B

Medical PDA

by

MOHAMMAD MAHJOUB

Project submitted in partial fulfillment of the requirements for the Degree of Master of Science in Computer Science

Department of Computer Science and Mathematics

LEBANESE AMERICAN UNIVERSITY

June 2008
LEBANESE AMERICAN UNIVERSITY
School of Arts and Sciences - Beirut Campus

Project Approval Form (Annex V)

Student Name: Mohammad Mahjoub  I.D. #: 200403306
Project Title: Medical PDA

Program: Master of Science in Computer Science
Division/Dept: Computer Science and Mathematics
School: Arts and Sciences
Approved by: Ramzi A. Haraty
Project Advisor: [Redacted]
Member: Abdul Nasser Kassar

Date: June 27, 2008
Plagiarism Policy Compliance Statement

I certify that I have read and understood LAU's Plagiarism Policy. I understand that failure to comply with this Policy can lead to academic and disciplinary actions against me.

This work is substantially my own, and to the extent that any part of this work is not my own I have indicated that by acknowledging its sources.

Name: Mohammad Mahjoub

Signature: [Signature]

Date: 27/6/2008
I grant to the LEBANESE AMERICAN UNIVERSITY the right to use this work, irrespective of any copyright, for the University's own purpose without cost to the University or its students and employees. I further agree that the University may reproduce and provide single copies of the work to the public for the cost of reproduction.
Acknowledgment

I would like to thank my advisor Dr. Ramzi Haraty for his guidance throughout my project work. Also, I would like to thank Dr. Abdul Nasser Kassar for being in my project committee. Finally, I would like to thank my friends and family for their constant support.
Abstract

This idea of the software is to launch an initiative to encourage more physicians to use electronic prescribing in terms of recording all medical information about patients, their progress, and their examinations, and take the first steps to full clinical automation. In addition, patients will carry their medical history on their PDAs on the run. Elderly patients will get advantage of the reminder capabilities on their PDAs, which will set off an alert according to the doctor's prescription, thus reminding the patient to take the appropriate pill at the appropriate time. The patient will be reminded of doctor's advice in terms of dos and don'ts while taking the medicament (e.g., dosage, side effects, advisable food, and avoidable food). The patient will be able to retrieve his/her medical records via a synchronization process after being connected to the Internet.
List of Figures

Figure 1.1: iPAQ PDA Used in the Project
Figure 1.2: Architecture of the Project
Figure 2.1: .NET Framework Structure (Wikipedia, 2008)
Figure 2.2: Classes Used in the Project
Figure 2.3: In-house Charting Component
Figure 2.4: Creating Mobile Database
Figure 2.5: Synchronization Process
Figure 2.6: Opening Forms in .Net CF
Figure 3.1: Merge Replication Agent Architecture (Tiffany, 2008)
Figure 3.2: Merge Replication User Architecture (Tiffany, 2008)
Figure 3.3: Configuring Distribution – 1
Figure 3.4: Configuring Distribution - 2
Figure 3.5: Configuring Distribution - 3
Figure 3.6: Configuring Distribution – 4
Figure 3.7: Configuring Distribution - 5
Figure 3.8: Configuring Distribution - 6
Figure 3.9: Configuring Replication - 1
Figure 3.10: Configuring Replication - 2
Figure 3.11: Configuring Replication - 3
Figure 3.12: Installing Compact Tools - 1
Figure 3.13: Installing Compact Tools - 2
Figure 3.14: Installing Compact Tools - 3
Figure 3.15: Web Synchronization - 1
Figure 3.16: Web Synchronization - 2
Figure 3.17: Web Synchronization - 3
Figure 3.18: Web Synchronization - 4
Figure 3.19: Web Synchronization - 5
Figure 3.20: Web Synchronization - 6
Figure 3.21: Web Synchronization - 7
Figure 3.22: Web Synchronization - 8
Figure 3.23: Web Synchronization - 9
Figure 3.24: Web Synchronization - 10
Figure 3.25: Testing Setup
Figure 3.26: Replication Code
Figure 4.1: Web Services Platform (Wenz, 2007)
Figure 4.2: Web.config File Protocols
Figure 4.3: WSDL of the Web Service Used in the Project
Figure 4.4: Web Service Code
Figure 4.5: Replication Approaches (SQL Server CE Portal, 2008)
Figure 5.1: AJAX Architecture (Wenz, 2007)
Figure 5.2: Sample AJAX Code
Figure 6.1: Database Design
Figure 6.2: SQL CE Database
Figure 6.3: SQL CE – Transaction Input
Figure 6.4: SQL CE – Result Set
Figure 6.5: SQL CE – Statistics Screen
Figure 7.1: Web Application – Login Page
Figure 7.2: Web Application – Main Page (Tab 1)
Figure 7.3: Web Application – Main Page (Tab 2)
Figure 7.4: Web Application – Main Page (Tab 3)
Figure 7.5: Web Application – Main Page (Tab 4)
Figure 7.6: Web Service – Main Page
Figure 7.7: Web Service – Function Page
Figure 7.8: Main Screen
Figure 7.9: Activation Screen
Figure 7.10: Initialize Database Sub-Menus
Figure 7.11: Initialize Database Prompt
Figure 7.12: Synchronization in Progress
Figure 7.13: Synchronization Prompt
Figure 7.14: Compacting Database
Figure 7.15: Medical Record Form
Figure 7.16: Upcoming Appointments
Figure 7.17: Appointments Reminder
Figure 7.18: Graphs Form
Figure 7.19: Blood Pressure Chart
Figure 7.20: Cholesterol Chart
Figure 7.21: Glucose Chart
Figure 7.22: Weight Chart
List of Tables

Table 1.1: Important URLs for the Project

Table 4.1: Comparing Merge and Web Services Replication
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADO</td>
<td>ActiveX Data Objects</td>
</tr>
<tr>
<td>AJAX</td>
<td>Asynchronous JavaScript and XML</td>
</tr>
<tr>
<td>AppDomain</td>
<td>Application Domain</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASPX</td>
<td>Active Server Pages.Net</td>
</tr>
<tr>
<td>CE</td>
<td>Compact Edition</td>
</tr>
<tr>
<td>CF</td>
<td>Compact Framework</td>
</tr>
<tr>
<td>CLI</td>
<td>Common Language Infrastructure</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CORBA</td>
<td>Common Object Request Broker Architecture</td>
</tr>
<tr>
<td>CSS</td>
<td>Cascading Style Sheet</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>DCE</td>
<td>Distributed Computing Environment</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>GAC</td>
<td>Global Assembly Cache</td>
</tr>
<tr>
<td>GDI</td>
<td>Graphics Device Interface</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HTML</td>
<td>HyperText Markup Language</td>
</tr>
<tr>
<td>Acronym</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td>HTTPS</td>
<td>HyperText Transfer Protocol over Secure Socket Layer</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IIS</td>
<td>Internet Services Server</td>
</tr>
<tr>
<td>IL</td>
<td>Intermediate Language</td>
</tr>
<tr>
<td>JIT</td>
<td>Just-In-Time Compiler</td>
</tr>
<tr>
<td>OEM</td>
<td>Original Equipment Manufacturer</td>
</tr>
<tr>
<td>OLE</td>
<td>Object Linking and Embedding</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>PDA</td>
<td>Personal Digital Assistant</td>
</tr>
<tr>
<td>RDA</td>
<td>Remote Data Access</td>
</tr>
<tr>
<td>RPC</td>
<td>Remote Procedure Call</td>
</tr>
<tr>
<td>SMB</td>
<td>Server Message Block</td>
</tr>
<tr>
<td>SOA</td>
<td>Simple Object Access</td>
</tr>
<tr>
<td>SOAP</td>
<td>Service Oriented Architecture Protocol</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VS</td>
<td>Visual Studio</td>
</tr>
<tr>
<td>WWW</td>
<td>World Wide Web</td>
</tr>
<tr>
<td>WYSIWYG</td>
<td>What You See Is What You Get</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
Chapter 1
Introduction

Doctors and patients are not generally making use of PDAs to help improve patient care, and those that do use handheld devices are liable to stick to administrative rather than clinical tasks. Technology was meant to save time and help us provide better care, in addition to reducing medical faults.

With the Medical PDA program, doctors can quickly and professionally document all medical information about their patients, their progress, and their examinations. The software summarizes the patients' information in a very ordered way. At any time, it will be possible to easily retrieve information about them and update their records systematically during or after a clinical visit. Medical PDA is easy to use with a pleasant GUI interface.

In this chapter, the increased usage of technology in the health care field and the benefits it has for doctors and patients will be discussed.

1.1 PDAs usage in Health Care

Growth in multimedia tools has been growing remarkably. People are being tackled more and more with these technologies which both attract them by
their performances and worry them by their complications. Fortunately, developers have been, in the last few years, greatly improving the grab in hand of their programs in order to make these new products accessible to everyone.

![Figure 1.1: iPAQ PDA Used in the Project](image)

As for computers, the PDA has been the most recent discovery which has taken the market by storm. These portable tiny computers are so small that they can be glided in one's pocket, hence their name: Pocket PC (see Figure 1.1). Their structure, similar to the well-known desktop PC Windows, has been optimized under the name of Windows CE (Yao & Durant, 2003). Their use is therefore somehow matching.

The screen (up to 640x480 pixels) of these small devices displays up to 65,000 diverse colors. Many programs such as Word, Excel, Windows media player, Internet explorer, Acrobat reader exist on Pocket PC. They can also be connected to the Internet, which allows one to browse and surf as they do on desktop PCs. To get the utmost of it, this huge technology must be extended to the medical world. Most medical references, encyclopedias, textbooks, medical
conversion charts and formulas are available on the Internet. These can be stored directly in one computer, which enables doctors to carry a whole library in their pocket.

Pocket PC is therefore a crucial tool for doctors. Due to this new technology, medical patients' files have been replaced by computerized ones, which is the trend now in Lebanon in many hospitals and clinics. Doctors can now place in their pocket the equivalent of many references and records that contains information about their patients. Doctors and patients can check through medical records at any time of the day, without having to go through the conventional paper work routine.

1.2 Mechanism of the project

Two versions are introduced to fulfill the needs of doctors and patients. The desktop version will have:

- Patient tracker system used by doctors to keep records of their patients.
- An e-prescription component that enables the doctor to prescribe drugs and advise medical examinations.
- Synchronization component controlled by the doctor in order to synchronize his/her records with that of the patient.

The mobile version will have:

- Patient tracker system that enables patients to track personal health history on their PDAs.
- Appointments component that enables patients to follow up on his/her appointments.
- A reminder system that works in parallel with e-prescriptions to instantly alert the patient to take his/her drug.
- Graphical component that enables patients to track their weight, blood pressure, glucose, or cholesterol.
- Body mass index component that enable patients to measure their body fat based on height and weight.

The product will be desktop and mobile-based software used by health care professionals and patients. Doctors will use the desktop version to prescribe and update the history of their patients in terms of visits and medicaments. The desktop version is a web-based AJAX built website. The website will be hosted on the Internet using the www.mohammadmahjoub.com domain for the purpose of demonstration. Patients will use the mobile version to track their medical history and get various reminders and alerts anywhere, anytime since the Internet connection will be used to perform the synchronization. The middle layer that does the synchronization will be hosted on the Internet also, and part of the processing will be done on the mobile locally. It is worth noting that the web version can be used in a PDA environment, but this needs fine-tuning which is outside of the scope of this project.

Desktop (web-based) and mobile platforms (Pocket PC/Windows Mobile) will be used to realize this project, in addition to various hardware devices and software development applications and tools:
Hardware:
- HP iPAQ PDA with Pocket PC support
- Cradle
- Windows XP based computer (Intel Duo Core processors)

Software:
- Microsoft .NET Framework 2.0
- Microsoft .NET CF 2.0
- SQL Server 2005
- Microsoft Access Database
- SQL Server CE 3.1 Database
- Microsoft VS .Net IDE
- C# Language
- AJAX
- XML
- Advanced HTML
- JavaScript
- Web Services over SOAP Protocol
- Microsoft ActiveSync
- Pocket PC emulator
- Internet Domain Registration and Hosting for Demo Purpose

1.3 Installation and Deployment

The AJAX website and the Web Service will be built in a ready-way to be hosted on the Internet in an application root folder, under a registered domain. The mobile application can be installed using the CAB compressed installation
file. This file can be downloaded from the website www.mohammadmahjoub.com and installed locally on a PDA.

The URLs shown in table 1.1 apply:

<table>
<thead>
<tr>
<th>Description</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download and install Medical PDA on the PDA</td>
<td><a href="http://www.mohammadmahjoub.com/PDA.cab">www.mohammadmahjoub.com/PDA.cab</a></td>
</tr>
<tr>
<td>Web Service URL</td>
<td><a href="http://www.mohammadmahjoub.com/webservice/service.asmx">www.mohammadmahjoub.com/webservice/service.asmx</a></td>
</tr>
<tr>
<td>AJAX-based application</td>
<td><a href="http://www.mohammadmahjoub.com/webapplication/index.aspx">www.mohammadmahjoub.com/webapplication/index.aspx</a></td>
</tr>
</tbody>
</table>

1.4 General Architecture of the Project

The infrastructure needed in realizing this project is depicted in Figure 1.2. The physician will use the browser to enter appointments details and update the medical records. He/she will use the ASP.NET AJAX software hosted on the IIS server on www.mohammadmahjoub.com/webapplication/index.aspx. The IIS server will fetch the incoming requests from the database server. The database can be of any type (Microsoft Access, SQL Server, Oracle, DB2, MySQL, etc. . . ) since we are using a platform independent method for fetching and sending data. XML web services. The patient will be connected to the Internet and just pressing the synchronize button to retrieve his/her records. This requires Internet connectivity since the web service is hosted on the Internet.
www.mohammadmahjoub.com/webservice/service.asmx. As can be noticed, the synchronization process is unidirectional since the patient can’t edit his records and send them back to the doctor. The patient requires .NET CF and SQL CE installed on his/her PDA.

Figure 1.2: Architecture of the Project
Chapter 2
The .NET Framework Usage

The Medical PDA project is developed under the .NET framework, which is a convenient framework as the infrastructure of the project due to the heterogeneous nature of the services it offers.

.NET envelops a very wide range of software technologies. .NET offers the option to develop ASP.NET dynamic pages using very well-known programming languages; it was a point of interest in this project. Trivial HTML pages can be written using a notepad and can be hosted on a web server to offer static display, meanwhile ASP.NET pages can be authored in an IDE and offer dynamic environment that can be hosted on a web server and interact with the world. This was one of the reasons to go for .NET in this project since the desktop version of the Medical PDA application will be web-based. ASP.NET divides the HTML from the code, C# in this case, so that the code is compiled in .dll files are referenced as code behind in each .aspx form. Design is in one place, and coding is in another place; thus, providing a comprehensive dual environment. Another reason for adopting .NET framework was the requirement in building web-based database version of the program, so that physicians will use the web forms that will in turn communicate a database engine.
Many commercial IDEs support creating mobile applications, such as Sun’s Java NetBeans, Carbide C++, in addition to many others. But those post many requirements on the environments they run in. For example, creating .JAR executable files using NetBeans require a virtual machine to run on PocketPC and SmartPhones. Creating .SIS applications using Carbide C++ basically supports only Symbian operating systems, which is the proprietary of Nokia, Siemens, Motorola, and BENQ vendors. Each of these environments has its cons and pros, but my major reason for choosing .NET platform was its support in creating scalable mobile applications. Moreover, the same developing environment for desktop applications will apply, and this code will largely run unchanged on desktop PCs. Since data synchronization was at the heart of this project, it was necessary that the chosen platform must address the below issues:

1- Ability to work with the mobile database when the PDA is offline.
2- Synchronizing the local information with a server when the PDA is online.

In the Medical PDA application, the patient will have a read-only access to his/her medical records and will have the ability to do a unidirectional merging or synchronization process.

2.1 Types of Applications

.NET offers three types of application development platforms; desktop applications, web applications, and web services (Yao & Durant, 2003). Each is
based on a different set of services. The major thing to note here is that web services play the mediator role between those applications, providing data transfer with not GUI interface. XML Web Services are also adopted in this project as the layer implementing the synchronization of medical records.

Everyone is familiar with desktop applications, which are usually local software installed to serve a local need or act as a node in a distributed system. Such applications are the most powerful class, offering valuable GDI libraries, numerous components, extensible compatibility and driver options. But applications like that have to be accessed from the machine they are installed on.

Due to the turn down of desktop applications, web applications are used to give the program more accessibility options. A Web Form in the Medical PDA application is an ASP.NET AJAX form accessed from a Mozilla web browser. Those are dynamic pages supported on a web server, IIS in this case. To access those pages, the client must be able to connect to the web server, whether on the Internet or Intranet. The client requests the information by typing a URL, submitting data in a form, or sending Get/Post/Put commands and waits for the result processed by the server and sent over HTTP/HTTPS protocols. The client, a physician in this case can add a patient to his records, ask the server to retrieve information regarding a certain patient, for view, edit, delete, or update purposes.
Regardless of any platform used, web services support heterogeneous tunnels to send over data from one server to another. It is a great advantage which can be a massive reason to adopt .NET in implementing any project. It gives any project scalability and expandability support. There’s no need to go through all the hassle of using legacy middleware layers in which each suffers from a down side, web services come to eliminate those problems by unifying the mechanisms of transferring data. By time, web services are going to grow in significance to be the standard RPC protocol. Enough reasons to start learning web services and implement them are reliability and speed offered by web services. Not to forget security reasons; which are controllable by the programmer depending on the environment.

2.2 Programming Paradigm

.Net framework is designed to accommodate multiple platform programming. The paradigm supports three different types of applications that ranges from desktop, mobile to web. It is worth mentioning the components that make this environment realizable. From Programming the .NET Compact Framework (Yao & Durant, 2003), the essential components are discussed below:

- **Common Intermediate Language**: Is a low level readable programming language.
- **Common Language Runtime**: C# code must be run by a virtual machine named Common Language Runtime.
• **Common Language Specification**: This is a basic set of rules that put constraints on any language in .NET in order to ensure interoperability with other languages.

• **Common Type System**: It takes care of allocating data types in memory.

• **Common Language Infrastructure**: This is a pattern that describes how languages written in different platforms interoperate.
The image depicted in Figure 2.1 demonstrates the role each of the components describe play.

![Diagram of .NET Framework Structure](image)

Figure 2.1: .NET Framework Structure (Wikipedia, 2008)

2.3 .Net CF

Since the Medical PDA project has a major mobile application, it is essential to discuss the capabilities of the framework supported, the CF or the
.NET CF. It is a tiny version of the .NET framework that targets mini devices such as PocketPCs and SmartPhones.

To understand the infrastructure implemented in this project, it is necessary to shed the light on some low level details of the CF. Windows CE was developed to be a tiny operating system with Win32 defining its programming interfaces (Yao & Durant, 2003). Even if access to device drivers is required, the same APIs allow such access. Another API is COM, which runs on Windows OS. Using COM, applications are divided into modules and components with each having a defined function. Those modules can be used among different applications. So reusability was the idea behind COM.

When the .NET project was announced, the perception of an application through APIs changed. This referenced the .NET Framework on desktops and CE platforms. The change was the introduction of assemblies to take over previous APIs. Basically two types of assemblies, .dll and .exe named library and process assemblies respectively.

CF runs on all versions of the Pocket PC. At the beginning of the project it was hard to differentiate between those versions, but here is a list of Pocket PC versions.

Pocket PC 2003 which is used in this project demonstration runs on Windows CE 4.2. (Used in this project).

Design Goals

To develop a well-structured mobile application, it will be essential to understand the design specification before starting the programming phase. Programming and designing any application in a mobile environment must take into consideration the below constraints:

- Tiny Kernel Image

  Small kernel images accompany any PDA. Those images can be flushed and upgraded through special utilities provided by the manufacturer. No installation of OS takes place as in the case of desktops. The OS lies on the embedded ROM. Those kernels support trivial OS functions such as memory and thread management, major protocols support.

- Component-Based

  Many operating systems are monolithic, that is each access request to system hardware must pass through the operating system’s kernel. The situation is different when it comes to Windows CE architecture since its made up of many modules.
• Handy

Many mobile operating systems failed due to the lack of support and introduction of up-to-date programs, in addition to interoperability issues and driver’s unavailability. Such reasons can be a major confront to the success of any mobile operating system since it would be obsolete and not fully utilized. Many applications and support drivers must be available to help in developing mobile applications.

• Convenient

A very important feature when programming in .NET CF is the code portability between PDAs made by various CPU manufacturers, and between a PDAs and desktop computers. This is very important factor, especially with the rise of many CPU manufacturers these days.

• Linked

A PDA without communication is just another dull device. Interfaces are increasing with the passage of time to include Bluetooth, Wi-Fi, Infrared, and extension to RFID and digital modems through compact flash extension slots. Communication is the basic feature that made the Medical PDA project possible.

• Real-time

Real-time programming is the capability to execute tasks (calculation, sending/receiving, displaying graphs) within a time limit. A feature that is capable
of doing a lot, especially in the health and stock domain. Health parameters such as blood pressure and heart beat rate are measured using real-time devices incorporated with PDAs. Dynamic stock prices are also integrated in many PDA applications.

2.4 CF Files

Visual Studio.NET uses many .dll and .exe assembly files to connect to the PDA over ActiveSync in order to deploy installation files (Yao & Durant, 2003). The size of the typical CF library is 2,384 KB. Using additional references and classes such as database drivers will increase the size to 2,400 KB. This can be thought as a huge size to a PDA environment, but PDAs compresses those libraries before deploying them; it also uses GAC to store libraries for faster loading and less space usage (Yao & Durant, 2003).

2.4 CF Capabilities Explored in This Project

CF has 18 libraries in total, which is 2.5 MB in size. The .NET Framework has 86 libraries and occupies nearly 40 MB.

2.4.1 Runtime Data

CF utilizes the CLI, which means that it supports all the data types used on desktops. C Sharp will be used in implementing this project. Visual Basic is also supported in developing mobile applications in .NET environment.
As one can notice, this is a huge reduction from the 20 programming languages supported on desktop to just two languages. But after all C# and VB.NET can work hand in hand since their assemblies can easily interoperate.

CF provides just in time compilation of code to authenticate proper loading of modules before using them, and as in the normal framework, CF supports OS functions such as memory allocation, heap and stack management, and garbage collection.

CF supports AppDomains, a type of lightweight process. This lessens the cost and the burden of switching from one context to another; thus, leading to better management of resources.

In Windows CE, there's a limit to the number of processes created, which is 32. The very new version of Windows Mobile 6 supports unlimited number of processes creation (Whitney, 2007).

2.4.2 Supported Classes

As in the desktop environment, GUI is supported in the CF namely in the namespace System.Windows.Forms. 28 out of 35 components found in the normal .NET framework are available in the CF. All of the components are fine tuned to handle size and screen constraints found in PDAs.
Another important class supported is XML web services. As used in this project, PDA client can invoke XML web services and use them, but in contrary to the normal framework, the PDA can not be used to provide XML web services to other. To speak about the major difference between the normal framework and CF, it is well-known that the CF has no support for Web Forms. These forms can only run on ASP.NET web server such as IIS hosted in the following environments:

- Windows 2000, 2003 Server (All versions)
- Windows Vista and XP

CF does not support authoring environment for web forms, but all PDAs support web browsing and include a tiny version of Internet Explorer, which is less featured than the desktop one. The AJAX form of the application developed in this project can surely be accessed by the PDA. Some of the classes utilized listed in Figure 2.2 below:

```csharp
using System;
using System.ComponentModel;
using System.Data.SqlClient;
using System.Data.Common;
using System.IO;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Web.Services;
using MobileApp.localhost;
using System.Threading;
using Microsoft.WindowsCE.Forms;
using System.Runtime.InteropServices;

Figure 2.2: Classes Used in the Project
```
2.4.3 Graphics

With its competent graphics generation engine, Windows CE can not be compared to the GDI functions of the normal .NET Framework. It provides support for the three essential drawing classes (Yao & Durant, 2003):

- Raster (Bitmaps)
- Vector (Mathematically related Figures)
- Text (Strings)

Minimal number of fonts is supported for drawing strings. A normal Pocket PC, come with only four fonts. Rotation and displaying of fonts is not supported though.

Among the problems faced in developing a mobile application is drawing coordinated, especially when writing the code for the medical charts. CF only supports drawing in pixels and not in any other unit.

Graphics Code Snippet

The code below illustrates the generation of the weight graphs. It is worth noting that chart component is not originally supported by Microsoft. There are two options to display a chart. The two options were either to buy a ready made component on the Internet or to develop in-house charting generation routine. The decision was to use the second option. The code in Figure 2.3 fetches the required Figured from the local SQL CE database and displays the chart taking
into consideration the width and height of the client’s PDA screen size. Those charts are adaptable to run on any screen size.

```csharp
    g = this.CreateGraphics();
    connString = "Data Source=\My Documents\MainDB.sdf;";
    width = this.ClientSize.Width;
    height = this.ClientSize.Height;

    myFont = new System.Drawing.Font("Arial", 7, FontStyle.Regular);
    myFont2 = new System.Drawing.Font("Arial", 10, FontStyle.Bold);
    myBrush = new SolidBrush(System.Drawing.Color.Black);

    myconn = new SqlConnection(connString);
    myconn.Open();
    mycmd = myconn.CreateCommand();
    mycmd.CommandText = "SELECT Weight, DOV FROM Patient ORDER BY DOV DESC";
    reader = mycmd.ExecuteReader();

    int x = 1, y = 4;
    float get;
    g.Clear(Color.LightBlue);
    while (reader.Read() && x <= 5)
    {
        if(reader[0].ToString() != "")
        {
            get = float.Parse(reader[0].ToString());
            g.FillRectangle(new SolidBrush(Color.Blue), ((x + y) * width / 6), height * 2 / 3, 30, (int)get * (height * 2 / 3) / 200);
            g.DrawString(reader[0].ToString(), myFont, myBrush, ((x + y) * width / 6) + 5, (height * 2 / 3 - 12 - ((int)get * (height * 2 / 3) / 200)));
        }
        x++;
        y = y - 2;
    }

    g.DrawLine(blackpen, width / 6 - 10, height * 2 / 3, width - 10, height * 2 / 3);

    g.DrawLine(blackpen, width / 6 - 10, height * 2 / 3, width / 6 - 10, 10);

    g.DrawString("Weight Chart (in Kg)", myFont2, myBrush, width / 5, 185);

    g.DrawString("- 200 Kg", myFont, myBrush, 20, 5);
    reader.Close();
    myconn.Close();
```

Figure 2.3: In-house Charting Component
2.4.5 ADO.NET (Data Classes)

Data classes take a great segment of the CF libraries. Those classes were fully utilized in this project. Among the classes used is the DataSet, which is an in-memory relational structure with XML support. DataSet is the main ADO.NET class that contains data in the form of tables, implementing DataTable instances (Yao & Durant, 2003). On its own, a DataTable is a set of fields and rows used in CF instead of multidimensional arrays holding string data. The idea of a table lying in a database can be an analogy to a DataTable lying in a DataSet.

ADO.NET objects can be accessed by invoking the System.Data namespace. The DataGrid control offers an easy way to display data in DataTables. DataGrids is implemented on the web and mobile version of Medical PDA. Data grids are deployed on the mobile and web versions of this project. The major difference is simple, web data grids come with more options such as select, update, and delete data, while the mobile version comes with only one option; select data for display (Trotschi, 2002).

ADO.NET supports accessing database files and servers. Data providers come to play the role of the mediators between a database server and an application. Those providers help in accessing the database, issuing transactions and retrieving information back to the application. CF comes with two data providers for SQL Server environment (Wagner, 2004). The first is System.Data.SqlClient.dll. It helps accessing remote database servers from a
mobile application. This needs a direct connection to the server. Another data provider is System.Data.SqlClient.Ce.dll. It is used to access the mobile databases from a mobile application. In this case, accessing MainDB.sdf from the Medical PDA application. To make this possible, SQL Server CE must be installed on the mobile device.

2.4.6 ADO.NET Code Snippet

Using the above mentioned SQL CE data provider, the code in Figure 2.4 creates a local mobile database (MainDB.sdf) and initializes the schema. Before permitting the patient access to his/her records, the below routine must be invoked.

```csharp
try
{
    SqlCeEngine engine = new SqlCeEngine(connString);
    engine.CreateDatabase();

    SqlCeConnection sqlConn = new SqlCeConnection("Data Source=" + My Documents + "\MainDB.sdf;");
    sqlConn.Open();
    SqlCommand createTable = sqlConn.CreateCommand();
    try
    {
        createTable.CommandText = "CREATE TABLE Patient(id1 INT, FName NVARCHAR(20), LName NVARCHAR(30), Sex NVARCHAR(10), DOB NVARCHAR(20), PatientID1 NVARCHAR(20), Height NVARCHAR(20), id2 INT, PatientID2 NVARCHAR(20), DOV DATETIME, Diagnosis NVARCHAR(255), Medication1 NVARCHAR(50), Medication2 NVARCHAR(50), Time1 NVARCHAR(20), Time2 NVARCHAR(20), Time3 NVARCHAR(20), Time4 NVARCHAR(20), Time5 NVARCHAR(20), Time6 NVARCHAR(20), NextVisit NVARCHAR(20), Weight NVARCHAR(20), Cholesterol NVARCHAR(20), BloodPressure NVARCHAR(20), Glucosa NVARCHAR(20), ToDo NVARCHAR(255))"";
        createTable.ExecuteNonQuery();
        MessageBox.Show("Database and schema initialized successfully", "Setup", MessageBoxButtons.OK, MessageBoxIcon.Asterisk, MessageBoxIcon.Default, Button1);
    }
}
```
catch (SqlCeException exSQL4)
{
    MessageBox.Show("Unable to create table. " + exSQL4.Message, "Error",
        MessageBoxButtons.OK, MessageBoxIcon.Hand, MessageBoxIcon.DefaultButton.Button1);
}
sqlConn.Close();
}

Figure 2.4: Creating Mobile Database

2.4.7 XML Support

Another advantage of the CF is the XML support. XML allows the programmer to customize the creation of tags to fit any situation and structures data for heterogeneous access (Turtschi, 2002).

XML is a multi-purpose markup language. It is used in ADO.NET in defining DataSets and is the backbone of the SOAP protocol which is foundation of web services. This protocol will be utilized to perform the synchronization routine between the PDA and the main records storage database. In brief, CF gives a small subset of XML services found in the normal framework, which will meet the needs in accomplishing this project.

2.4.8 Synchronizing medical history

The most tedious part in this project is the synchronization routine done for the medical records. The process starts with referencing a remote web service found at the URL: www.mohammadmahjoub.com/webservice/service.asmx and passing the PatientID parameter. Results will be returned as a DataSet over the SOAP
protocol. Those data is a set of XML tag entries. A local table is created and assign to the XML fetched data. Now a table is established with the required retrieved medical records. The Patients table is named $dt$ and the records were moved over SOAP, then the DataSet is updated to reflect the changes to the mobile database, as can be shown in Figure 2.5.

```csharp
try {
    StreamReader tr = new StreamReader("\My Documents\ID");
    id = tr.ReadLine();
    tr.Close();

    localhost.Service mojo = new localhost.Service();
    retrieved = new DataSet();
    retrieved = mojo.GetDataSet(id);
    myTable = retrieved.Tables["Patient"];  
}

catch (Exception ex) {
    MessageBox.Show("Please activate the application first. " + ex.Message, "Sync Error",
        MessageBoxButtons.OK, MessageBoxIcon.Hand, MessageBoxIcon.DefaultButton.Button1);
}

SqlConnection sqlConn2 = new SqlConnection("Data Source=\My
Documents\MainDB.sdf; ");

try {
    sqlConn2.Open();

    string qry = "select * from Patient";

    SqlCeDataAdapter da = new SqlCeDataAdapter();
    da.SelectCommand = new SqlCommand(qry, sqlConn2);

    SqlCommandBuilder cb = new SqlCommandBuilder(da);

    DataSet ds = new DataSet();
    da.Fill(ds, "Patient");

    DataTable dt = ds.Tables["Patient"];

    SqlConnection sqlConnCheck = new SqlConnection("Data Source=\My
Documents\MainDB.sdf; ");
    SqlCommand cmdCheck = sqlConnCheck.CreateCommand();
    cmdCheck.CommandType = CommandType.Text;

    sqlConnCheck.Open();

    sqlConn2.Close();

    MessageBox.Show("Data Synced Sucessfully.");
} catch (Exception ex) {
    MessageBox.Show(ex.Message);
}
```

25
if (myTable != null)
{
    foreach (DataRow myRow in myTable.Rows)
    {
        cmdCheck.CommandText = "SELECT id2 FROM Patient WHERE id2 =" + myRow[7];
        dreader = cmdCheck.ExecuteReader();
        if (dreader.Read() == false)
        {
            DataRow newRow = dt.NewRow();
            newRow[0] = myRow[0];
            newRow[1] = myRow[1];
            newRow[2] = myRow[2];
            newRow[3] = myRow[3];
            newRow[4] = myRow[4];
            newRow[5] = myRow[5];
            newRow[6] = myRow[6];
            newRow[7] = myRow[7];
            newRow[8] = myRow[8];
            newRow[9] = myRow[9];
            newRow[10] = myRow[10];
            newRow[12] = myRow[12];
            newRow[13] = myRow[13];
            newRow[14] = myRow[14];
            newRow[15] = myRow[15];
            newRow[16] = myRow[16];
            newRow[17] = myRow[17];
            newRow[18] = myRow[18];
            newRow[19] = myRow[19];
            newRow[20] = myRow[20];
            newRow[21] = myRow[21];
            newRow[22] = myRow[22];
            newRow[23] = myRow[23];
            newRow[24] = myRow[24];
            dt.Rows.Add(newRow);
        }
    }
}

da.Update(ds, "Patient");
dreader.Close();
sqlConnCheck.Close();
sqlConn2.Close();
MessageBox.Show("Synchronization completed successfully", "Note", MessageBoxButtons.OK,
MessageBoxIcon.Asterisk, MessageBoxIcon.DefaultButton.Button1);
}
}
catch (SqlCeException exSQL5)
{
    MessageBox.Show("Synchronization error. " + exSQL5.Errors[0].Message, "Error",
    MessageBoxButtons.OK, MessageBoxIcon.Hand, MessageBoxIcon.DefaultButton.Button1);
}
The usage of HTML tags in CF is demonstrated in Figure 2.5. This shows the upcoming appointments notification implementation.

```html
notifyHtml.Append("<p><form method="GET" action=mynote;">\nnotifyHtml.Append("<p>Next appointment on: ");
notifyHtml.Append(next);
notifyHtml.Append("<p>Would you like to set it as a reminder? <input type="button" name="cmd:2" value="Yes"> <input type="button" name="cmd:2" value="No">\n\nnotifyHtml.Append("</body></html>\n```

Figure 2.5: Synchronization Process

2.5 Opening and Closing Forms in CF

**Medical PDA** application consists of many forms. In order to avoid memory leakage, these forms must be handled carefully upon opening and closing. This can cause a huge impact on the performance of the application. When the user presses a display form button, a routine must be triggered to test if this form is in memory before opening another replica. If it is in memory, it is only shown and no new instance is created. The procedure followed in this project is depicted in Figure 2.6.
Figure 2.6: Opening Forms in .Net CF
Chapter 3
Merge Replication

Having a mobile database and a standalone server urges the need to replicate data between both storages. Although the Medical PDA application is finalized using the XML Web Services data synchronization approach, but before doing that, replication using SQL Server 2005 merge replication was tested. The first option was merge replication, for that this chapter discusses the steps taken. Working with merge replication is about configuring the SQL Server 2005 database system to accept incoming synchronization requests (Nielsen, 2007). It is a matter of server administration and configuration. XML Web Services option was used since it allows full control over replicated data. Everything needs to be written in code from zero and there is no place for configuration.

In the project, the mobile database is named the Subscriber database, and the residing SQL Server 2005 database is named the Publication database. By means of a hub and spoke design, this technology is used to replicate data from the mobile database out to main database. In general, the mobile database can undergo many transactions to be then reflected to the Publisher. The major advantage of merge replication is the offline architecture. Merge replication used to be adopted between servers at different sites, but with the introduction of SQL Server CE databases, merge replication got adopted by the mobile domain.
(Tiffany, 2008). Making changes to database does not require a live connection, but replicating the data requires that indeed. To realize this, merge replication uses HTTP and SSL protocols in addition to the well known IIS server. It is known that merge replication offers the most sophisticated and scalable data synchronization technology. The basic units in performing merge replication are listed below:

- **Publisher Database**

  This is the parent SQL Server 2005 database hosted on the server.

- **Publication**

  The SQL Server 2005 database might contain many databases. The Publication is the database we are making available, MainDB in our case. We can choose the granularity level of making this database available to replication; we can choose tables, couple of tables, or the whole database.

- **Article**

  The portions of our database we need to make available, as discussed above, are named articles.

- **Distributor**

  It is the SQL Server, which gathers data and snapshots about the replication. It is usually found on the publisher server.

- **Subscriber**

  The SQL CE database running on the mobile client is the subscriber. It receives data from the publisher, edits this data, and then sends it back. The agent that makes this process achievable resides at the subscriber side. It manages all
communication between the mobile database engine and the SQL Server Compact Agent running on IIS.

- Subscription
This is the client’s request to a publisher to get a copy of the data it needs.

- IIS
The most sophisticated part lies in understanding the merge replication approach. It uses IIS and SQL Server Compact Agent to provide HTTP(S) access to all the Subscribers.

Things might be vague so far, so the configuration for the above components is presented in the next section. To fully utilize the benefits or merge replication, the above servers must be installed under a domain, and Active Directory must be in place.
Figure 3.1 depicts the logical setting for the above servers described. They can be seen working together in a unified environment with lots of agents. The environment is based on a domain implemented and the security lookups are testing credentials in the Active Directory.

Figure 3.1: Merge Replication Agent Architecture (Tiffany, 2008)
Figure 3.2 describes the merge replication environment with respect to user’s physical view.

![Diagram of merge replication environment](image)

Figure 3.2: Merge Replication User Architecture (Tiffany, 2008)

3.1 Configuring the Distributor

Before configuring the server, the snapshot folder that contains the schema and data from the published database was created. To do it the right way, a domain user with appropriate access rights must be created. This account will be used to perform the synchronization via IIS. To configure the distributor,
one must go to the replication node in the SQL Server 2005 and right click it and press configure distribution to start the process as shown in Figure 3.3.

![Configure Distribution Wizard](image)

Figure 3.3: Configuring Distribution – 1

To make things easier, one server will be used as distributor and publisher. The first option is chosen as shown in Figure 3.4.

![Configure Distribution Wizard](image)

Figure 3.4: Configuring Distribution – 2
Next, the shared path of the snapshot folder created previously is selected, as shown in Figure 3.5.

![Figure 3.5: Configuring Distribution – 3](image)

As indicated in Figure 3.6, the database name to be distributed is entered and the local server to be the publisher is chosen.

![Figure 3.6: Configuring Distribution – 4](image)

Upon getting the screen shown in Figure 3.7, the second option is checked to apply the configuration made.
Upon completing the configuration process, the distributor will be configured and the results will be prompted as displayed in Figure 3.8.
Many additional options were used for configuring security in communication with the distributor. This will not be discussed in details here since it is all related to Active Directory users and groups security.

3.2 Replication over Web

The whole replication process is accomplished over the HTTP(s) protocol (Tiffany, 2008). So it is important to install and configure the part that deals with sending data to such service. SQL Server replication components must be installed before going further. These can be installed from the OEM SQL Server 2005 CD, as shown in Figure 3.9. The installation script checks the compatibility of the system as shown in Figure 3.9.

![System Configuration Check](image.png)

Figure 3.9: Configuring Replication - 1
The next step is to install the replication components, as shown in Figure 3.10.

![Figure 3.10: Configuring Replication - 2](image)

Installation process begins, as shown in Figure 3.11.

![Figure 3.11: Configuring Replication – 3](image)
3.3 SQL Server Compact Server Tools

SQL Server Compact Server Tools need to be installed after performing the upgrade above. We go to the path: \Program Files\Microsoft SQL Server\90\Tools\Binn\VSShell\Common7\IDE and install the .msi file sqlc30setup.msi, as depicted in Figure 3.12.

![Image](image.png)

Figure 3.12: Installing Compact Tools - 1
This installation performs a quick check for the systems requirements as shown in Figure 3.13.

Figure 3.13: Installing Compact Tools – 2

Figure 3.14 shows the version of SQL Server to synchronize with. It also illustrates the installation path where the Server Agent (isapi.dll) and log files will be placed to be mapped to IIS Virtual Directories.

Figure 3.14: Installing Compact Tools – 3
3.4 The Web Synchronization Wizard

SQL Server components are installed and SQL Server compact tools are ready on the IIS server, the final step is to configure web synchronization. One needs to create a virtual directory and establish a proper access list to this folder to be available to the subscriber. The wizard is initiated as shown in the Figure 3.15.

![Figure 3.15: Web Synchronization - 1](image1)

After the welcome screen, the subscriber type is chosen. Of course it is a mobile database as shown in Figure 3.16.

![Figure 3.16: Web Synchronization - 2](image2)
In Figure 3.17, the server hosting the IIS is entered. Also the option for creating a virtual directory is needed.

![Web Synchronization - 3](image)

Figure 3.17: Web Synchronization - 3

SSL option gives the chance to configure secure communication between the SQL Server and the mobile database. Certificates can be used if they are available on the IIS server. For the sake of simplicity, SSL option will not be used, as shown Figure 3.18.

![Web Synchronization - 4](image)

Figure 3.18: Web Synchronization - 4
Subscribers can directly connect to the IIS server and synchronize, or they need to provide credentials. The second option is chosen to secure the synchronization process as shown in Figure 3.19.

Figure 3.19: Web Synchronization - 5

Subscribers' authentication can be made in several ways: integrated and basic authentication can be made if the client is a member of the Active Directory domain or not respectively. The method used is shown in Figure 3.20.

Figure 3.20: Web Synchronization - 6
Figure 3.21 shows the directory access, which permits users and groups to access the virtual directory for replication purposes.

![Figure 3.21: Web Synchronization - 7](image)

The final step is to refer to the snapshot folder which is seen as a network shared folder, as shown in Figure 3.22.

![Figure 3.22: Web Synchronization - 8](image)
Figure 3.23 summarizes the configuration part.

![Image of configuration wizard for Web Synchronization]

Figure 3.23: Web Synchronization – 9

The configuration progress is shown in Figure 3.24 where the virtual directory is created, isapi dll file is copied, and appropriate permissions are set.

![Image of details of configuration wizard]

Figure 3.24: Web Synchronization - 10
3.5 Putting it all to test

After configuring all of the agents and servers, it is a good idea to test what has been done. In place is a website hosted in a virtual directory on an IIS server. The file sqlesa30.dll is copied to that directory as part of configuring the agents. To see if the agent is functioning, we open the URL of the dll file http://syncweb/ssce/sqlesa30.dll. A window prompting username/password is displayed. Upon entering the right credentials the message in Figure 3.25 must be seen in the browser launched. This will conclude the appropriate configuration of big chain used by merge replication. More parameters can be checked upon opening the URL http://syncweb/ssce/sqlesa30.dll?diag.

![Image](http://syncweb/ssce/sqlesa30.dll)

Figure 3.25: Testing Setup

Finally, data synchronization sessions must be kicked off from Windows Mobile by setting properties and calling a method in the Replication object using the code in Figure 3.26. Compared with the XML Web Services replication method, the below code is negligible. Merge replication is all about servers...
configuration and agents installations. Coding part comes at last to play the role of requesting the data. Comments explain well the purpose of each line.

```csharp
try{
    _sync = new SqlCeReplication();
    // Assemble the connection string
    _sync.ConnectionString = "Data Source=\My Documents\MainDB2.sdf";
    // Set Publisher properties
    _sync.Publisher.SecurityMode = SecurityType.NTAuthentication;
    _sync.Publisher.Password = "P@ssword";
    _sync.Publisher.Database = "MainDB";
    _sync.Publisher.Publication = "MainDBPublication";
    // Set Web properties
    _sync.InternetUrl = "http://BEUWL00440090/ssa ssence/sq ces30.dll";
    _sync.InternetPassword = "P@ssword";
    // Set Distributor properties
    _sync.Distributor = "BEUWL00440090";
    _sync.Distributor.Password = "P@ssword";
    _sync.Distributor.SecurityMode = SecurityType.NTAuthentication;
    // Set Subscriber properties
    _sync.Subscriber = "Mohammad";
    _sync.HostName = 1;
    // Call replication methods
    _sync.AddSubscription(AddOption.CreateDatabase);
    _sync.Synchronize();
    MessageBox.Show("Database Synchronized");
} catch (SqlCeException sqlEx)
{
    MessageBox.Show(sqlEx.Message.ToString());
} finally
{
    _sync.Dispose();
}
```

Figure 3.26: Replication Code
Chapter 4
Web Services Replication

Networking notion nowadays is directly related to the Internet. There is an important reason why the Internet is spread worldwide to become a standard of communication replacing all traditional means. The reasons are very simple; it is the simplicity of the Internet protocol and its interoperability. Web pages are the most efficient mean to communicate globally. From a client's point of view, if you can type, you become available to millions of people worldwide. Technically speaking, web pages are set of get and posts methods in addition to the implementation of many mark-up languages. The idea of web services sprang from the concept that web can provide services in addition to providing information.

Web services are the newest generation of web applications (Turtschi, 2002). They are set of functions contained within namespaces that can be published and hosted online to be invoked in programs. The idea is very simple, so instead of invoking applications written in the same class or in different class on a local PC, now the programmer can invoke a function that is published on the Internet to avoid repeating already done work and speed up the process of development. Web services can be free or for a charge, the typical usage of these services is to provide global information and do some processing. For
example, a web developer can invoke a web service to get weather, stock, conversion, news information with minimal usage of bandwidth and diversity in platform access. Web Services are used as a middleware, so to speak.

One can say that there are many middleware programs already in use such as RMI, CORBA, and DCOM. Although in action, these middleware platforms do not provide perfect solution, so implementing them needs a compromise. The web platform comes to resolve this compromise issue since it is a global carrier for data and information with unified interfaces and implementations on all platforms.

4.1 Why Web Services?

The computer trend these days is about shifting applications from the traditional centralized monolithic systems to distributed ones, which are open and available to everyone. Web Services will support enterprise applications as the Internet did to millions of users, especially that they enable different systems to communicate in an easy way. The complexity of integrating different systems is very well-known, sometimes it is very complicated. Integrating lots of different systems is somehow impossible. SOAP and Web Services offer hope here, because that technique is simple, and because it is a universally accepted standard.
4.2 Web Services Platform

They are called XML Web Services, so they are based on XML in addition to HTTP. HTTP is needed since it is an omnipresent protocol, deployed on each computer and server connected to the Internet. XML offers a mean in which one can write particular languages to convey interactions between clients and services. After being processed by a web server, the XML messages are transformed to a middleware request and the results transformed back to XML. So the user receives XML formatted data that is described within tags. This sounds easy, but there are some requirements to realize this platform. To get it functional, we need the support of XML, HTTP, SOAP, WSDL, UDDI, and if needed XAML, XLANG, XKMS, and XFS services (Turtschi, 2002). The focus in this project will be on the first set of protocols briefed below, the later ones are not accepted in a platform independent way. Figure 4.1 explains the platform.

- SOAP (data exchange)
- DISCO (discovering web services)
- UDDI (publishing web services using directory service)
- WSDL (description language)
4.2.1 SOAP

SOAP stands for Simple Object Access Protocol. The goal of this protocol is to run over Internet, be easy to implement, and be based on XML. SOAP is an open Internet standard (Turtschi, 2002). SOAP is a protocol that lets two systems (client and server) exchange data. Although the SOAP protocol was written to be employed on a mixture of Internet transport protocols, it is most often used on top of HTTP.

SOAP supports two message models (Wenz, 2007). The first is one-way exchange, where a client sends out a request to a server, and will not accept an answer back. The second message model consists of a request-response mechanism. A client sends out an HTTP request to a server and the server replies by sending an HTTP response. SOAP affixes to that a standard way to forward data back and forth, taking into consideration as well a standard way to report errors to the client. In conventional Web applications, the only thing that is
standardized in a Web request is the URL, the HTTP requests (GET, and POST), and a few HTTP headers. The method of sending and receiving other types of data is in the hand of the programmer to specify. SOAP addresses the standardization challenges by defining a procedure to pass simple and structured data between clients and servers via a standard XML syntax, and a mechanism to call procedures running remotely on servers.

In brief, SOAP has two faces. It can be a remote procedure call (RPC) protocol implemented in distributed environment. On the other hand, it can be used as a standardized way to swap (XML) documents. With some fine tuning and with the usage of other protocols, web services can offer security, transaction management, and QOS delivery. The web.config file, which is rooted at the web service application, must contain the following directives to support SOAP, as shown in Figure 4.2.

```xml
<webServices>
    <protocols>
        <add name="HttpSoap12" /><!-- Change to HttpSoap10 for compatibility with other SOAP clients -->
        <add name="HttpSoap" />
        <add name="HttpGet" />
        <add name="HttpPost"/>
    </protocols>
</webServices>

Figure 4.2: Web.config File Protocols

4.2.2 WSDL

WSDL is an XML language that describes a web service in terms of structure and how to access its functions and pass the correct parameters (Turtschi, 2002). Using VS IDE one can generate the WSDL Web Service
description in different ways. One can get the WSDL description dynamically by calling the Web Service URL attached by the WSDL parameter; such as http://www.mohammadmahjoub.com/webservice/service.asmx? WSDL
One can also generate the WSDL description by using the disco.exe application that comes with VS. It gets the URL of the Web Service as an argument and creates the information into an XML file. Finally, one can programatically generate WSDL files by making use of the classes under the System.Web.Services.Description namespace.

Typing http://www.mohammadmahjoub.com/webservice/service.asmx?

WSDL will populate the text shown in Figure 4.3.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<wsd1:definitions xmlns:soap="http://schemas.xmlsoap.org/wsd1/soap/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:soap12="http://schemas.xmlsoap.org/wsd1/soap12/"
 xmlns:wsdl1="http://schemas.xmlsoap.org/wsdl/">
  <wsd1:types>
    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
      <s:element name="GetDataSet">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="id" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="GetDataSetResponse">
        <s:complexType>
          <s:sequence>
            <s:element minOccurs="0" maxOccurs="1" name="GetDataSetResult">
              <s:complexType>
                <s:element>
                  <s:sequence>
                    <s:complexType>
                      <s:element name="DataSet" nillable="true">
```
4.2.3 DISCO

DISCO is a procedure developed by Microsoft to locate Web Services (Turtschi, 2002). More specifically, DISCO guides clients to the WSDL files describing the call syntax of Web Services.

DISCO has two divisions. Files with the .vsdisco extension have information where to dynamically search for Web Services on the local server. Files with the .disco extension have information about existing Web Services on the server, particularly where the corresponding WSDL information is located. The problem with DISCO is its limited solution in that we need to provide the name of the server and the DISCO location on that server before we can query for Web Services. VS automatically maintain a file with extension .vsdisco to Web Service projects. It also puts a VSDISCO file into the root of the Web server.
4.2.4 UDDI

As discussed earlier, DISCO offers a non-practical way to search for web services. A more thorough way to locate Web Services exists, and it is called Universal Description, Discovery, and Integration (UDDI). UDDI is a Web Service itself, and it permits businesses and individuals to publish information about themselves and the Web Services they are offering (Turtschi, 2002). It is conceived as a global directory service, open to everybody, simple to use, and comprehensive in its scope.

VS supports UDDI through the possibility to query the UDDI directory and add references to Web Services into applications. Because UDDI is itself a Web Service, we can certainly do everything ourselves and interface with it by simply issuing SOAP requests and parsing the SOAP responses from the UDDI server for the information being sought. The way web services are offering synchronization approach in this project is referenced as explained earlier. Once it is referenced, it can be manipulated and its functions called to send and receive the information needed.

4.3 Code snippet

Figure 4.4 shows a sample C# script being implemented as a web service. It receives the patient’s id, fetches the related records and sends them back to the requestor in the form of a data set.
public DataSet GetDataSet(string id)
{
    DataSet myDataSet = new DataSet();
    try
    {
        OleDbConnection cnn = new OleDbConnection();
        cnn.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + Server.MapPath("./MainDB.mdb") + ";
        OleDbDataAdapter adapter = new OleDbDataAdapter("Select * From Patient, Records
        WHERE Patient.PatientID = Records.PatientID and Patient.PatientID = " + id + "", cnn);
        adapter.Fill(myDataSet,"Patient");
        adapter.Dispose();
    }
    catch (Exception ex) {}
}

return myDataSet;

Figure 4.4: Web Service Code

4.4 Comparing Replication Methods

![Diagram of replication methods]

Figure 4.5: Replication Approaches (SQL Server CE Portal, 2008)
When work on this project was initiated, it was a tedious mission to find replication and synchronization approaches adopted these days. And choosing a technology to go for was even harder since the cons and pros for each technology have to be weighted before adopting it in the project. Approaches were studied. The most important are merge replication, RDA, and web services. That is in addition to message passing and transactional approaches.

Using appropriate drivers, mobile devices can directly communicate with SQL Servers to fetch appropriate data. This technique can be used if the SQL Server and the mobile users are on the same site, but if they are distant, merge replication comes to edit this technique by introducing approaches for offline connections and data recovery and conflict resolution.

RDA is a variant of Merge Replication. But we have to write code to pull down each table and take care of data integrity (Wagner, 2004). Updates are not supported, so the table must be deleted and new data must be downloaded. This sounds not feasible at all, but it has its own environment to be deployed in.

It is obvious that there is a problem in the above two approaches. Merge replication needs many agents and configuration efforts, while RDA can not function well over slow links; moreover, we need to use same database drivers. Web Services come into play to offer a solution. It gives complete control over the server API and is the only way to communicate with other databases such as SQL Server, Access, Oracle, Informix, and DB2.
The down side is that one has to write thousands of coding lines to mimic the functionality of merge replication. Another disadvantage is the poor performance in transferring data. As can be seen, it is not a perfect solution that fits all environments and provides optimal performance. Table 4.1 presents a comparison in important networking and replication aspects between merge replication and web services.

Table 4.1: Comparing Merge and Web Services Replication

<table>
<thead>
<tr>
<th>Approach</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web Services</td>
<td>Platform Independent</td>
<td>Needs a live link</td>
</tr>
<tr>
<td>Merge Replication</td>
<td>Http based</td>
<td>Requires many agents to run</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Best Option</th>
<th>Feasible</th>
<th>Demanding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Crossing Firewalls</td>
<td>Web Services</td>
<td>Merge</td>
<td></td>
</tr>
<tr>
<td>Synchronization Conflicts</td>
<td>Merge</td>
<td>Web Services</td>
<td></td>
</tr>
<tr>
<td>Coding efforts</td>
<td>Merge</td>
<td>Web Services</td>
<td></td>
</tr>
<tr>
<td>Ease of deployment</td>
<td>Web Services</td>
<td>Merge</td>
<td></td>
</tr>
<tr>
<td>Corporate Environment</td>
<td>Merge</td>
<td>Web Services</td>
<td></td>
</tr>
<tr>
<td>Corporate Databases</td>
<td>Merge</td>
<td>Web Services</td>
<td></td>
</tr>
<tr>
<td>Heterogeneous DBMSs</td>
<td>Web Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mobile database creation</td>
<td>Merge</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Security for data in transit</td>
<td>Merge</td>
<td>Web Services</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 5
ASP.NET AJAX

ASP.NET AJAX is arising to be an accepted browser technology; AJAX brings the operations and user interface of web applications closer to that of desktop applications (Wenz, 2007). The main concept behind AJAX is to enable web pages to make HTTP requests in the background, or asynchronously, without reloading an entire page. Now web users can handle parts of the page and request some data or changes without refreshing the whole page and waiting. This is the trend in today’s web applications. Key success reasons for many famous web sites like Facebook, Google, and YouTube are ought to AJAX usage. AJAX also permits responsive UIs to be developed based on technologies like JavaScript, DOM, and CSS. Developing AJAX web applications requires understanding of JavaScript, DOM, and knowing how to deal with the XMLHttpRequest object.

AJAX-based applications can be written without an IDE like VS, but this is a very hard task to achieve. One of the goals of ASP.NET AJAX is to get rid of the need for writing lots of code and to provide the developers the same environment as in developing usual ASP.NET web applications. An important aspect of this integrated development is to provide JavaScript some of the functionalities of object oriented programming; this is realizable using libraries
that provide these advantages to the JavaScript/DOM/CSS developer. Browser compatibility layer permits ASP.NET AJAX scripts to run in many browsers and removes the need to customize and tests target browsers. AJAX offers JavaScript extensions that make object oriented scripting achievable in sense of providing classes, event handling and inheritance capabilities. The base class library offers a number of components, such as string builders and timers, which are found in the .NET framework. Script controls offers ASP.NET AJAX some HTML controls that are developed with functions as data binding, drag and drop.

AJAX is designed to provide functionality; the programmer need not know all the aspects of developing this technology to work with it. ASP.NET AJAX can administer AJAX functionality the same way that ASP.NET manages HTTP functionality. Since ASP.NET AJAX works as part of ASP.NET, it inherits all the benefits of developing in such environment. ASP.NET AJAX controls can interrelate with other ASP.NET controls and components and contribute in the page life cycle. It can be related to ASP.NET 2.0 features, such as sessions, authentication, and profiles.

AJAX is about two important controls: ScriptManager, and UpdatePanel. They are the core of each AJAX application (Darie & Brinzarea, 2007). In addition, AJAX provides other components similar to those found in ASP.NET that interact with AJAX client framework. Using VS 2005, AJAX support can be
added via installation of additional components. VS 2008 fully supports the AJAX platform without the need to install additional add-ons.

5.1 ASP.NET AJAX Packages

Many packages can be downloaded from the Internet as extensions to VS 2005 IDE. One package that has it all called ASP.NET AJAX Core Extensions is being used in this project.

5.1.1 ASP.NET AJAX Prerequisites and Installation

ASP.NET AJAX is very powerful. The major objects needed to work with such an environment are a web browser enabled with JavaScript and a web server, IIS for example. In addition, using a development environment with IntelliSense, debugging, and WYSIWYG functionalities will save a lot of time. VS Express edition supports all of these functionalities.

5.1.2 Installing ASP.NET AJAX

Using Microsoft VS environment, ASP.NET AJAX is incorporated into the IDE. Referring to the site http://ajax.asp.net, one can find a link to ASP.NET AJAX with a .msi installer called ASPAJAXExtSetup.msi. When installation is over, a new option for creating a web site ASP.NET AJAX Web Site will be enabled. The installer copies all the appropriate files and places them in the appropriate directories.
Typical web pages contain two parts, a request and a response. The whole page is refreshed upon the response part. AJAX-enabled web applications can constantly swap data with the server without the need to do a refresh (Wenz, 2007). Client script libraries ease the communication between browser and web server and make client coding easier through using the ScriptManager component. The code written in the ASP.NET AJAX server assembly handles XMLHttpRequest requests and employs many web controls. Figure 5.1 depicts the architecture behind ASP.NET AJAX and explains the request and data transfer.

![Figure 5.1: AJAX Architecture (Wenz, 2007)](image)

One important thing to note is that the ASP.NET AJAX client framework, bottom layer of the client side in the above Figure, is pushed to the browser from the server the first time an ASP.NET AJAX page is requested. Succeeding requests to the server from the same page can then be made with HTTP requests that send text and XML.
5.2 AJAX Code Snippet

Figure 5.2 show sample AJAX code implemented at the web version of the project. It is used to display personal information about a patient.

```xml
<ajaxToolkit:ScriptManager runat="Server" ID="ScriptManager1" />
<ajaxToolkit:TabContainer ID="TabContainer1" runat="server" ActiveTabIndex="2" Width="100%" Height="600px" ScrollBars="Auto">
    <ajaxToolkit:TabPanel ID="TabPanel1" runat="server" HeaderText="TabPanel1">
        <table width="100%">
            <tr>
                <td><asp:Label ID="Label7" runat="server" Text="<%# Eval("PatientID") %>"></td>
            </tr>
            <td width="20%"><asp:Label ID="Label1" runat="server" Text="<%# Eval("FName") %>"></td>
        </table>
        <ajaxToolkit:HoverMenuExtender ID="hme2" runat="Server"
            HoverCssClass="popupHover"
            PopupControlID="PopupMenu"
            PopupPosition="Left"
            TargetControlID="Panel9"
            PopDelay="25" />

Figure 5.2: Sample AJAX Code
Chapter 6

Databases

The core purpose of this project is to show the capability that web services provide in terms of deploying heterogeneous databases. Merge replication can only be deployed in one type of infrastructure that is based on SQL Servers whether on servers or mobile platforms. In this project, Microsoft Access is being used to save the medical records through the AJAX application, and SQL CE is used to store the records on the mobile phone. It is worth noting that SQL Server, Oracle, DB2, MySQL databases can be used as a replacement for the Microsoft Access database without affecting the functionality of the whole program. This is because the code uses core database functionality irrespective of the DBMS used. The Web Service connects to the DBMS with a specified parameter; it fetches the requested data by issuing an appropriate SELECT statement, and then returns the result in the form of a dataset to the user. These functions can be performed on any DBMS platform, as said previously. Microsoft Access is used for web hosting purposes.

The design of the database consists of three major tables (Patient, Records, and Authentication). The Patient table has the primary information of the patient such as name, sex, ID, and date of birth. The Records table contains the information in which the doctor enters upon the patient’s visit. These
information include the patient ID, date of visit, diagnosis, medications with respective timings used to alert the patient, data of next visit, weight, cholesterol, blood pressure, glucose, and a to do list. The third table has the authentication parameters for web login. The code connects to the database using OLEDB driver in order to insert, update, delete, and fetch data.

![Figure 6.1: Database Design](image)

On the mobile side, it is required to install .Net CF and SQL Server CE (Mobile). The program does not come with a ready database. After the program is launched, the user must activate the application by entering the appropriate patient ID. The next step is to initialize the local database. This step creates a mobile .sdf database that has the appropriate schema and table. Once the database is ready, the program becomes ready for web synchronization. Once synchronization is done, the database is populated with data to be used by the
program for archiving medical history, for initializing reminders and upcoming appointments, and for generating health graphs. It will be of no importance for the user to access the mobile database directly and query its data, but for the sake of this report a brief description is provided on how to access the database and handle queries. Tables from the database are shown in Figure 6.1.

After the initialization of the mobile database under My Documents, the user can click on the file named MainDB.sdf to get the screen shown in Figure 6.2, which a table named Patient and its schema definition.

![Figure 6.2: SQL CE Database](image)

Pressing on the next tab, SQL, the user can enter a text query, insert, update, or delete, and press the run button to perform the transactions, as in Figure 6.3.
The results of the transaction performed will be displayed in the third tab, the Grid, as shown Figure 6.4.

Figure 6.4: SQL CE – Result Set

The last tab, Notes, will give statistics about the operations performed. Whether successful or not, in addition to the time taken to perform the transactions, as shown in Figure 6.5.
Figure 6.5: SQL CE – Statistics Screen
Chapter 7
The Medical Application

7.1 Web version

The entry page for the web application can be found on www.mohammadmahjoub.com/webapplication/index.aspx. Upon issuing this URL, the login page displayed in Figure 7.1 will appear. The physician has to enter the appropriate credentials to access the medical records.

Figure 7.1: Web Application – Login Page
Correct credentials will refer the user directly to the below page displayed in Figure 7.2, else the user gets an error. Users without a session token and trying to access the home page directly will be redirected to the index page and denied access. Figure 7.2 is the main panel for accessing medical records. As can be seen, the page is tabbed. The first tab is Add Patient, the second tab is Edit Patient, the third tab is Medical Records, and the fourth tab is Appointments.

In the first tab, as shown in Figure 7.2, the physician can create an account for the patient by entering basic information such as first name, last name, gender, date of birth, and height. An automatic ID will be assigned to each patient incrementally.

Figure 7.2: Web Application – Main Page (Tab 1)
The second tab, as shown in Figure 7.3, displays all the accounts for all patients, and allows editing options such deleting and updating. The user has just to hover over any of the records to be prompted a panel that gives the editing options. The editing will be directly reflected the specific record.

![Figure 7.3: Web Application – Main Page (Tab 2)](image)

The third tab, as shown in Figure 7.4, gives the user the option of searching for a specific patient and displaying the medical history. Search can be based on patient's ID or patient's name. Upon pressing the search button, the dynamic grids placed in the accordions will be populated. The grids are formatted to display the information in a comprehensive way, and the accordion is a control that helps displaying information in a condensed way. Much known and used in
the popular Facebook site, it helps the user hides the unnecessary information. The history displayed includes the personal information in the first accordion pane and the detailed medical records in the second pane. Medical records include date of visit, diagnosis details, medication, and timings to take the medication, next appointment, weight, blood pressure, cholesterol, glucose, and advice list.

![Online Medical Records](image)

Figure 7.4: Web Application – Main Page (Tab 3)

The medical records, as detailed earlier, are entered through the fourth tab. As shown in Figure 7.5. On average, one or two medications are prescribed by the physician. Two medications are taken into consideration in this project, in addition to three time reminders per day since the maximum times a patient take a specific medication is three times per day. These parameters are taken to
represent a usual case. They can be easily upgraded and customized to reflect specific cases.

Figure 7.5: Web Application – Main Page (Tab 4)

7.2 Web Service

Web services are meant to work without a user interface; however they can be tested for functionality. The web service in this project can be accessed via www.mohammadmahjoub.com/webservice/service.asmx. The user will get the page depicted in Figure 7.6 in which a function named GetDataSet is displayed. This function is called by the PDA. The program in the PDA passes a
parameter which is the patient ID. Based on the patient’s ID, the medical records are fetched.

![Figure 7.6: Web Service – Main Page](image)

Upon pressing the below function, it’ll prompt to enter an ID, as shown in Figure 7.7. Entering the ID and pressing invoke will return all the medical records in the format of XML.

![Figure 7.7: Web Service – Function Page](image)
7.3 Mobile Version

Upon initializing the application on the PDA, the main screen shown in Figure 7.8 will open. This is the home page for the application. It has menus and submenus to use all the functionalities of the system. The first thing to do is to activate the application by entering the patient ID.

![Figure 7.8: Main Screen](image1)

![Figure 7.9: Activation Screen](image2)

After initializing the application, the patient needs to retrieve his/her medical records onto his/her PDA via a synchronization process. The Database menu has all the required functions to do this. As shown in Figure 7.10, it has four sub-menus.
The **Initialize Database** must be pressed first to initialize the schema of the mobile database. The user is prompted a message box to assure the successful initialization, as shown in Figure 7.11.

The synchronization is also done via a sub-menu button; the patient has just to click the **Synchronize Medical Records** sub-menu to perform this. The
patient will see the progress of the synchronization by a progress bar, as shown in Figure 7.12. And once the synchronization is completed successfully, the patient is prompted, as shown in Figure 7.13.

Doing lots of synchronization sessions will affect the size of the database, so a compact utility is needed. The patient can click on the Compact Local Database button to compress the database. As shown in Figure 7.14.
Pressing the *Medical History* button on the main page will display the screen shown in Figure 7.15. It shows the date of the visit, the diagnosis, the *To Do* list advised by the doctor. In the grid it displays the medications prescribed and the health parameters. The size of the medical records depends on how much the PDA can handle storage.
Pressing the *Upcoming Appointments* button will inform the patient if he has a near appointment with the exact date, or not. It will also give the patient an option to set a reminder on this date, so that an alarm will be fired when the appointment time is to arrive, as shown in Figure 7.16.

![Figure 7.16: Upcoming Appointments](image)

![Figure 7.17: Appointments Reminder](image)

Pressing the *Graphs* button will display the screen shown in Figure 7.18.

![Figure 7.18: Graphs Form](image)
Pressing each of the buttons (Cholesterol, Glucose, Weight, and Blood Pressure) will display the related health parameters in a chart, as depicted in Figure 7.19 and Figure 7.20. Upon pressing any button, the patient can see a chart showing the indicators of the health parameters for the last five visits. All the buttons function the same way. The charting component is designed to take into consideration integer, float, and ratio numbers to display the scientific units of each measurement in mmHg, Kg, mmol/L, mg/Dl.

Figure 7.19: Blood Pressure Chart

Figure 7.20: Cholesterol Chart
An important health indicator is the Body Mass Index. The patient can check his BMI by opening the related form and entering his weight and height, as shown in Figure 7.23. The program will do the proper calculations based on a well known function and display the status to the patient by giving the BMI indicator and an advice on the weight, whether normal, under normal, or obese, as shown in Figure 7.24.
When the patient synchronizes the medical records, the program automatically sets reminders for medication timings as prescribed by the doctor. For example, when the doctor prescribes a certain medication, he/she adds the times in which it is advisable to be taken by the patient. Once the patient synchronizes the medical records, the program automatically reminds the patient by firing an alarm at the appropriate time. This is set only for the most recent visit. The reminders will be working throughout the lifecycle of the application, as shown in Figure 7.25; and the patient can turn them off by pressing the *Disable Reminders* button as shown in Figure 7.26.
Figure 7.25: Medication Time Reminder

Figure 7.26: Disabling Reminders
Chapter 8
Conclusion and Future Work

This report is a technical summarization for the efforts done for designing, coding, and implementing the Medical PDA application. The main goal is to automate the prescription process and enable patients to carry their medical records wherever they go. This sounds trivial in many commercial applications, but Medical PDA extends those specifications to include alerts for taking medications and following up on upcoming appointments, in addition to charting important factors in medical records. The application is 4-tiers, deploying client (Web or PDA), XML Web Services, Web Server, and a Database server. It is simple to install the tiers and get the application ready to function. The application takes into consideration general medical practices performed by a general practitioner. More functions and components can be included to reflect other specialties. It can include many parameters to be customized for each specialty. Alcohol, allergy, cardio vascular, respiratory, digestive, neurology inputs, radiography, scanner images and many other parameters can be added as fields in the database to be reflected upon synchronization.

The goal of this project is to show the usefulness of technology in the medical field and the benefits of its applications. Further work can dig more into expanding the capabilities of this project by going beyond the parameters and customizing the application to fit any medical specialty.
References


Microsoft®. MSDN Code Gallery is a portal offering tutorials and code for Pocket PC. http://code.msdn.microsoft.com/default.aspx?SiteEntry=gdn


85
Wikipedia, the world's first online free encyclopedia.  
http://www.wikipedia.com