
16	C/I/S	No Ca	No Co	No En

Table 5-21: Experiment design of 4-factor analysis

From the four-factor data analysis we discovered that (experiment result is in appendix D):

1. For large data, caching significantly improves data transmission time.
2. For large data, encryption increase data transmission time.
3. Caching and compression together improve data transmission time.
4. Caching and encryption together increase data transmission time.
5. Caching, compression and encryption on the same data together increase the transmission time, because at the server end compression and encryption computing takes time, on the other hand the client end decompression and decryption take computing time.
6. No significant differences were found due to the software architecture (C/A/S vs. C/I/S), but in the case of large data, C/I/S seems to perform somewhat better than C/A/S.

The results of the analysis of variance (ANOVA) on the large data scenario (data size is larger than 50 KB) shows that the percentage of variation explained by the factors is: Caching (Ca) – 21.39%, Encryption (En) – 7.25%, the interaction of Software Architecture and Encryption (SA+En) – 7.22%, the interaction of Compression and Encryption (Co+En) – 7.23%, and the interaction of Software Architecture Caching and

Compression (SA+Ca+Co) – 8.45%; the other interaction terms explained a small percentage of the variation. Clearly, caching has the greatest impact on the variation in results; also, interaction of software architecture, caching and caching have the large impact on the variation in results. Encryption, combination of software architecture with encryption, and combination of compression with encryption also have impacts on the variation in results. The results of the ANOVA of the four-factor analysis for large data scenario are presented in Table 5-22.

Caching	Encryption	SA+En	Co+En	SA+Ca+Co
21.39%	7.25%	7.22%	7.23%	8.45%

Table 5-22: The results of the analysis of variance (ANOVA) on the large data scenario.

The results of the analysis of variance (ANOVA) on the small data scenario (data size is less than 10 KB) shows that the percentage of variation explained by the factors is: Encryption (En) – 11.49%, the interaction of Software Architecture and Encryption (SA+En) – 8.20%, the interaction of Caching and Encryption (Ca+En) – 8.80%, the interaction of Software Architecture, Caching and Encryption (SA+Ca+En) – 9.30%, the interaction of Software Architecture, Compression and Encryption (SA+Ca+En) – 7.29%, the interaction Caching, Compression and Encryption (Ca+Co+En) – 8.62%, and the interaction of Software Architecture, Caching, Compression and Encryption (SA+Ca+Co+En) – 9.75%; the other interaction terms explained very small percentages of the variation. The results of the ANOVA of the four-factor analysis for the small data scenario are presented in Table 5-22.

En	SA+En	Ca+En	SA+Ca+En	SA+Co+En	Ca+Co+En	SA+Ca+Co+En
11.49%	8.20%	8.80%	9.30%	7.29%	8.62%	9.75%

Table 5-23: The results of the analysis of variance (ANOVA) on small data scenario of the four-factor analysis

Summary of experiment 5: In the case of large data transmission, caching, and caching with compression performed better. On the other hand encryption slows down data transmission, even if we use encryption with caching and compression. Data

transmission with caching, compression and encryption together significantly slows down data transmission.

5.4 Summary

In this Chapter we describe the experiments and analyze their results. At the end of every experiment, we summarize the findings from each experiment. In the next Chapter, we discuss the findings and their implications.

Chapter 6

Discussion of Results

In this Chapter, we will discuss about our findings from Chapter 5, as well as provide our observations on mobile application middleware, the middleware API, and mobile intercept API. We also identify limitations of the research and our observations based on that.

6.1 Realizing Data Transmission Techniques

We experimented with caching, encryption and compression techniques for mobile application data transmission from remote data sources through a middleware. In Sections 5.3.1 (basic experiments), 5.3.2 (experiments with caching), 5.3.3 (experiments with compression), and 5.3.4 (experiments with encryption), we examined the impact of the various techniques on data transmission time with different scenarios involving different sized data sets. In Section 5.3.5, we analyzed additional experiments using combinations of these techniques in order to do a four-factor analysis on the small and large data set scenarios.

6.1.1 Caching

“Data caching in the middleware significantly improves the performance of data transmission; especially in the case of large data sets.”

In our research, we experiment on data transmission for mobile application connecting to remote enterprise databases or data sources. When caching is enabled, if mobile

application requests remote data, the request goes to the middleware which then looks at its local cache. If the data is present, the middleware returns the data back to the mobile application from its local cache. There is no database execution in this process, so data access happens faster than the basic process (without caching). In the case of large data, the impact of caching is noticeable because it saves expensive transaction time between the middleware and the remote database; also no data processing (converting raw data to XML) is needed in the middleware before returning data back to the mobile application.

Suggestions for mobile application developers:

1. Mobile application developers should use caching for large data which does not change frequently.
2. Caching can be used for small data, though it will not make any significant difference on data transaction time but it will save some database execution.

6.1.2 Compression

“Compression is an effective technique for very large data transactions for mobile applications; also, it improves data transmission time and provides data security.”

Our experiments showed that we can move large data sets to mobile applications by compressing. Compression squeezes data so that small amounts of data are passed over the wireless network with least transmission time.

Suggestions based on our observations:

1. Mobile application developers should use compression in the middleware to transmit large data over network faster.
2. Do not use compression for small data transmission, as it may increase total transaction time due the overhead of compression in the middleware and again in decompression in the mobile application.

3. Using data compression to minimize the data size before transmission saves bandwidth consumption.

6.1.3 Encryption

“Encryption increases data transmission time, especially if the data size is larger, but it provides security for the data, which can be important in a wireless network.”

Encryption can be very important in protecting the data transferred via a wireless network. This is particularly the case when transmitting data across public networks.

In our experiments, we found that encryption increase data transmission time because the total size of the data increases due to encryption; also, encrypted data needs to be decrypted in the mobile application which is expensive in mobile computing. We discovered that our test mobile application cannot decrypt the data if the data size is more than 60KB; this would depend on the amount of memory in the mobile device, but indicates that are limits that need to be considered.

Suggestions based on our observations:

1. Mobile application developers should use encryption in the middleware for small data to secure it.
2. Do not encrypt large data because it increases computing time and can cause memory limit exceptions.

6.1.4 Combination of techniques

Our suggestions based on the combination of caching, compression, and encryption techniques are as follows:

1. In case of large data sets, caching with compression provides faster data transmission, so a software developer can use the combination of compression and caching on large data sets to improve data transaction performance.

2. Do not use caching, compression and encryption techniques together for data transaction; it significantly slows down data transmission performance.

6.1.5 Mobile software architectures

We performed our experiments with the C/A/S and C/I/S models because they had been identified as the most plausible models for data requests by previous researchers. One objective of our research was to investigate whether one performed better than the other and for which scenarios and techniques.

Based on our research results, we conclude that there is little difference, performance-wise, between the techniques – both perform almost the same in a wireless network. In most cases we found data transaction time for the C/A/S model is a little smaller than the C/I/S model. In the case of larger data sets, transfer with the C/I/S model performed better than C/A/S model. But the transaction time differences were small.

Our suggestions regarding the mobile software model architectures are:

1. In the C/I/S model, the communication is through the *Intercept* API. In our experience, using the API makes mobile programming simpler and may be an advantage when developing application for heterogeneous mobile operating systems.
2. Using the C/I/S model provides reusable code, which can help improve the quality of the product. So even if in some circumstances the C/I/S model quantitatively require a little extra transaction time, it qualitatively improves the system.

Chapter 7

Conclusion

The demand for mobile applications is increasing every day. Today, people expect to have mobile applications to aid in banking, shopping, meetings, medical services, and other services. In our research, we focused on accessing remote data from enterprise databases or data sources in mobile application. We explored the use of the Client-agent-Server (C/A/S) model and the Client-Agent-Intercept (C/I/S) models; C/A/S and C/I/S are two well-known models for application development. Using middleware is one of the well-known techniques for enabling mobile applications to access remote data. We considered the impact of using caching, compression and encryption with these models in data transmission. Our goal was to understand the impact on performance, as measured by transmission time, of these techniques and the impact on different sizes of data sets. Based on our experiments, we were able to compare the performance of the software models and help clarify the good and bad effects of using caching, compression, encryption, and/or combination of these techniques in mobile applications relying access to remote data.

As part of our work, we developed a middleware API and intercept API. Using the middleware API and mobile application intercept API, the mobile application developer should be able to more easily develop mobile applications accessing remote data sources and more easily develop them for heterogeneous mobile environments.

A number of future directions exist based on this work:

1. We experimented with the C/A/S model and C/I/S model. Future work could look at extending the C/I/S model for client side pre-fetching and background threading to retrieve data from the middleware.
2. Our experiments used a Nokia E72 smart phone which runs the Symbian Operating System. It would be useful to evaluate the APIs by porting them to other mobile operating systems and developing mobile applications. There are many new and different mobile devices, and testing the APIs with smart phones such as iPhones, iPads, Tablets, etc would be useful.
3. With other devices, it would be useful repeat some of these experiments to see if similar results can be obtained. Newer devices have more capabilities, faster processors, more memory so the absolute times may be faster or data sets larger. It would, however, be interesting to see if the impact in performance of these techniques follows a similar pattern.

Finally, this research will help to identify efficient middleware models and techniques that can be used with mobile applications which must retrieve data from remote data bases or data sources. The results will be useful to developers of mobile applications.

References

- [1] **eMarketer:** Social Network Marketing to Reach \$2.5 Billion in 2011. *marketwire*. [Online] May 09, 2007. <http://www.marketwire.com/press-release/social-network-marketing-to-reach-25-billion-in-2011-733494.htm>. Last visited 2nd March, 2012
- [2] **Meyers J:** US Mobile Social Networking and the Millennial Generation. : In-Stat Mobile Consumer Service, Reed Elsevier, 2008.
- [3] **Insight Ipsos:** The Face of the Web. : Ipsos Insight Marketing Research Consultancy, Tavel, Modeling and Simulation Design. AK Peters Ltd. 2007.
- [4] **London: The Time:** The Future of Social Networking: Mobile Phones. : Times Newspapers Ltd. 2008. Last visited online on 2nd March, 2012
- [5] **Apple Inc.:** Apple Special Event. Apple Events. [Online] Apple Inc., October 04, 2011. <http://events.apple.com.edgesuite.net/11piuhbvdlbkvoih10/event/index.html>. Last visited 2nd March 2012
- [6] **Paul I:** Android Market Tops 400,000 Apps. PCWorld. [Online] January 04, 2012. http://www.pcworld.com/article/247247/android_market_tops_400000_apps.html. Last visited 2nd March 2012
- [7] **Christina B:** Google's 10 Billion Android App Downloads: By the Numbers. WIRED. [Online] December 08, 2011. <http://www.wired.com/gadgetlab/2011/12/10-billion-apps-detailed/>. Last visited 2nd March 2012.
- [8] **Gartner:** Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth. GartnerNewaroom. [Online] February 15, 2012. <http://www.gartner.com/it/page.jsp?id=1924314>. Last visited 2nd March 2012
- [9] **Rocha B P S, Rezende C G, Loureiro A A R:** Middleware for multi-client and multi-server mobile applications; 2nd International Symposium on Wireless Pervasive Computing, 2007. ISWPC '07, doi 10.1109/ISWPC.2007.342643
- [10] **Swaroop V, Shanker U:** Mobile distributed real time database systems: A research challenges; 2010 International Conference on Computer and Communication Technology (ICCCT), 2010, pp. 421-424, doi 10.1109/ICCCT.2010.5640495.

- [11] **Capra L, Emmerich W, Mascolo C:** Middleware for mobile computing: Awareness vs. transparency. Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, 2001. HOTOS '01. Washington, DC, USA: IEEE Computer Society, 2001, p. 164, doi:10.1109/HOTOS.2001.990080.
- [12] **Mascolo C, Capra L, Emmerich W:** Middleware for mobile computing (a survey). In Tutorial Proceedings of the International Conference of Networking 2002. Springer, 2002, pp. 20–58.
- [13] **Capra L, Blair G S, Mascolo V, Emmerich W, and Grace P:** Exploiting reflection in mobile computing middleware. SIGMOBILE Mobile Computing Communication. Rev., vol. 6, no. 4, pp. 34–44, 2002.
- [14] **Mascolo C, Capra N, Zachariadis S, and Emmerich W:** Xmiddle: A data-sharing middleware for mobile computing. Wireless Personal Communications, vol. 21, no. 1, pp. 77–103, 2002.
- [15] **Capra L, Emmerich W, and Mascolo C:** Carisma: Context-aware reflective middleware system for mobile applications. IEEE Transactions on Software Engineering, vol. 29, no. 10, pp. 929–945, October 2003, doi:10.1109/TSE.2003.1237173.
- [16] **Campbell A T:** Mobiware: Qos-aware middleware for mobile multimedia communications. In Proceedings of the IFIP TC6 seventh international conference on High performance networking, HPN '97, VII. London, UK, UK: Chapman & Hall, Ltd., 1997, pp. 166–183.
- [17] **Bellavista P, Corradi A, and Stefanelli C:** Mobile agent middleware for mobile computing. Computer, vol. 34, no. 3, pp. 73–81, 2001.
- [18] **Chan A T, Chuang S N:** Mobipads: A reflective middleware for context-aware mobile computing. IEEE Transactions on Software Engineering, vol. 29, no. 12, pp. 1072–1085, December 2003, doi:10.1109/TSE.2003.1265522.
- [19] **Rocha B P S, Rezende C G, Loureiro A A R:** Middleware for multi-client and multi-server mobile applications. 2nd International Symposium on Wireless Pervasive Computing, 2007. ISWPC '07
- [20] **Gehlen G, Mavromatis G:** Mobile Web Service based Middleware for Context-Aware Applications. In Proceedings of the 11th European Wireless Conference 2005, Vol. 2, p.p. 784-790, Nicosia, Cyprus, VDEVerlag, 2005.

- [21] **Gehlen G, Bergs R:** Performance of mobile Web Service Access using the Wireless Application Protocol (WAP). In Proceedings of World Wireless Congress 2004, p.p. 427-432, San Francisco, USA, 2004.
- [22] **WAPForum:** Binary xml content format specification. Version 1.3, wap-192-wbxml-20010725-a.[Online] <http://www.wapforum.org>, July 2001. Last visited 2nd March 2012
- [23] **Gupta S, Joshi A, Santiago J, Patwardhan A:** Query distribution estimation and predictive caching in mobile ad hoc networks. In Proc. of MobiDE, 2008, pp. 24–30
- [24] **Yin L and Cao G:** Supporting cooperative caching in ad hoc networks. IEEE Transactions on Mobile Computing, vol. 5, no. 1, pp. 77–89, Jan. 2006, doi:10.1109/TMC.2006.15.
- [25] **Chelubaraju B, Kousik A S R, and Rao S:** Anticipatory Retrieval and Caching of Data for Mobile Devices in Variable-Bandwidth Environments. 5th Annual IEEE International Systems Conference (IEEE SysCon 2011), Montreal, Canada, April 2011
- [26] **Huang J, Xiao Y, Liang Y:** A Novel Secure Access Method for Remote Databases Based on Mobile Agents. Natural Computation, 2009. ICNC '09. Fifth International Conference on , vol.5, no., pp.519-522, 14-16 Aug. 2009
- [27] **Peine H:** Application and programming experience with the Ara mobile agent system, 2002. Software-Practice & Experience 32, 515–541.
- [28] **Gray R S, Cybenko G, Kotz D, Peterson R A, Rus D:** D'Agents: applications and performance of a mobile-agent system.2002, Software-Practice & Experience 32, 543–573
- [29] **Spyrou C, Samaras G, Pitoura E, Evripidou P:** Mobile agents for wireless computing: the convergence of wireless computational models with mobile-agent technologies. 2004, Mobile Networks & Applications
- [30] **Meier J D, Alex H, David H, Jason T, Prashant B, Lonnie W, Rob B J, Akshay B:** App Arch Guide 2.0. [Online] <http://apparchguide.codeplex.com/wikipage?title=Chapter 19 - Mobile Applications>. Last visited 2nd March 2012.
- [31] **Komiya T, Ohsida H, Takizawa M:** Mobile agent model for distributed systems. 22nd International Conference on Distributed Computing Systems Workshops, 2002.
- [32] **IBM Corporation:** Aglets Software Development Kit Home. [Online]<http://www.trl.ibm.com/aglets/> . Last visited 2nd March 2012

- [33] **ic#code**: The Zip, GZip, BZip2 and Tar Implementation For .NET [Online] <http://www.icsharpcode.net/opensource/sharplib/> Last visited 2nd March 2012
- [34] **Greenberg M S, Byington J C, Harper D G**: Mobile agents and security. IEEE Communications Magazine, Vol.36, July 1998, pp. 76~85
- [35] **kSOAP2**: kSOAP. [Online] <http://ksoap2.sourceforge.net/>. Last visited 2nd March 2012
- [36] **ORACLE**: J2ME Web Services APIs (WSA), JSR 172. Sun Developer Network (SDN). [Online] <http://java.sun.com/products/wsa/>. Last visited 2nd March 2012
- [37] **kXML**: kXML, [Online] <http://kxml.sourceforge.net/>. Last visited 2nd March 2012
- [38] **Jonathan K**: Parsing XML in J2ME. Sun Developer Network (SDN). [Online] March 7, 2002. <http://developers.sun.com/mobility/midp/articles/parsingxml/>. Last visited 2nd March 2012
- [39] **Jazzlib**: Jazzlib. [Online] <http://jazzlib.sourceforge.net/>. Last visited 2nd March 2012
- [40] **Bouncy Castle**: The Legion of the Bouncy Castle [Online] http://www.bouncycastle.org/latest_releases.html. Last visited 3rd March, 2012
- [41] **Howard R**: Web Services with ASP.NET. MSDN. [Online] Microsoft Corporation, February 22, 2011. <http://msdn.microsoft.com/en-us/library/ms972326.aspx>. Last visited 2nd March 2012.
- [42] **AdventureWorks Database**: AdventureWorks Database for MySQL, [Online] <http://sourceforge.net/projects/awmysql/>. Last visited 4th March 2012.

Appendix A: Three-Factor Analysis for Caching

In case of three-factor analysis on software architectures, scenarios, and caching; we analyze the results of data with caching and without caching, in scenario 1 and scenario 2 for C/A/S model and C/I/S model.

The Results of the three-factor analysis of software architecture, scenario, and caching are as follows:

Software architecture (SA), scenario (S) and caching (Ca)	y-mean	y1	y2	y3	SD
C/A/S + S1	525.67 ms	501.00 ms	515.00 ms	561.00 ms	31.39
C/A/S+ S1 + Ca	470.67 ms	470.00 ms	472.00 ms	470.00 ms	1.15
C/A/S + S2	1073.00 ms	1030.00 ms	1093.00 ms	1096.00 ms	37.26
C/A/S + S2 + Ca	544.67 ms	535.00 ms	562.00 ms	537.00 ms	15.04
C/I/S + S1	577.67 ms	588.00 ms	550.00 ms	595.00 ms	24.21
C/I/S+ S1 + Ca	492.67 ms	495.00 ms	478.00 ms	505.00 ms	13.65
C/I/S + S2	1073.00 ms	1030.00 ms	1093.00 ms	1096.00 ms	37.26
C/I/S + S2+ Ca	622.33 ms	644.00 ms	619.00 ms	604.00 ms	20.20

Appendix B: Three-Factor Analysis for Compression

In case of three-factor analysis on software architectures, scenarios, and compression; we analyze the results of data with compression and without compression, in scenario 1 and scenario 2 for C/A/S model and C/I/S model.

The Results of the three-factor analysis of software architecture, scenario, and compression are as follows:

Software architecture (SA), scenario (S) and Compression (Co)	y-mean	y1	y2	y3	SD
C/A/S + S1	525.67 ms	501.00 ms	515.00 ms	561.00 ms	31.39
C/A/S+ S1 + Co	478.67 ms	461.00 ms	473.00 ms	502.00 ms	21.07
C/A/S + S2	1073.00 ms	1030.00 ms	1093.00 ms	1096.00 ms	37.26
C/A/S + S2 + Co	875.67 ms	922.00 ms	844.00 ms	861.00 ms	41.01
C/I/S + S1	577.67 ms	588.00 ms	550.00 ms	595.00 ms	24.21
C/I/S+ S1 + Co	545.00 ms	570.00 ms	573.00 ms	492.00 ms	45.92
C/I/S + S2	1073.00 ms	1030.00 ms	1093.00 ms	1096.00 ms	37.26
C/I/S + S2+ Co	817.67 ms	861.00 ms	783.00 ms	809.00 ms	39.71

Appendix C: Three-Factor Analysis for Encryption

In case of three-factor analysis on software architectures, scenarios, and encryption; we analyze the results of data with encryption and without encryption, in scenario 1 and scenario 2 for C/A/S model and C/I/S model.

The Results of the three-factor analysis of software architecture, scenario, and encryption are as follows:

Software architecture (SA), scenario (S) and Encryption (En)	y-mean	y1	y2	y3	SD
C/A/S + S1	525.67 ms	501.00 ms	515.00 ms	561.00 ms	31.39
C/A/S+ S1 + En	669.33 ms	665.00 ms	687.00 ms	656.00 ms	15.94
C/A/S + S2	1073.00 ms	1030.00 ms	1093.00 ms	1096.00 ms	37.26
C/A/S + S2 + En	1134.00 ms	1188.00 ms	1109.00 ms	1105.00 ms	46.80
C/I/S + S1	577.67 ms	588.00 ms	550.00 ms	595.00 ms	24.21
C/I/S+ S1 + En	806.00 ms	775.00 ms	898.00 ms	745.00 ms	81.07
C/I/S + S2	1073.00 ms	1030.00 ms	1093.00 ms	1096.00 ms	37.26
C/I/S + S2+ En	1236.00 ms	1219.00 ms	1250.00 ms	1239.00 ms	15.71

Appendix D: Four-Factor Analysis

In case of four factor analysis, we experiment with the combination of techniques and mobile software architecture we have been used in this research work. We perform this experiment for different scenarios. The results of the combination of software architectures and techniques are as follows:

Combination of techniques & software architectures	Small data scenario Transaction time (SD)	Large data scenario Transaction time (SD)
C/A/S +Ca+Co+En	483.00 ms (1.73)	566.67 ms (8.50)
C/A/S +Ca+Co	460.00 ms (17.44)	547.33 ms (17.68)
C/A/S +Ca+En	475.00 ms (8.51)	547.67 ms (14.47)
C/A/S +Ca	464.67 ms (3.06)	548.00 ms (14.53)
C/A/S +Co+En	532.67 ms (19.04)	887.00 ms (25.24)
C/A/S +Co	524.33 ms (28.15)	860.67 ms (9.61)
C/A/S +En	526.33 ms (39.58)	805.33 ms (22.30)
C/A/S	510.33 ms (13.01)	849.33 ms (24.79)
C/I/S +Ca+Co+En	502.00 ms (26.89)	607.67 ms (15.04)
C/I/S +Ca+Co	512.33 ms (9.72)	549.33 ms (44.23)
C/I/S +Ca+En	498.00 ms (4.59)	532.00 ms (24.98)
C/I/S +Ca	474.67 ms (30.43)	502.67 ms (11.72)
C/I/S +Co+En	652.67 ms (47.09)	927.00 ms (53.11)
C/I/S +Co	569.33 ms (11.24)	902.00 ms (28.67)
C/I/S +En	570.33 ms (28.36)	930.00 ms (30.79)
C/I/S	549.33 ms (27.32)	914.00 ms (25.52)

We represent the results of the analysis of variance (ANOVA) in the following table which shows the percentage of variation for the combination of technique in four factor analysis. In column 1 we represent the factors, in column 2 and column 3 we represent effect of the factors for small data scenario (scenario 1) and large data scenario (scenario 2).

Factors	Small data scenario (Scenario 1)	Large data scenario (Scenario 2)
Software Architecture (SA)	0.98%	2.87%
Caching (Ca)	1.10%	21.39%
Compression (Co)	3.35%	3.18%
Encryption (En)	11.49%	7.25%
SA+Ca	3.76%	2.56%
SA+Co	4.18%	5.59%
SA+En	8.20%	7.22%
Ca+Co	5.01%	6.34%
Ca+En	8.80%	4.57%
Co+En	7.54%	7.23%
SA+Ca+Co	5.73%	8.45%
SA+Ca+En	9.30%	5.42%
SA+Co+En	7.29%	4.92%
Ca+Co+En	8.63%	5.93%
SA+Ca+Co+En	9.75%	4.71%

Appendix E: Queries of Inventory System

There are four steps of the inventory system; the queries of the inventory system are as follows:

Step	Query
Step 1	select productcategoryid id, name from productcategory
Step 2	select productsubcategoryid id, name from productsubcategory where productcategoryid in (select ProductCategoryID from productcategory where name="'+_selectedString+'")"
Step 3	select productid id, concat(productid,'-',name, '-', productnumber) as name from product where ProductSubcategoryID IN(select productsubcategoryid id from productsubcategory where name="'+_selectedString+'")"
Step 4	select p.ProductID, Name, ProductNumber, Color, StandardCost, ListPrice, Size,Description, ph.ThumbnailPhoto photo from product p, productdescription d, productphoto ph where p.productid=d.productid and p.productid=ph.productid and p.productid="+id

Appendix F: Queries of Medical Information System

There are four steps of the medical information system; the queries of the inventory system are as follows:

Step	Query
Step 1	SELECT username, password, fullname FROM doctors WHERE username="paramUsername" and password="paramPassword"
Step 2	No database query calling from step 2
Step 3	select concat(p.patientid,'- ',firstname,' ',lastname,' [Age:',pd.age,', Sex:',pd.sex,']') name from patient p, patient_data pd where p.patientid=pd.patientid and (p.firstname like '%"'+name+'%" or p.lastname like '%"'+name+'%')
Step 4	SELECT patientid, Hospital_Number, age, sex, Height, Weight, Cardiovascular, Respiratory, Trauma, Surgical, Infectious, Malignancy, hosp_arrival_location, hosp_admission_unit, other_medical, Hosp_Outcome, Source, ICU_Outcome, Operative, dxcategory_text, ICUDIagnosisCode, Other_Diagnosis, LowPulse, HiPulse, LowMBP, HiMBP, LowSBP, HiSBP, LowTem, HiTem, LowRR, HiRR, rrVented, LowPaO2, LowPaCO2, LowFiO2, bgVented, LowPH, HiPH, LowHemat, HiHemat, LowWBC, HiWBC, LowCreatin, HiCreatin, Urine, LowUrea, HiUrea, LowSodium, HiSodium, LowPotas, HiPotas, LowAlbum, HiAlbum, LowBili, HiBili, LowGlu, HiGlu, Eye, Verbal, Motor, AIDS, Hepatic, Lymphoma, Cancer, Leukemia, chf, lungdisease, renalfailure, CVT, Cirrhosis, Immunosup, comorbidity, SpecifyDischarge, source_specify, Prior_Care, Prior_Care_Other, Chronic_Type, Acute_Type, classification, liver, immunosuppression, icu_lag, star, hosp_los, icu_los, hosp_death, icu_death, gcs, apache_aps, apache_age, apache_chpts, apache2, apache2_risk, apache2_predicted_death, apache3, Sepsis FROM patient_data WHERE patientid="+id

Curriculum Vitae

Name: Md. Ashrafur Rahaman

Post-Secondary Education and Degree: The University of Western Ontario
London, Ontario, Canada

September 2010- April 2012

M.Sc. Computer Science

American International University-Bangladesh

Dhaka, Bangladesh

May 2002- December 2005

B.Sc. in Computer Science

Related Working Experience:

Teaching and Research Assistant

The University of Western Ontario

September 2010- April 2012

Senior Software Engineer

Wolters Kluwer Financial Services

From October 2009 to September 2010

Lead Software Engineer

eGeneration Limited, Dhaka, Bangladesh

From November 2009 to October 2010