2008

Implementation of active collections framework using .NET

Sushil Magdum

Project Report
Submitted to the Faculty of the
Rochester Institute of Technology
In partial fulfillment of the requirements for the
Degree of Master of Science in
Computer Science
January 28, 2008
Version 1.6

Implementation Of
Active Collections Framework Using .NET

Sushil Magdum
{sdm0903@rit.edu}

Signature Block

Approved By:

1. **Committee Chairman: _____ Prof. R. K. Raj**

2. **Committee Reader: _____Prof. James Heliotis**

3. **Committee Observer: _____TBD**

Acknowledgements

I want to thank a number of individuals at Rochester Institute of Technology for helping me finish the project successfully. My sincere thanks are extended to:

1. Prof. R. K. Raj, for taking the time to answer my questions related to the project and helping me in all aspects during all the phases of my project.
2. Prof. James Heliotis, who agreed to be the reader of this project.
3. Finally, I want to thank my friends at Rochester Institute of Technology for their trust and support.

Abstract

For many years, large distributed enterprises have faced a common problem of near real time sharing of enterprise data typically stored in databases. As these databases may be located globally, distributed nature of this data makes it difficult to access instantaneously. The Active Collections Framework (ACF) acts as a good foundation on which to build distributed applications. This framework requires distributed applications to view changes to data as events of interest. The ACF framework integrates access to data and data changes through active collections.

ACF framework is based on two different research areas: event management in distributed computing and active database systems. Active databases support mechanisms to monitor changes to the database state. The central concept in ACF event management is active collections. Each active collection is a collection of all objects specified by a query on the enterprise data. For each client interested in obtaining data, an entry is made in the active collection. This information is then used by windows service to notify the registered client of any data changes.

This project implements the Active Collections Framework using Microsoft .Net and Visual Studio .Net. Two sample applications using the developed framework have been developed to demonstrate the efficiency of data storage and event notification capabilities of the developed ACF framework.

Table of Contents

1. Introduction

1.1 Background on Domain

For many years, large distributed enterprises have faced a common problem of sharing the enterprise data in near real time [8]. The sharable enterprise data is typically stored in the databases. As these databases may be located globally, the distributed nature of this data makes it difficult to access it instantaneously.

There are different platforms that allow developers to implement enterprise applications on top of distributed databases. The two well-known platforms that facilitate development of these distributed enterprise applications are Microsoft's .NET Platform and Sun Microsystems' J2EE platform (Java 2 Platform, Enterprise Edition). These platforms provide the class libraries for developers to implement the distributed applications

Regardless of the platform selected by the organization, the way in which data is fetched from the database affects the accuracy of the data displayed on the clients. In the current scenario, all enterprise application's clients have to keep making requests to some centralized server in order to get the latest data. Hence the client does not have the up-to-date information until it asks for it.

Active Collections Framework is the proposed mechanism to solve this problem. This mechanism can act as a good foundation to build distributed applications. The framework requires the distributed applications to view changes to the data as events of interest. The ACF framework integrates access to the data and the data changes through the active collections. Active collections are based on two different research areas: event management in distributed computing and active database systems. Both these approaches are elaborately discussed in sections 1.3.1 and 1.3.2

1.2 Problem Definition

The current enterprise architectures do not provide full integration of the data access and data changes. Clients need to keep on requesting the changed data from some centralized servers and thus, updated information is not sent to the clients immediately. The aim of this project is to implement an Active Collections Framework that integrates access to the data and the data changes through active collections. The framework developed should be generic and can be used by different clients by implementing the libraries provided by the framework and get the up-to-date information of the changed data.

1.3 Approaches to the Solution

The two major research areas in propagating the data changes to the clients automatically are Active Collections Framework and Active Database Systems [6].

1.3.1 Active Collections Framework

Active Collections Framework was the proposed mechanism to solve the issue of updating the client with the up-to-date information of the changed data. The proposed mechanism used the concept called Active Collections to integrate data access with the data change. The central concepts of the proposed Active Collections Framework are Active Collections, the ACF object model, and the Active Collections Framework Services. Similar to other frameworks, the Active Collections Framework is also object oriented to facilitate the development of distributed applications.

The data stored in the database is represented by an ACF object. According to the proposed framework the data attributes of the ACF object corresponds to a single row in a relational database. Any behavior can be added to the ACF object through the methods that operate on object's data. Using the methods of the ACF, the ACF object can interact with the framework to modify the values for its data properties.

Active Collections is the main mechanism provided by the ACF to integrate access to the data and data changes. It is the collection of all the objects specified by all queries on the enterprise

data. If an object satisfies a query on the enterprise data, it will be added to the collection. In a similar way, if an object does not satisfy the query on the enterprise data, it will be removed from the collection. The ACF services continuously monitor the data changes and if needed, it can also update the collection with new objects.

Figure 1 represents the logical architecture of the proposed Active Collections Framework.

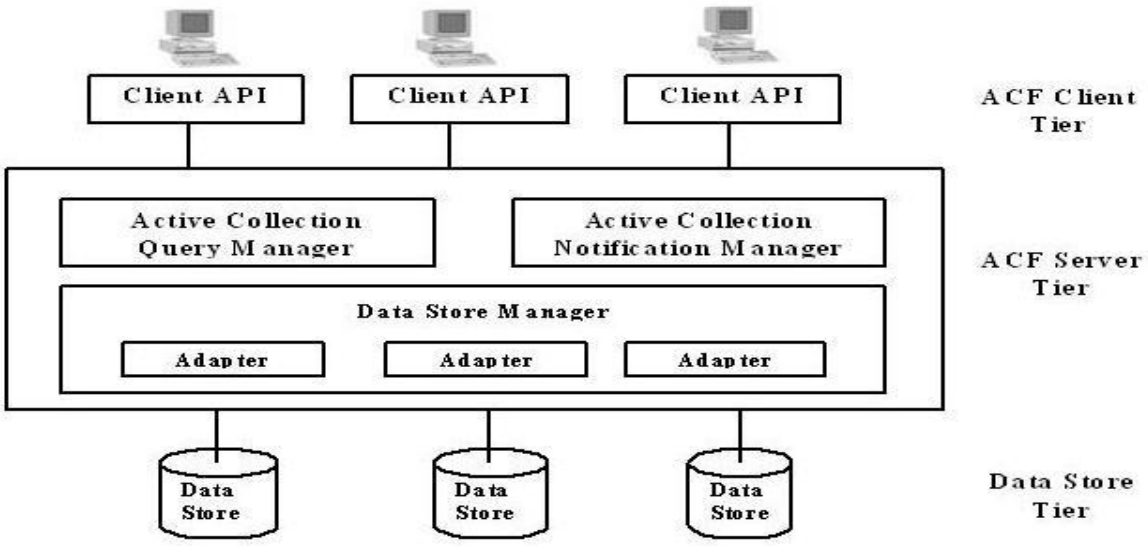


Figure1: Logical Architecture of ACF [1].

The ACF Server tier is the middle tier that provides different API's to manipulate Active Collections. The ACF Client tier interacts with the ACF Server tier using these API's. The Data Store tier is the underlying database like the relational database, object oriented database, xml database or simply flat files.

There is an existing implementation of Active Collections Framework by Bhaskar [1]. This framework was implemented using a Struts framework which is a variation of the MVC (Model-View-Controller) pattern. The approach taken in this project was to notify the middle tier if the data is changed in the database. The web client then pulls the data from the middle tier. However, the implementation was not a push mechanism. It was a push and a pull mechanism.

The advantage of this approach is that the middle tier has the exact changed information and hence the time efficiency of the application is increased.

There are various ways to implement the Active Collections Framework. However, the main aim of the framework is to provide up-to-date information to the client. Also, the framework should be developed in such a way that any client can use those libraries to get the real-time information.

1.3.2 Active Database Systems

Generally database systems are considered repositories to store the data or the information required by various applications. The user program manipulates the data by adding, deleting or modifying the data using the interface provided by the database system. Originally, database systems were not supposed to respond to any data change events, but gradually the responsibilities of the database systems started shifting. An active database system is a mechanism that allows the database system to respond to the data change events either inside or outside the database [3]. Active database systems also support the push mechanism rather than the traditional pull mechanism. As an example, if a person is interested in the stocks of a company, the application developed to keep a watch on the stock prices would have to continuously pull the data from the database. In this scenario if the database is updated by another application, the user program will not receive the updated value (price) of the stock until the next pull operation. The active database system provides a knowledge model and an execution model for supporting the push model and hence it can monitor and react to certain circumstances.

The knowledge model consists of three major components: an event, a condition, and an action. For example, events can be considered as a data update. The condition component checks the context in which the event occurred. The Action component describes the action to be taken if the condition is satisfied. As research on active databases is still in progress, current leading database system providers do not provide full implementation of active database systems.

1.4 Hypothesis

Statement:

If the application requires up-to-date sharing of enterprise data without the user explicitly asking for it, then the use of the proposed Active Collection Framework may help them to achieve it, thereby reducing the network traffic and database overheads.

In current enterprise architectures, the client has to keep requesting the changed data from some centralized servers. Hence the client does not get the up-to-date information of the changed data. The active collections framework is a proposed mechanism to solve this problem.

This project implements a generic active collections framework using Microsoft .NET framework and Visual Studio .NET. Two sample applications were built on this framework that demonstrates the efficiency of the data storage and event notification capabilities of the developed ACF framework.

1.5 Roadmap

Section 1 of the report explains the background of the problems encountered in distributed architectures. It also explains the related work to solve this problem. Finally, it explains the proposed active collections framework and its logical architecture. The hypothesis of the project is stated in the first section. Section 2 presents the architecture and design of the active collections framework. The flow chart depicted in this section represents the bird's eye view of different processes in the ACF framework. Section 2 also presents the proposed deliverables of the project. Section 3 explains the implementation details of the project. Section 4 presents the analysis of the project. Section 5 concludes the report.

2. Design

2.1. Architecture

For the implementation of the active collections framework, the platform selected is Microsoft's .NET Framework. The .NET framework is a managed code programming model that has a large set of pre-coded solutions for common programming requirements. Microsoft also provides Visual Studio which is a software development product for software programmers. It facilitates the rapid building of web applications, stand-alone applications, web services, etc. These applications run on any platforms that are supported by the .NET Framework. Visual Studio is an integrated development environment that will help in the implementation of the active collections framework. Visual Studio includes languages like, C#, Visual Basic, Visual C++, ASP.NET. For the implementation of this project, the C# programming language that targets the .NET platform will be used.

For this project, the database selected is Microsoft SQL Server which is a fully relational database management system. The reason for selecting MS SQL Server is that Microsoft and other vendors provide many software development tools that facilitate development of different applications using MS SQL Server. Microsoft SQL Server also includes the common language runtime component for Microsoft .NET.

The architecture of this proposed active collections framework is based on the architecture mentioned in the paper "Active Collections Framework", also described in section 1.3.1. The paper describes three-tier architecture. The ACF framework provides libraries that are used by the ACF client to update the ACF collections created on the ACF server tier.

The ACF server now has an added windows service. The windows service is used to maintain the active collection on the server. For the implementation of the ACF, the windows service registered on the server is responsible for maintaining this ACF collection list. This service will then pass the details to the active collection query manager. For the communication between the client and the server, Microsoft's .NET Remoting service is used.

Figure 2 depicts the logical architecture of the proposed ACF. It is an extension of the logical architecture depicted in Figure 1.

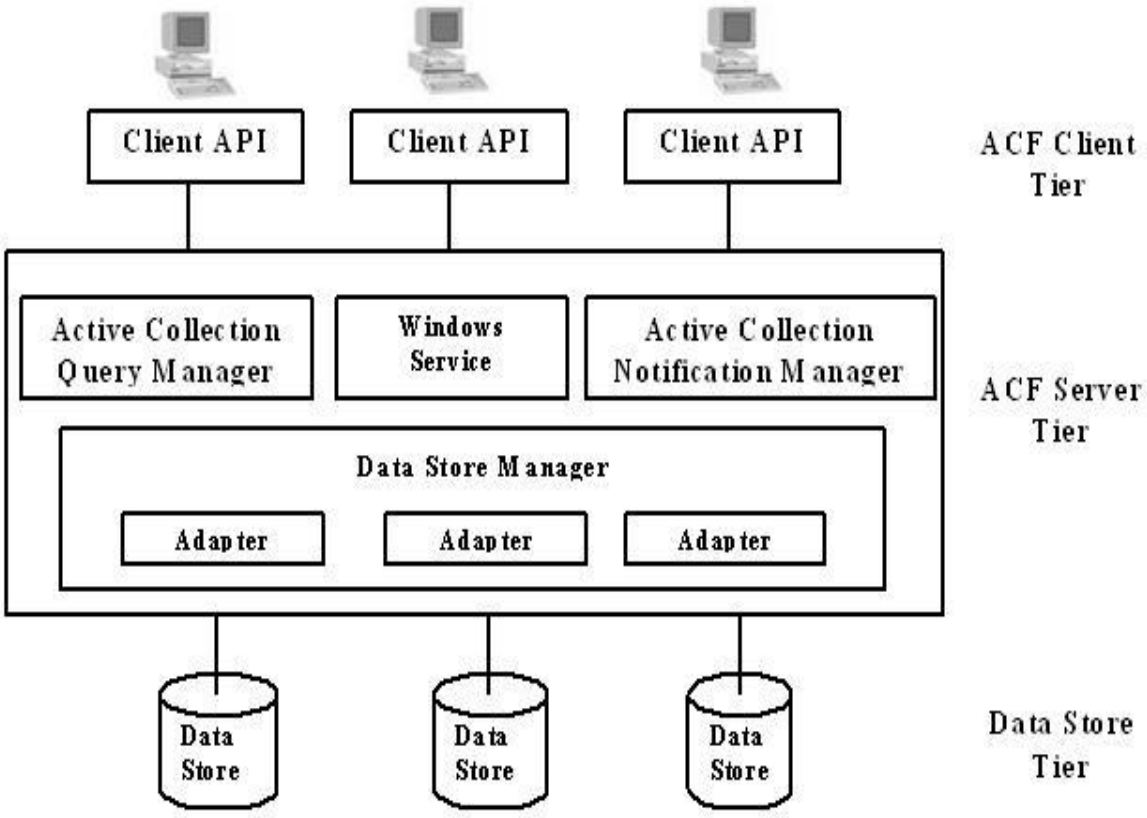


Figure2: Logical Architecture of Proposed ACF.

Figure 3 depicts the overview/flowchart of different processes in the ACF Framework.

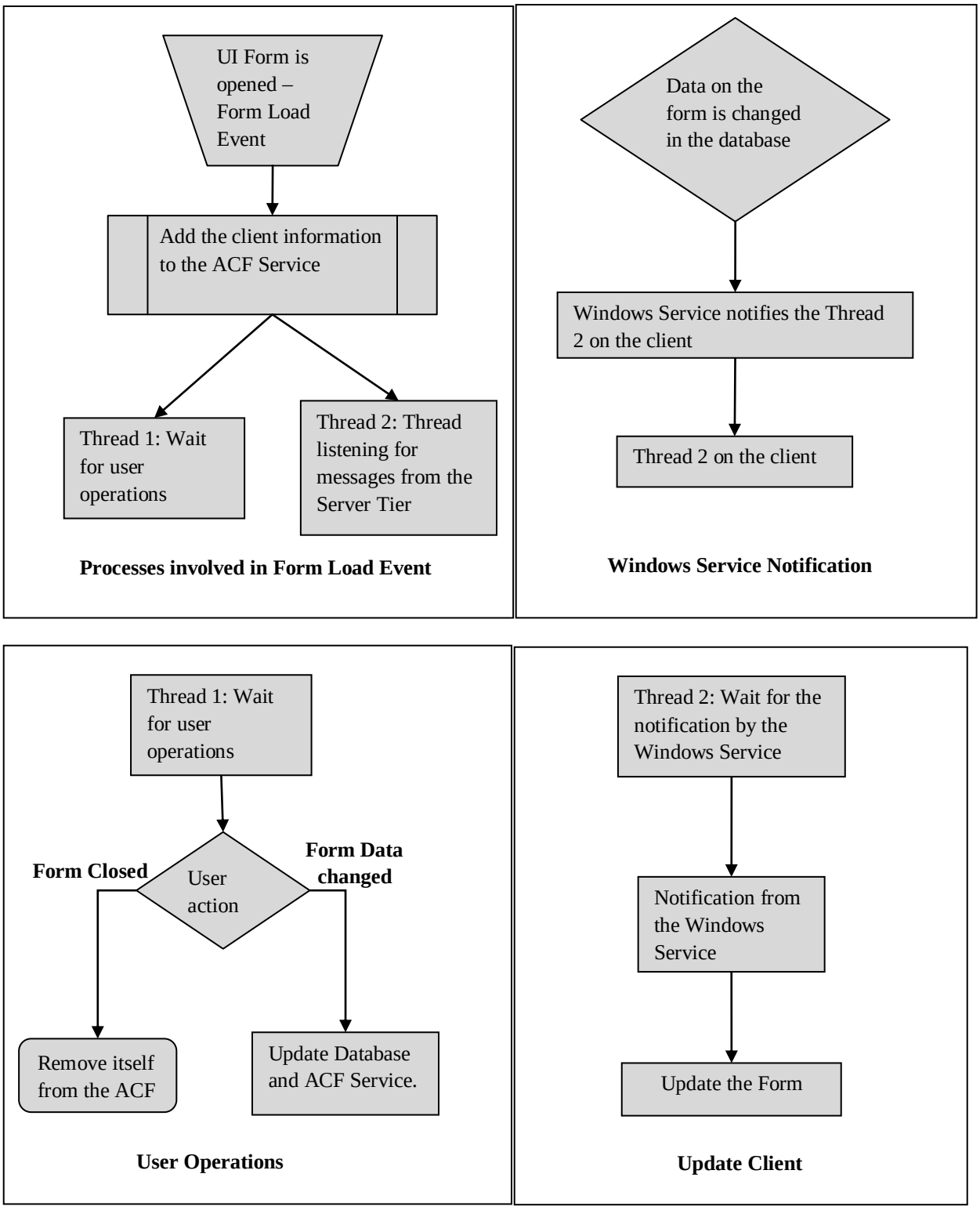


Figure 3: Overview of different processes in the ACF Framework

The walkthrough of the above flowchart is as follows. If a new form is opened or loaded, it adds its information like the table name, form name and other system information to the ACF list. At the same time one listening process is initialized. It will be notified by the ACF server tier if the data is updated. If the database is updated then the *isTableUpdated* property of the corresponding forms registered to the ACF service is set. The stop condition occurs when the form is closed. If the form is closed, then its reference is removed from the ACF list.

2.2. Evaluation Criterion

For the analysis of the implemented active collections framework, I have implemented two different applications. These applications will use the libraries provided by the developed ACF. The implementation will be evaluated based on the following criteria:

- Correctness of the framework with respect to immediate updating of the forms whenever the data is changed in the database;
- Check if the framework implemented is generic by evaluating the amount of hard coding and configurations required for using the developed framework.

3. Implementation Details

3.1 Software Detail

The main aim of the project is to provide an auto refresh mechanism. That is, the end user should always see the screen with the latest database record values without the need for an explicit refresh. This aim is achieved by implementing the Active Collection Framework using active collections. This mechanism has been implemented using the Observer design pattern along with the various .NET features like Remoting, Threads, Windows Service, Inheritance and Virtual Classes.

The Active Collection Framework resides in a customized .NET windows service and primarily maintains the active collection. It has functions like adding the data to the active collection, removing the data from the collection and sending notifications to the client whenever a data change takes place, thus allowing the users to see the updated data automatically.

The active collection maintains information of all the clients who are accessing the application. This data includes machine IP address, form name, table accessed by the form and a boolean variable *isTableUpdated*. This information is form level information and has to be sent to all client forms when they are initially loaded.

Each of these fields is explained below.

- **IP Address:** The address of the client machine on which the application is running. IP address acts as a primary key to differentiate between the same forms opened on various machines.
- **Form Name:** The name of the form being viewed by the user.
- **Table Name:** The name of the table being accessed by the form.
- **IsTableUpdated:** Boolean flag that determines if the table is updated in the database or not. Whenever a new entry is added into the list, this boolean flag goes in with a false value. When a table is changed in the database, the boolean flags in all the entries in the ACF list that are accessing this recently changed table are marked true.

The windows service starts on the server startup. As soon as the service starts, it registers itself using a TCP channel in a singleton mode. The singleton mode creates a single instance of the active collection object which is shared among all the clients connected to the server. When the service is stopped on the server, it unregisters itself and the connection to all the clients is lost. Hence it is necessary that the service is always kept running on the server as the core Active Collection Framework resides here. When a windows service dies for a while, the exception which is raised is handled by the client code and an appropriate message is displayed to the user. As soon as the service restarts, the client re-establishes the connection and performs normal operations.

On the client side, once the application starts on a client machine, it establishes a TCP connection with the server hosting the windows service which maintains the Active Collection Framework. This is done using .NET Remoting. When a form opens on a client machine, the form level information like the IP address, name of the opened form and name of the table the form access is added to the active collection. When a user on another client machine makes any changes in the database, the ACF list updates the flag of all the forms accessing this table to true. An update message is immediately sent by the server to the client machines. As soon as the application running on the client machines receives this message, it invokes the *RefreshMe* function of all the open forms accessing this table. Thus the screen is automatically refreshed and the user can see the latest database information without explicitly refreshing the screen. The form level information stored in the active collections is deleted as soon as the form window is closed on the client machine.

3.2 Class Structure

The project is implemented using three-tier architecture. It includes a UI layer, a business layer and a database layer. Each layer is explained as below.

3.2.1 UI Layer

The UI layer is represented by the ClientAPI project. It mainly contains the client forms which are application dependent. However it has two forms which are application independent. Those are the MainForm and the BaseForm which are explained below.

MainForm: Contains all the functions that are to be performed when an application is loaded. Among others, it performs the following functions:-

- Establishing a connection with the server.
- Initializing a thread in the background to get the messages from the server.
- Initializing the form level information.

The class diagram of the main form is depicted below in figure 4

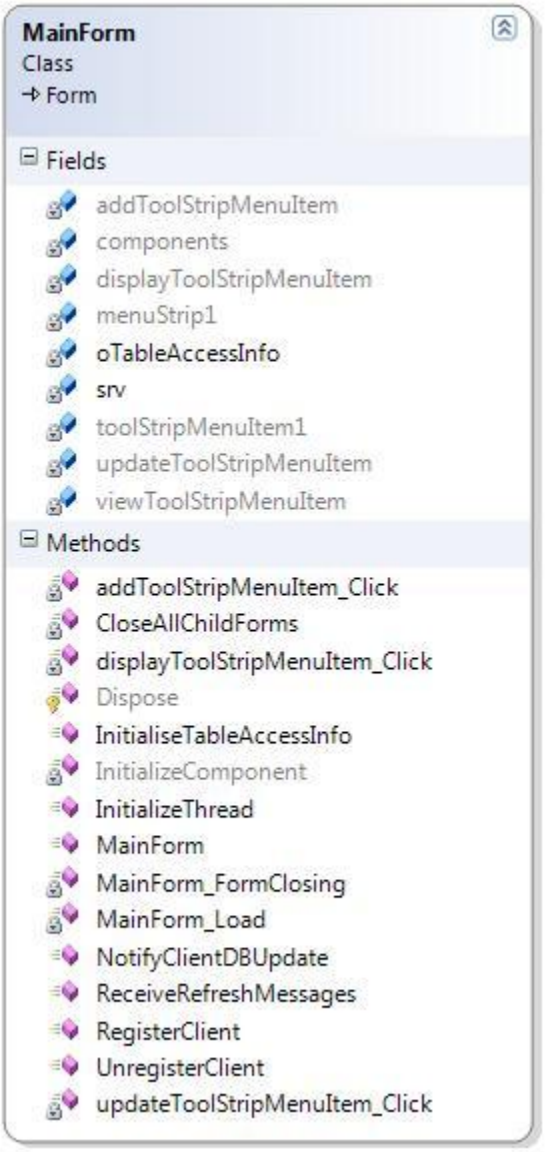


Figure 4: Class Diagram of MainForm

BaseForm: This form acts as a base form for all the application dependent client forms. All the client forms are inherited from this base form class. It contains a virtual function called *RefreshMe*. This function should be implemented in each of the inherited forms. The function contains the code to load the data from the database and display it on the page. When the thread, running in the background, gets the table update information, instance of each form accessing the updated table are acquired and their *RefreshMe* functions are invoked. Thus the auto refresh mechanism is achieved.

The class diagram of BaseForm and the Inheritance structure is depicted below in figure 5

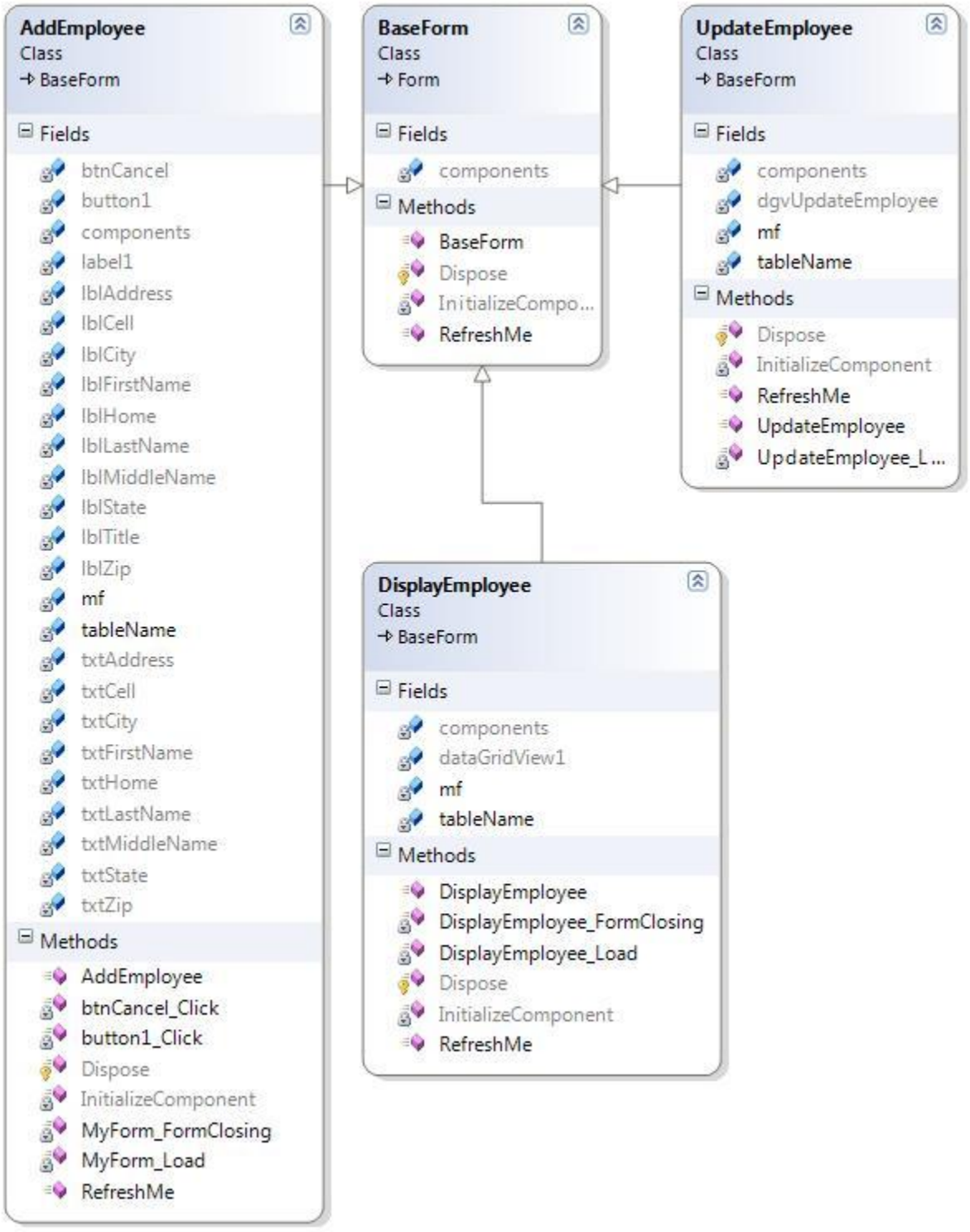


Figure 5: Class Diagram of BaseForm and the Inheritance Structure

3.2.2 Business Layer

This is the main layer in which the services that maintain the Active Collection Framework reside. This layer is common across all types of applications like Employee Information System, Company Stock Quotes etc. This is the core layer of the architecture. There are three project modules that constitute the Business layer.

3.2.2.1 ACLManager

ACL in the ACLManager stands for Active Collections List. This project contains a class having functions related to managing the ACF list. It has following functions.

- Add entries to the ACF list.
- Remove entries from the ACF list.
- Update the flag to true in all the entries in the ACF list when a table is updated in the database.
- Notify clients about a table update.

It also declares a structure *TableAccessInfo* to hold the form level information.

The client establishes the connection with the windows service on startup. Each client form has to invoke following methods of ACL Manager in order to be compatible with it.

Form Load -

1. Initialize the member variable of *TableAccessInfo* struct
2. Register itself with ACF list.

FormUnLoad –

1. Unregister itself with ACF list

The class diagram for the ACLManager is depicted in figure 6 below.

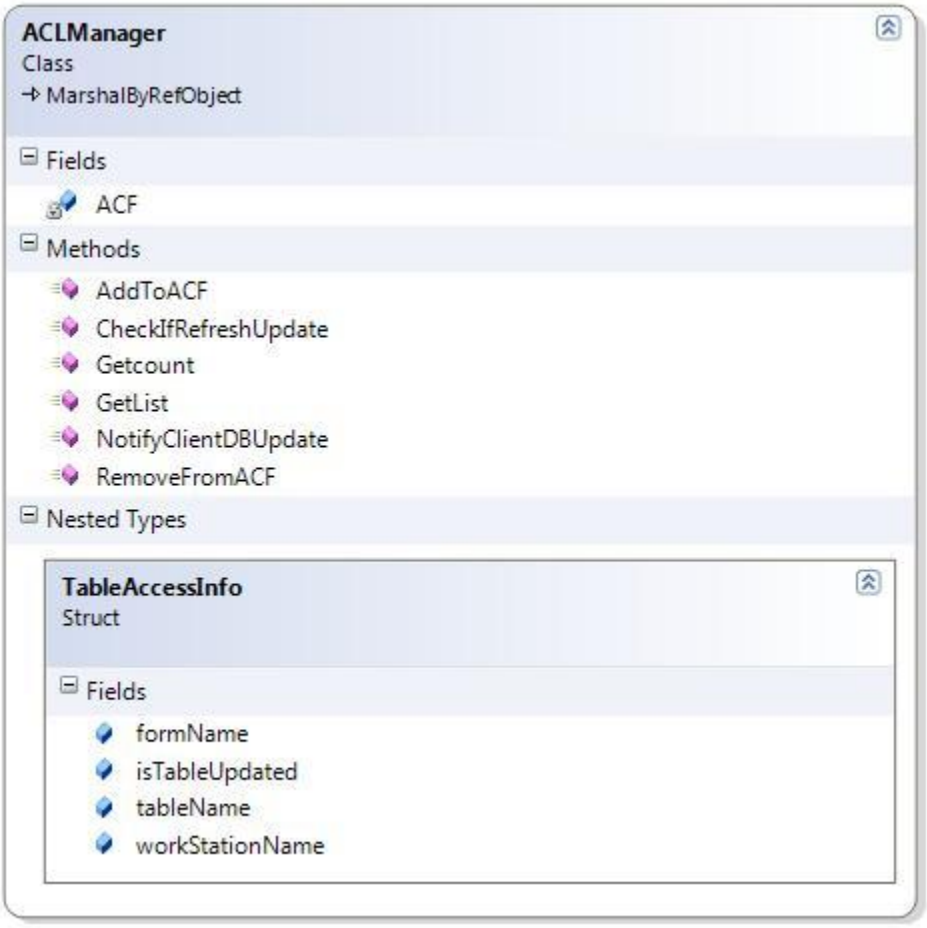


Figure 6: Class Diagram of ACLManager

3.2.2.2. ACLService

In this project the windows service is used to manage the active collection. It contains two classes explained below in detail.

ACLService: This class initializes the service properties like *ServiceName*, *CanStop*, *CanPauseAndContinue*, *AutoLog* etc. It defines what the service should do on start up and shutdown.

On Startup, the service registers *WellKnownServiceType* with ACLManager class mentioned in the point 3.2.2.1 on a TCP channel with *WellKnownObjectMode* as a singleton. Singleton creates single instance of the ACL object that is shared by all the applications.

OnStop service unregisters the channels to which it has registered.

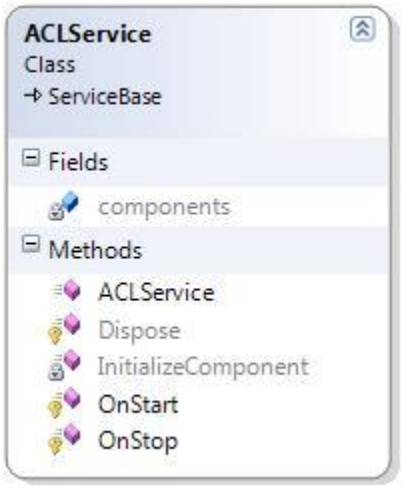


Figure 7: Class diagram of ACL Service

ACFControllerInstaller: This class initializes a Service Account Information and a Service Information.

- **Service Account Information:** Includes initialization of the account name, username and the password. It is used for authentication purposes.
- **Service Information:** Includes initialization of the DisplayName, StartType and the ServiceName. The StartType is set to automatic which means that the service starts automatically at the server startup. The service name is identical to the windows service which is set in the constructor of ACLService.cs i.e. ACLController

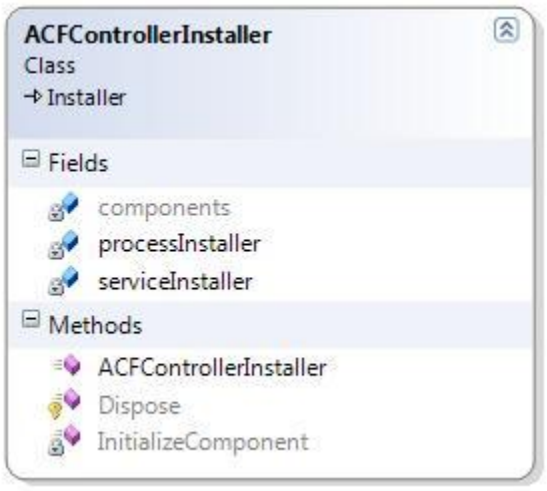


Figure 8: Class diagram for ACFControllerInstaller

3.2.2.3 DataManager

The main purpose of the DataManager class is to carry out database related functions. It contains generic database functionalities like inserting, updating, retrieving data from the database. It also contains functions related to establishing connection with the SQL Server on a remote desktop.

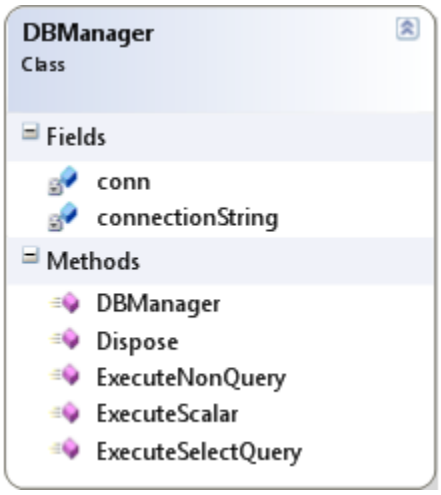


Figure 9: Class diagram for DataManager

4. Project Analysis

4.1 ACF Framework Code Analysis

The code analysis for this project is performed based on the following three metrics:

1. Class Coupling
2. Cyclomatic Complexity
3. Maintainability Index

4.1.1 Class Coupling

Class coupling is the degree to which an item relies on another item. When computing class coupling, the primitive and built-in types like string, object are excluded. The higher number indicates that if there is change in one type, more likely it ripples through to the other types.

4.1.2 Cyclomatic Complexity

Cyclomatic complexity is a software metric used to measure the complexity of a program. It measures the amount of decision logic in a module. The control flow of a module is represented in the form of graph and this graph is used to compute the cyclomatic complexity of that module. The cyclomatic complexity is defined as follows [11]:

$M = E – N + 2P$

Where,

M = Cyclomatic Complexity

E = the number of edges of the graph

N = the number of nodes of the graph

P = the number of connected components

For this project, the cyclomatic complexity is calculated by counting the number of loops in the flow graph. Cyclomatic complexity is calculated only for the main programs in the ACF framework.

4.1.3 Maintainability Index [5]

Maintainability is an index from 0 to 100 that indicates the overall maintainability of the member or type. Some other metrics like the lines of code, cyclomatic complexity, halstead volume, etc decide the maintainability index. The halsted volume factors in the number and use of operands and operator. The lower index means that the code is hard to maintain. Visual Studio provides the maintainability index for each member. The table below explains the level of maintainability and its corresponding maintainability index range.

Level Of Maintainability	Range Of Maintainability Index
Highly Maintainable	Between 20 and 100 (Inclusive)
Moderately Maintainable	Between 10 and 19 (Inclusive)
Least Maintainable	Between 0 and 9 (Inclusive)

Table 1: Maintainability metric

The code analysis based on above matrices is as follows:

Library Name: ACLManager

Name Space: ACLManager

Type	Member	Maintainability Index	Class Coupling
ACLManager	ACLManager()	100	1
ACLManager	AddToACF(params) : void	95	2
ACLManager	CheckIfRefreshUpdate()	59	4
ACLManager	Getcount() : int	87	2
ACLManager	GetList() : List	91	2
ACLManager	NotifyClientDBUpdate(string)	58	4
ACLManager	RemoveFromACF()	94	2

Table 2: Code analysis for ACLManager

Library Name: ACL Service

Name Space: ACFController

Type	Member	Maintainability Index	Class Coupling
ACFControllerInstaller	ACFControllerInstaller()	61	5
ACFControllerInstaller	Dispose(bool) : void	80	3
ACFControllerInstaller	InitializeComponent() : void	94	2
ACLService		74	11
ACLService	ACLService()	70	2
ACLService	Dispose(bool) : void	80	3
ACLService	InitializeComponent() : void	85	3
ACLService	OnStart(string[]) : void	71	8
ACLService	OnStop() : void	77	4
Program		84	2
Program	Main() : void	84	2

Table 3: Code analysis for ACL Service

Library Name: DataManager

Namespace: DataManager.

Type	Member	Maintainability Index	Class Coupling
DBManager	DBManager()	80	1
DBManager	Dispose()	98	1
DBManager	ExecuteNonQuery(string)	82	2
DBManager	ExecuteScalar()	100	0
DBManager	ExecuteSelectQuery(string)	75	3

Table 4: Code analysis for DataManager

The code analysis is performed only on the ACF framework and it does not include the client programs. The cyclomatic complexity of each module is less than 50. The closer the maintainability index gets to 100, maintenance of the code gets better. The framework gave good results with the average maintainability index of 82 thus making it easily maintainable.

The lines of code for this project is: 3274

The project is evaluated based on correctness of the framework with respect to immediate updates of the forms whenever the data is changed in the database. To test the correctness, I have implemented two client applications. Both these client applications use the libraries provided by the developed ACF framework. The applications are described as follows:

- Employee Management System
- Stock Quotes

4.2 Sample Applications

Employee Management System and Company Stock Quotes follow 3 tier architecture, having client, business and database layer. The ACF framework developed is an independent piece of code which can be compiled into a dll. The framework is implemented in these applications by adding a reference of this dll in the client project and following the basic steps, mentioned in the section 3.2.2.1, to be compatible with the framework code.

These two sample applications are explained below in detail in terms of their architecture and database schema.

4.2.1 Employee Management System

The Employee Management System manages an employee's personal information. This application can be used by the company management staff as well as the administrators. The management part of the application displays the list of the details of all the employees of the company. The administrator part of the application allows the administrators to add/update/delete employees in the database.

In order to test the framework all the scenarios were tested where the management is looking at the list of employees and at the same time the administrator updates the information of the employees. The expected behavior of the application is that, at the moment when the administrator updates the employee's information, the updated data should be visible to the management. The management should not manually refresh the display page to get the updated data but the updated data should get propagated to the display page automatically. All the test cases passed successfully. The management gets the updated data by the administrator immediately. The class diagram of the Employee Management System is follows:

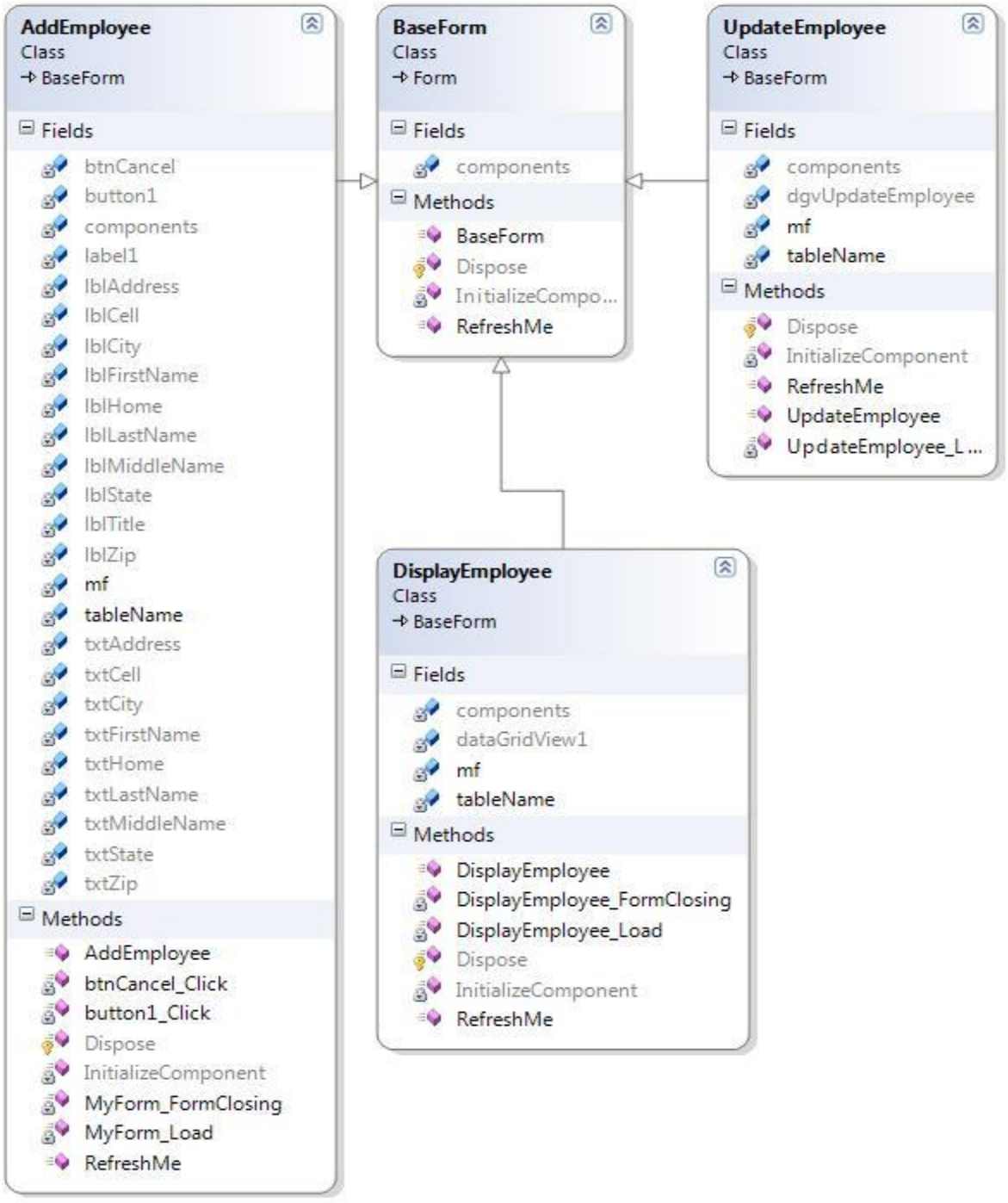


Figure 10: Class diagram of Employee Management System

The database schema for the Employee Management System is as follows.

Attribute	Data Type
ID	int
FIRSTNAME	varchar(50)
MIDDLENAME	varchar(50)
LASTNAME	varchar(50)
ADDRESS	varchar(50)
CITY	varchar(50)
STATE	varchar(50)
ZIP	Int
CELL	varchar(50)
HOME	varchar(50)

Table 5: Database schema for the Employee Management System

In the above schema, ID is the primary key of the table. ID is generated automatically through the code whenever a new employee is added to the system. The ID, FIRSTNAME and LASTNAME attributes do not allow null values. All other attributes allow null values.

4.2.2 Company Stock Quotes

The Company Stock quotes application notifies the users about the stock price updates. The users of this application keep a watch on some selected company quotes. The company stock prices can be modified by any other application.

For testing the framework, I developed this second application to see if the same ACF framework libraries can be used without any major change. The scenarios like adding new company, updating the stock price of a company, etc. were tested. The test case passed successfully. Whenever any stock price of a company is updated, the users are immediately notified about the updated price.

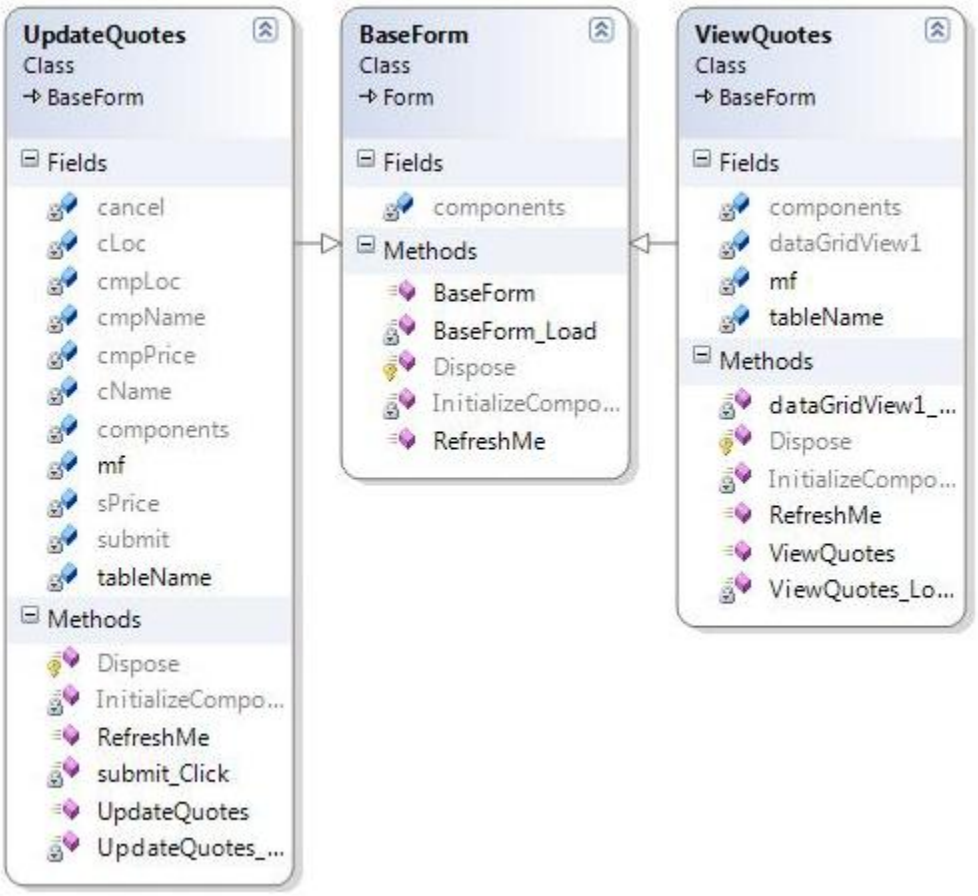


Figure 11: Class diagram of Company Stock Quotes application

The database schema for the Company Stock Quotes application is as follows.

Attribute	Data Type
COMPANY_ID	numeric(18, 0)
COMPANY_NAME	nvarchar(50)
COMPANY_LOCATION	nvarchar(50)
STOCK_PRICE	float

Table 6: Database schema for the Company Stock Quotes application

In the above schema, COMPANY_ID is the primary key of the table. COMPANY_ID is generated automatically through the code whenever a new company is added to the system. Except the COMPANY_LOCATION, all the attributes do not allow null values.

The ACF framework was tested using the Company Stock and Employee Management System applications. The business layer, which includes the ACL Service, ACL Manager and DataManager, is common for both the applications.

4.3 Hypothesis Evaluation

To evaluate the hypothesis, an ACF based application is compared with a traditional polling based application. Figure 12 shows the request-response model for these applications.

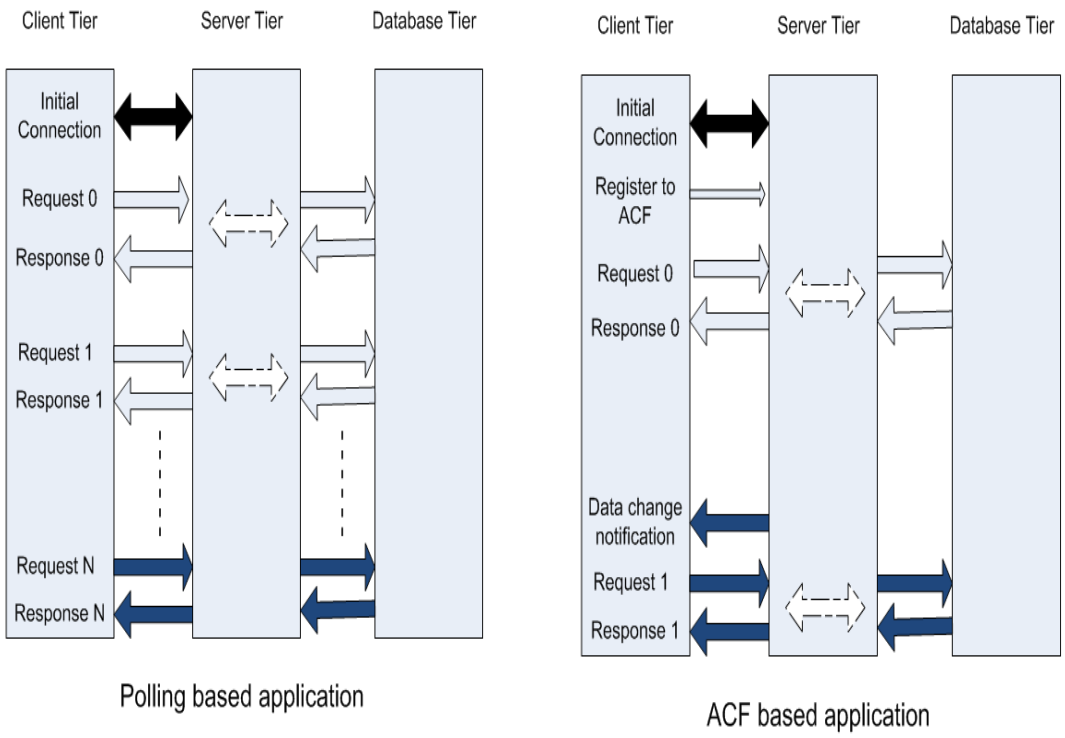


Figure 12: Request and Response

Network Traffic: Figure 12 depicts a scenario in which the client and the server communicate with each other in a request-response model. The first part in the figure 12 represents the request-response model in the traditional polling based application. The second part in the figure 12 represents the request-response model in the applications that use the ACF. The first bi-

directional dark line represents the initial connection between the client and the server. Request0 is the first request made by the client to the server to get the data. The server gets the data from the database and passes the data to the client (represented as Response0). In the polling based application, Request1 to RequestN represent the requests made by the client to get the updated information.

For the traditional pull applications, the client applications had to request the updated information regularly (the time interval of the request depends on urgency of the updated data being required by the client application. The time interval can span from seconds to hours). From Figure 12, we can see that Request1 to RequestN are the requests made by the client to get the updated data. The client gets the updated data when it makes RequestN. On the other hand, the client application that uses the developed ACF framework requests a message from the ACF server only if the data is changed in the database (represented by "Data change notification" in Figure 12). Here, the client need not make continuous request to get the updated data. Whenever the data changes, the ACF middle tier notifies the client tier and then the client can get the updated data. If we compare the number of requests made by the polling based applications to the ACF based application then we can see that the polling based applications make N-1 more requests than the ACF based applications. The client also get N-1 more responses from the server compared to the ACF based applications. This totals to 2N-2 messages between client and server.

There is some message overhead in the ACF based applications. One message each is required to register and unregister itself to the ACF service. One message is required to send notification from the server to the client. So overall the polling based applications require 2N-5 more messages passed between client and server compared to the ACF based applications. Hence, if the application requires up-to-date enterprise data without the user explicitly requesting for it, then the use of proposed ACF may help them to achieve it with less network traffic

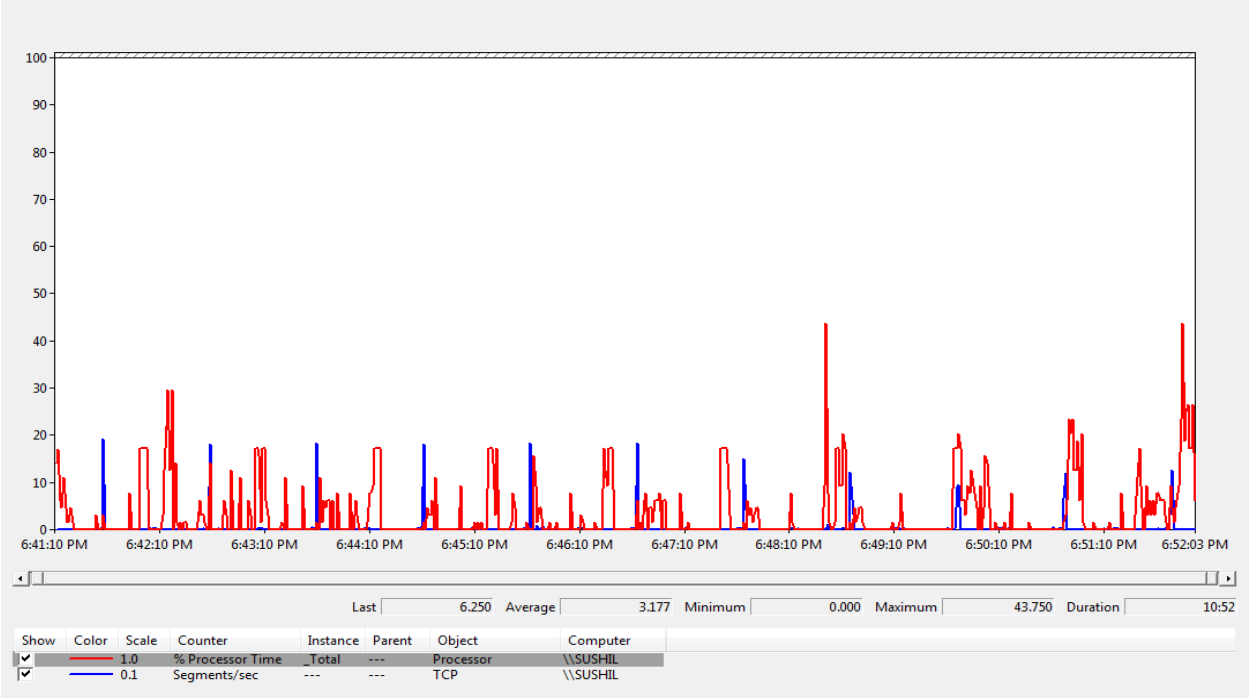
Database overhead: The traditional applications that use pull frameworks requests the data from the servers. These servers then fetch the data from the database irrespective of whether the data is changed. For small time intervals the ACF based application performs noticeably better compared to the traditional polling based application.

From Figure 12 we can see that the servers in traditional polling based applications will fetch the data N-2 times more compared to the ACF based application. The database overhead increases as the number of rows fetched from the database increases. In the applications that use the developed ACF framework, the servers fetch the data from the database only if the data is changed in the database and hence it avoids unnecessary database calls. This improves the database overhead in the ACF based application.

Experimental Results: The ACF based application is compared with a traditional polling based application based on the performance monitors provided by the MS SQL Server Profiler [7]. To monitor the performance of the application, the data generated from various components of the server are stored in a log file. The logged data are exported for the report generation and presented as graphs or histograms using system monitors. There are many components that can be monitored. The major components are as follows:

- Database processing
- Memory
- Network

For the evaluation of hypothesis, the network and database processing components of the server are monitored. To represent both the traditional and the ACF based application, the Company Stocks application is used. The database used for this application contains approximately 10000 rows. The main aim of both this application is to get the updated data immediately. To satisfy the above condition, the traditional polling based application needs to request the data from the server at regular intervals. The performance of the traditional polling based application is monitored with different time intervals. Figure 13-16 show the results for both the applications based on the database processing time (database processing) and segments per second (network) monitors. All the data presented below are collected using the same hardware and software configurations.



: Segments/sec **X –axis:** Time

: % Processor Time **Y-axis:** Scale (0-100)

Figure 13: Traditional Polling based application -1 minute interval (Span: 10 min)

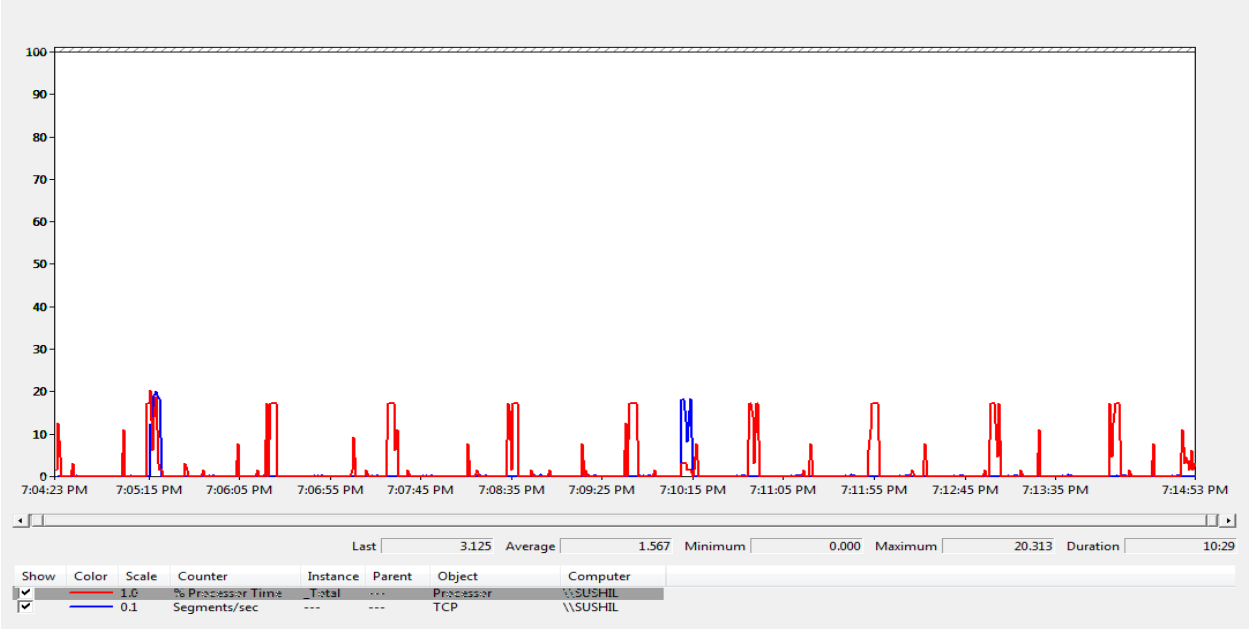


Figure 14: ACF based application (Span: 10 min)

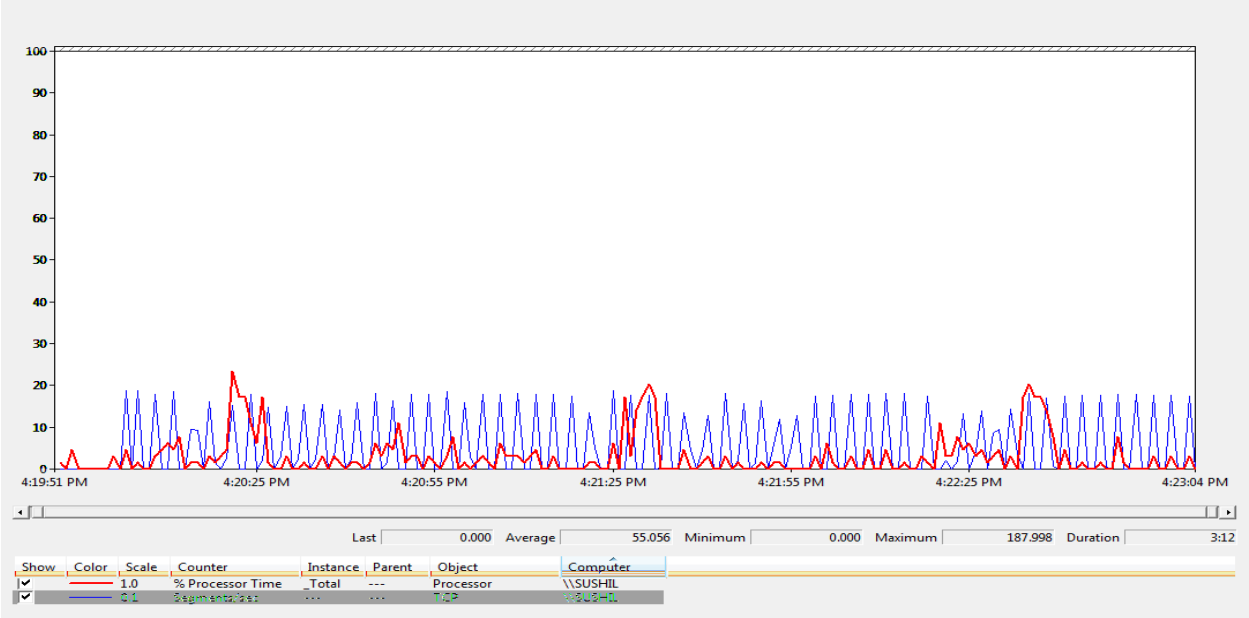


Figure 15: Traditional Polling based application - 2 seconds interval (Span: 5 min)

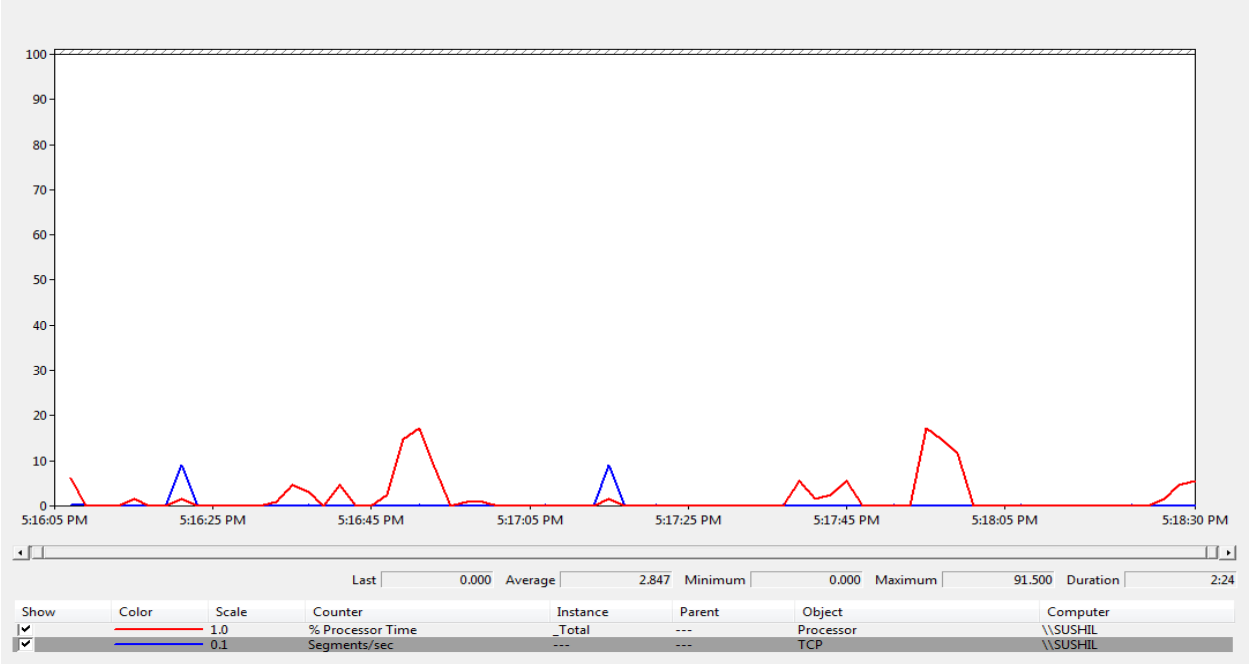


Figure 16: ACF based application (Span: 5 min)

Following are the hardware software configurations.

Database Server:

Operating System: Windows XP

RAM: 1 GB

Hard Disk: 80 GB

ACF Server:

Operating System: Windows Vista

RAM: 1GB

Hard Disk: 120 GB

The following performance monitors were considered to compare the traditional polling based application and ACF based application.

1. Segments/sec
2. Processor Time

Segments/sec: This is the general indicator about the intensity of the TCP/IP traffic. It is the total of Segments Received/sec, Segments Sent/sec, and Segments Retransmitted/sec.

Segments Received/sec is the rate at which segments are received, including those received in error. This count includes segments received on established connections.

Segments Sent/sec is the rate at which segments are sent on current connections, but excluding those containing only retransmitted bytes.

Segments Retransmitted/sec is the rate at which segments are retransmitted by TCP, that is, segments transmitted containing one or more previously transmitted bytes.

Processor Time: This counter provides a measure of the time spent by the SQL server working on productive threads and servicing requests

The graph shown in figure 13 represents the performance of the traditional polling based application. As mentioned earlier, this application polls for the data at a time interval of one minute. The time span for the collected data is ten minutes. The graph shown in figure 14 represents the performance of the ACF based application for the same time span. We can see from the graphs that the traditional polling based application has more network overhead and database processing compared to the ACF based application. The overview of this graph is as follows.

	Polling based application - 1 min interval	**ACF based application.**	**Polling based application - 2 sec interval**	**ACF based application.**
Span	10 min	10 min	5 min	5 min
Avg. DB processing time	3.177	1.567	3.035	1.930
Max. DB processing time	43.750	20.313	23.438	17.188
Avg. Segments/sec	3.583	2.339	55.056	2.847
Max. segments/sec	191.999	199.997	187.998	91.500

Table 7: Overview of the performance analysis

From Table 7 and the graphs mentioned above we can conclude that the average database processing time for the polling based application is greater than the average database processing time for the ACF based application. This is true for both the time intervals. Also the average Segments/sec in the polling based application is greater than the average Segments/sec in the ACF based application. If the application requires data immediately, then the ACF based application performs better than the polling based application. From the above table we can see that the average segments/sec for the polling based application with the interval of two seconds is 55.056. The average segments/sec for the ACF based application is 2.847. On the other hand,

for a higher time interval, the polling based applications perform equally well but still ACF based applications gave good results. We can observe that if the application requires immediate sharing of the data, the ACF based applications perform noticeably better compared to the traditional polling based applications. Thus, the mathematical data and the analysis of the results prove the stated hypothesis.

5. Conclusion

5.1 Current Status of the Project

Active Collections Framework has been successfully implemented. It has the following working modules/features.

- Active Collections Framework using Microsoft's .Net platform and Visual Studio IDE.
- Company Stock Quotes application using the developed active collections framework and MS SQL Server database
- Employee Management System using the developed active collections framework and MS SQL Server database

5.2 Future Work

The scope of the project does not include the deployment of the developed framework on mobile devices. Today smart phones offer advanced capabilities beyond the normal phones. Many features/applications that are available on the PC are also developed for the mobile/smart phones. Companies now look at the mobile devices as fully enabled computers operating within a mobile context [2]. Many companies have started to integrate handheld and wireless devices with their existing corporate information infrastructures. The upcoming revolution in mobile devices drives the need for implementing ACF on the mobile platform.

The mobile applications cannot be considered as the desktop applications reformatted for the smaller displays. The mobile applications are fundamentally different from the desktop applications for various reasons. There are some important areas that should be considered while developing the ACF framework on mobile devices.

Mobile devices follow different protocols than desktop applications. Maintaining the confidentiality of the data carried in these applications is difficult as mobile devices can be carried anywhere outside the secure network of a company. Varying signal strength is also a key issue in developing applications on mobile device. When a mobile device goes out of range, it loses its connection with the server and maintaining state of the application becomes difficult. One should consider these aspects while developing ACF enhancement for the mobile devices.

6. Appendix A

The Employee Management System application loads the Main Form as shown in figure 17 below. It has two menus, Manage and View. Manage menu has Add and Update as its sub menu. View has Display as its sub menu.

View →Display allows a user to see the detailed information of the Employees. The menu navigation is depicted in figure 18 below. The detailed user information is shown in a grid in the Display employee form as shown in the figure 19.

Manage →Add allows a user to add a new employee information to the database. The screen for adding new employee information is as shown in figure 20. The user can click on Add button to add new employee to the database. Clicking on Cancel Button will close the form.

Manage →Update allows a user to update the employee information.

Similarly, Company Stock Quotes Application opens the main screen as shown in the figure 21 below. It has the main menu Update and View.

Clicking on Update opens the update screen as shown in the figure 22 below. The Update button allows a user to update stock Information. The Cancel button allows a user to close the screen.

Clicking on View opens view screen as shown in the figure 23 below. View screen displays the Company Stock Information.

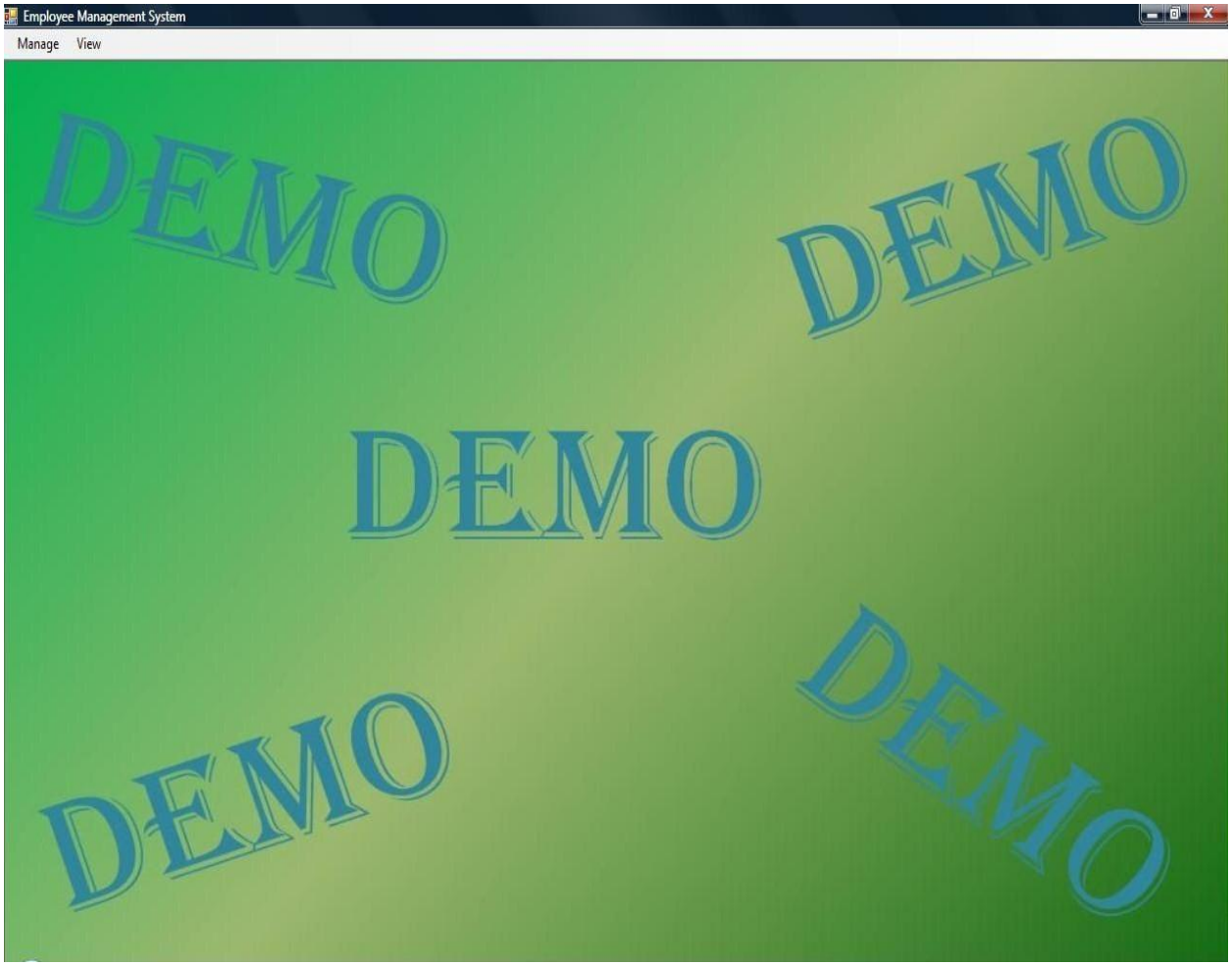


Figure 17: Employee Management System Main screen

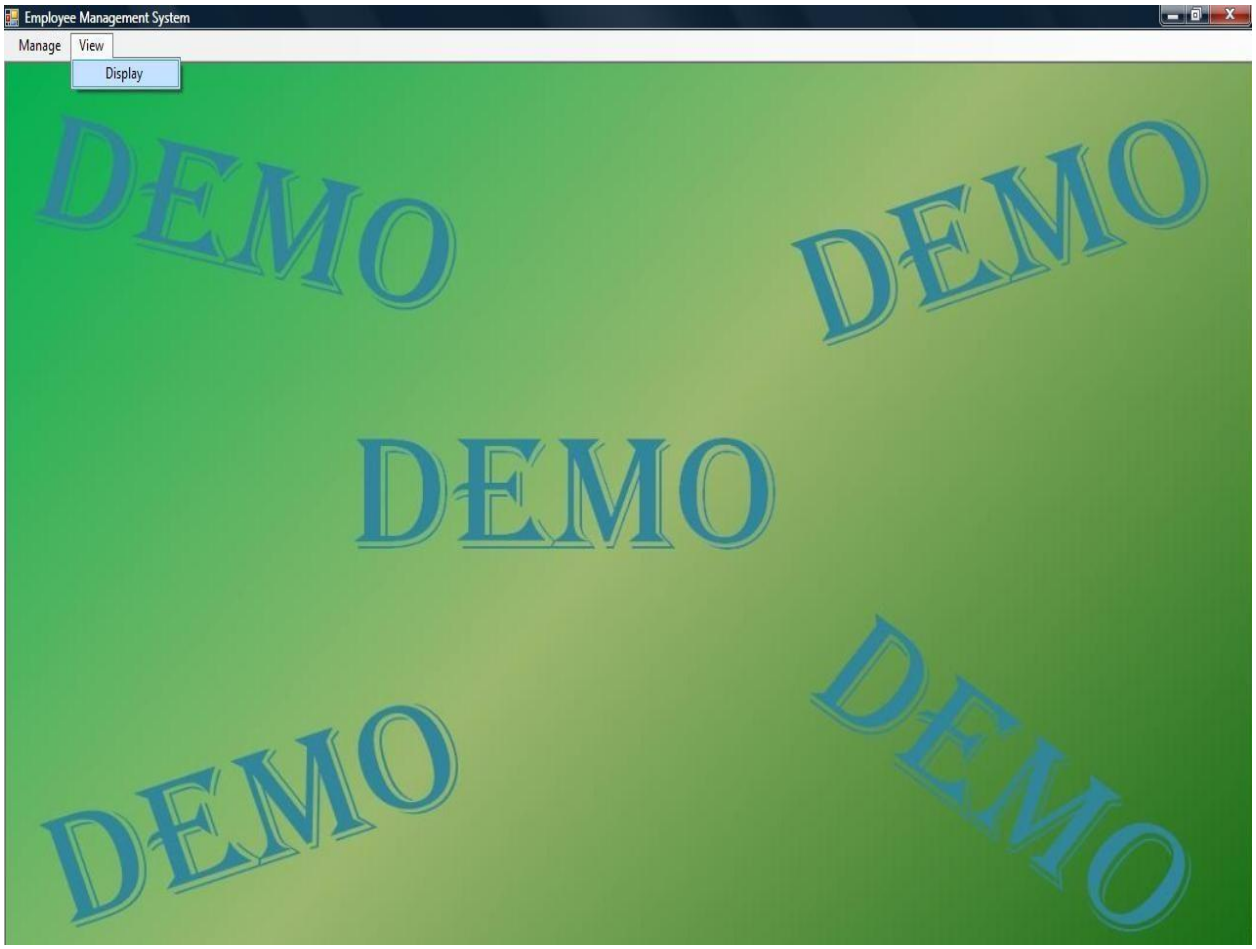


Figure 18: Employee Management System menu items

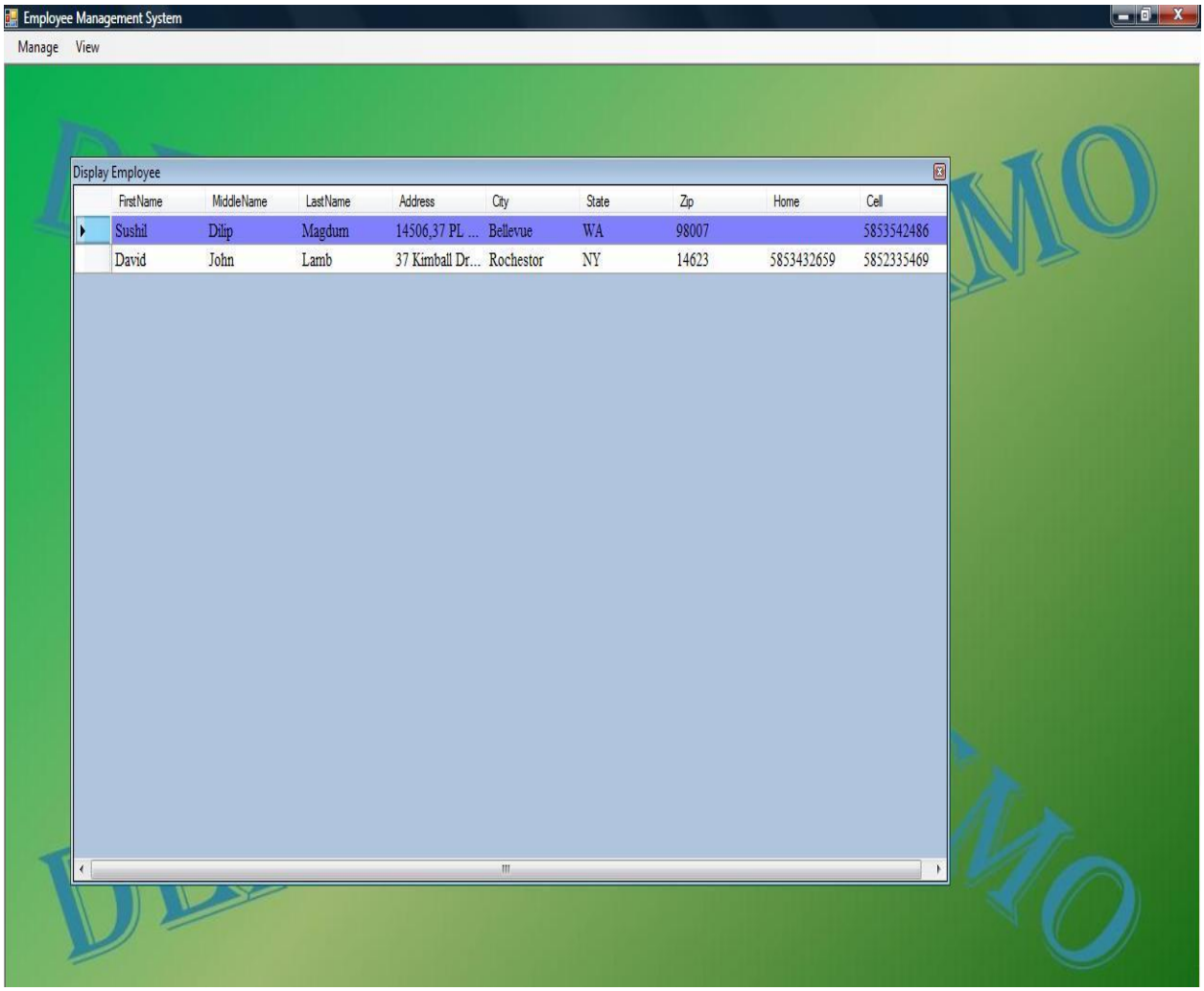


Figure 19: View Employee screen

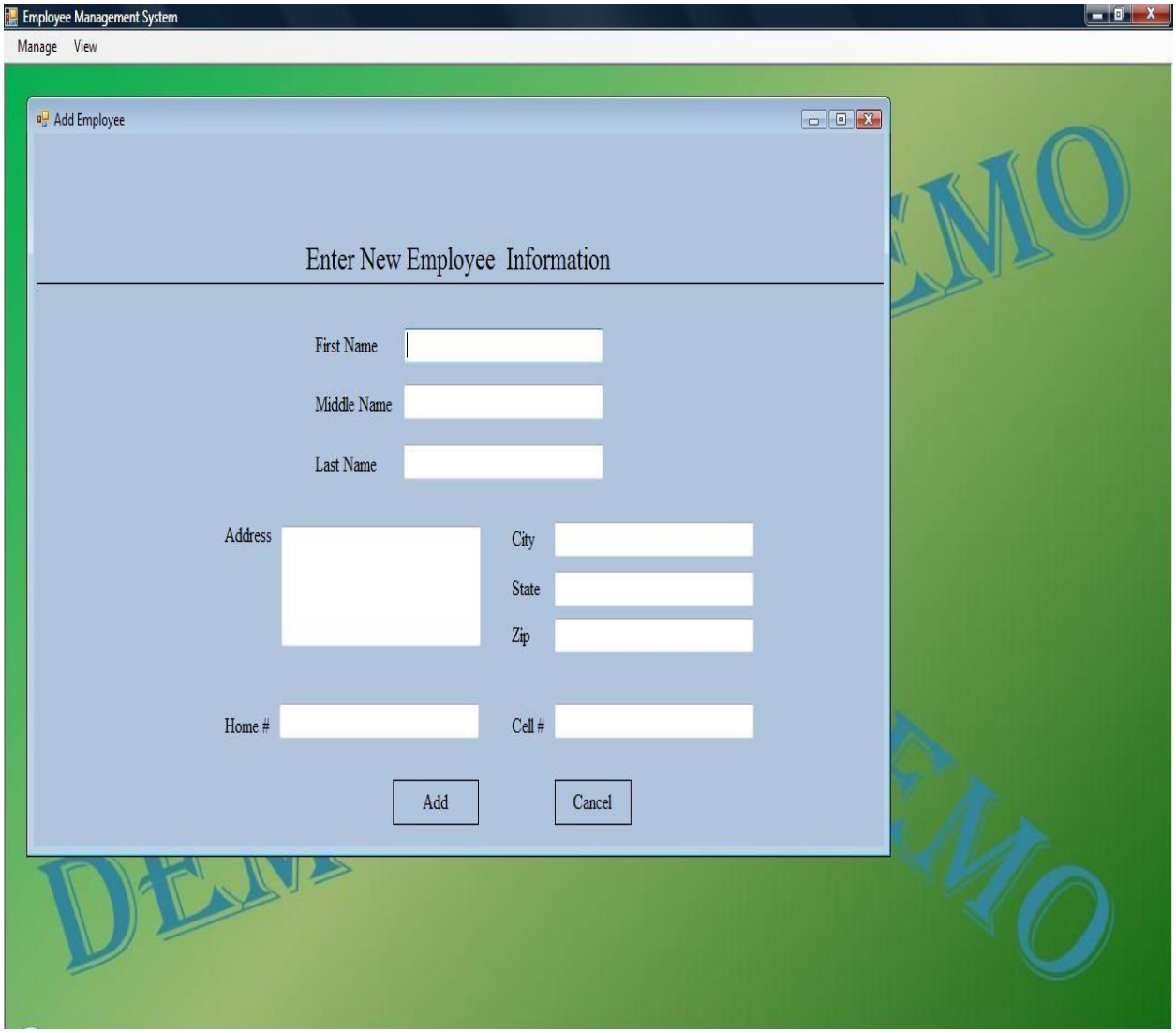


Figure 20: Add Employee Screen

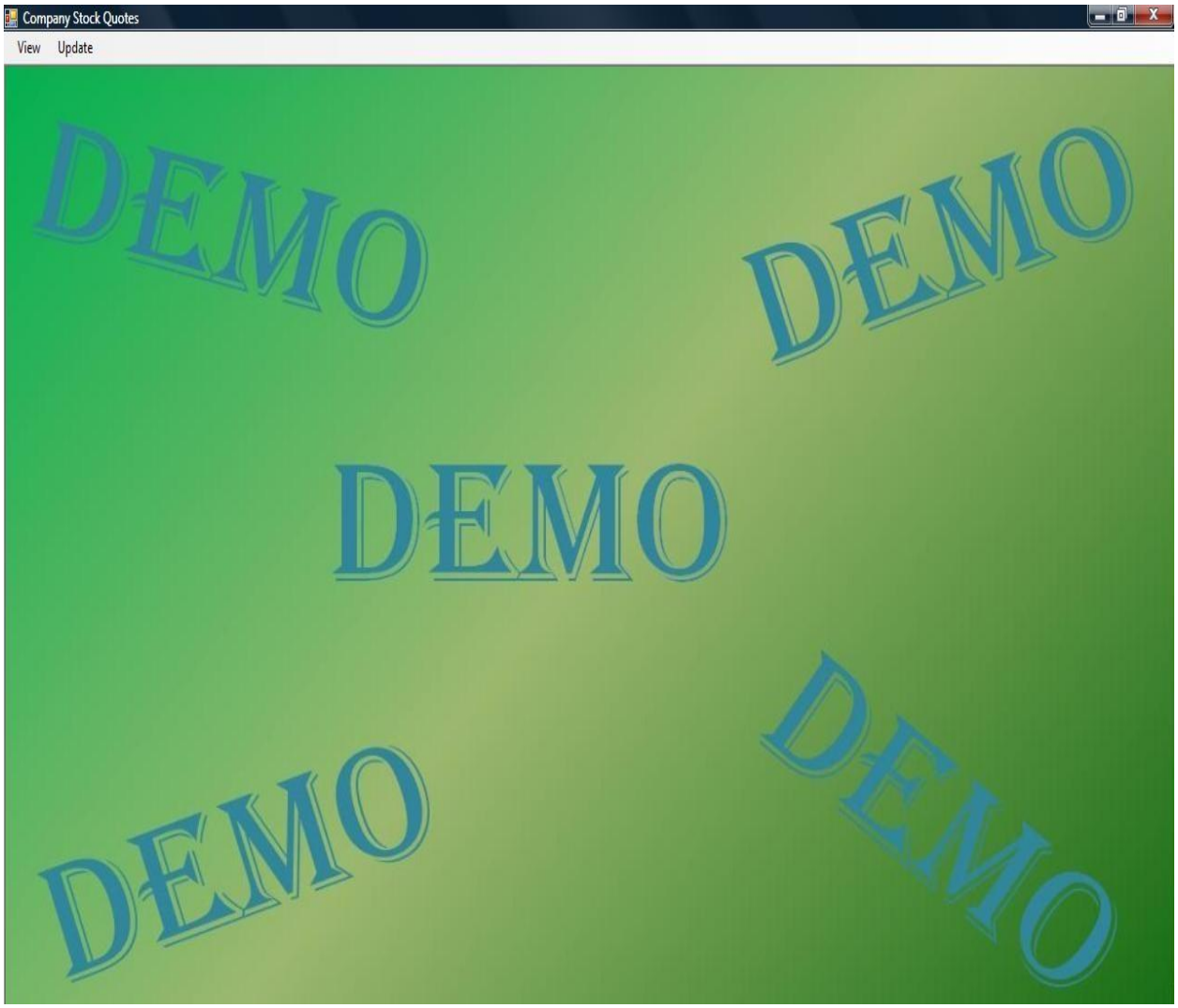


Figure 21: Company Stock Quotes main screen

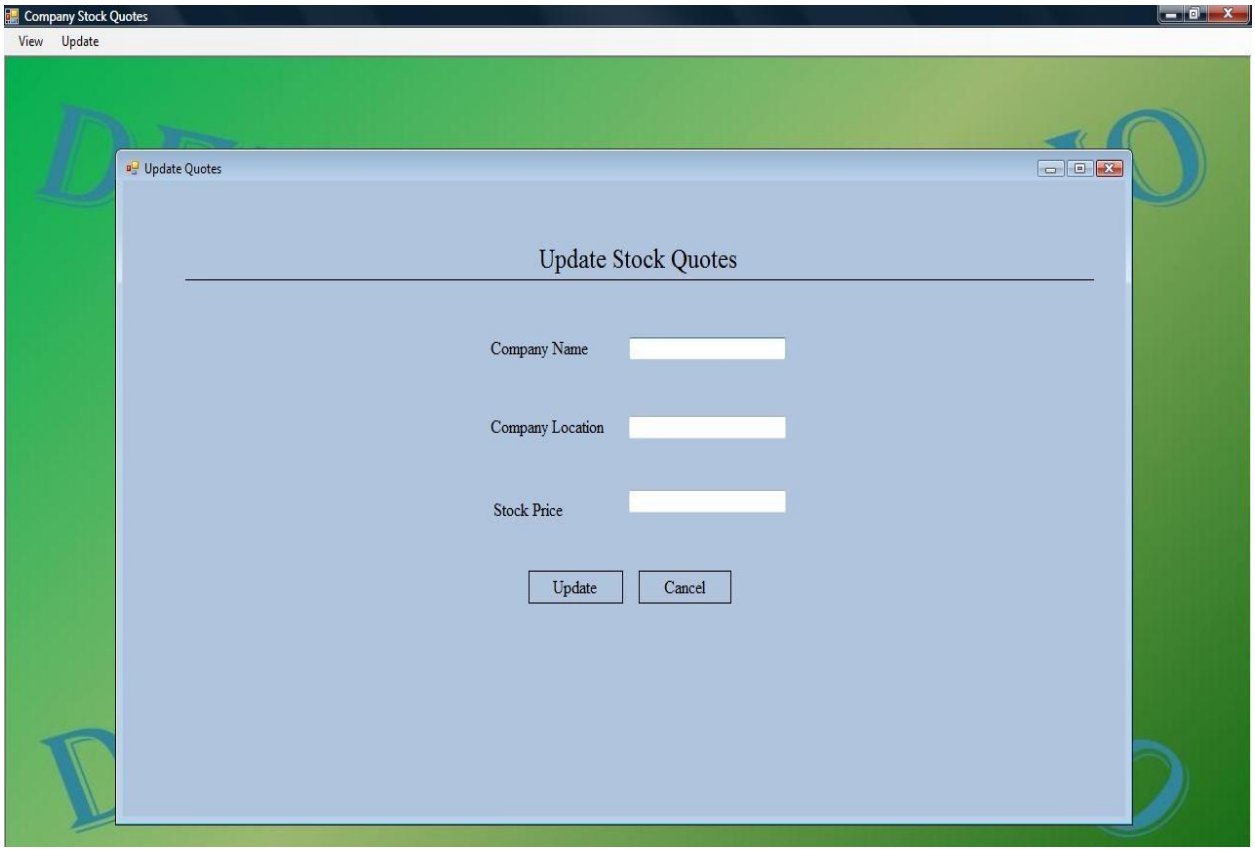


Figure 22: Update stock quotes screen

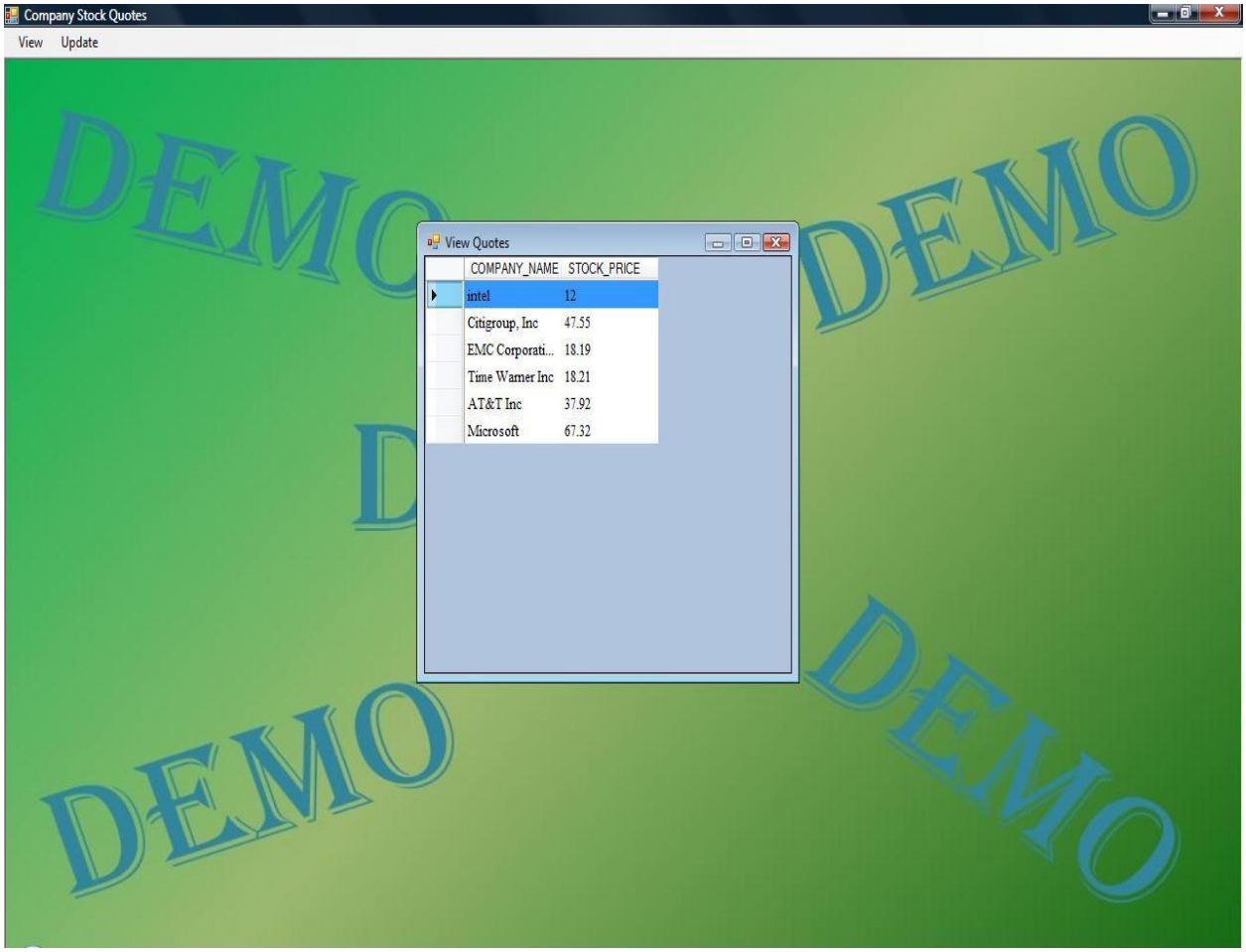


Figure 23: View stock quotes screen

7. Reference

1. Bhaskar, Gopalan. Implementing the Active Collections Framework. Master's Project. Rochester, NY: Rochester Institute of Technology, 2003.

2. Clevenger, Nathan. Mobile Enterprise Application Integration. September 2003. October 2007 <http://www.pocketpcmag.com/_archives/Sep03/ent_mobile-integration.asp>.

3. Kappel, G. and W Retschitzegger. "The TriGS active object-oriented database system—an overview." ACM Press (1998): Volume 27.

4. Microsoft Visual Studio .NET 2003 Professional Edition. February 2007. June 2007 <http://www.microsoft.com/products/info/product.aspx?view=22&pcid=9fdcc2af-6b86-4ee8-9b71-90cebe8626e6&type=ovr>.

5. New for Visual Studio 2008. October 2007. November 2007 <http://blogs.msdn.com/fxcop/archive/2007/10/03/new-for-visual-studio-2008-code-metrics.aspx>.

6. Paton, Norman W. and Oscar Diaz. "Active Database Systems." ACM Press 31 (1999).

7. Performance Counters. June 2007 <http://msdn2.microsoft.com/en-us/library/w8f5kw2e(VS.71).aspx>.

8. Raj, Rajendra K. "Active Collections Framework." ACM Press 7 (1999).

9. Technology Overview. February 2007. March 2007 <http://msdn2.microsoft.com/en-us/netframework/aa497336.aspx>.

10. The Singleton Pattern, and the Observer Pattern. January 2006. March 2007 <http://java.sun.com/developer/JDCTechTips/2006/tt0113.html>.

11. Wikipedia Cyclomatic complexity. February 2007. June 2007 < http://en.wikipedia.org/wiki/Cyclomatic_complexity>.