DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MESSAGING SYSTEMS / DESIGNING AND IMPLEMENTING MESSAGING ARCHITECTURES FOR SOFTWARE APPLICATIONS

by
Pınar KILINÇ

March, 2008
İZMİR

MESSAGING SYSTEMS / DESIGNING AND IMPLEMENTING MESSAGING ARCHITECTURES FOR SOFTWARE APPLICATIONS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science in
Computer Engineering, Computer Engineering Program

by
Pınar KILINÇ

March, 2008
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"MESSAGING SYSTEMS / DESIGNING AND IMPLEMENTING MESSAGING ARCHITECTURES FOR SOFTWARE APPLICATIONS"** completed by **PINAR KILINÇ** under supervision of **ASST. PROF. DR. ŞEN ÇAKIR** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Şen ÇAKIR

Supervisor

(Jury Member)

(Jury Member)

Prof.Dr. Cahit HELVACI
Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Asst. Prof. Dr. Şen ÇAKIR for her helps to complete my thesis.

I also would like to thank my friend Semih TURGUT for his support and ideas about writing this thesis. I would like to that to my parent and my grant mother about their patience during the writing phase of thesis.

Pınar KILINÇ

MESSAGING SYSTEMS / DESIGNING AND IMPLEMENTING MESSAGING ARCHITECTURES FOR SOFTWARE APPLICATIONS

ABSTRACT

In software applications, as an architecture multi-tier (N-tier) architectures generally are used. Multi-tier architectures isolates presentation, business logic and data operations from each other by dividing them generally into at least three layers.

Software framework is an application that is used to develop complicated applications and it is designed with reusable manner. It can be used for all applications that are in the same domain. While developing a software application, developers write less code by using software frameworks so requirement analysis phase of software development phases can be much longer than implementation phase.

The main goal of this thesis is researching software frameworks, application architectures "especially N-tier application architectures", enterprise integration patterns and implementing enterprise integration patterns to produce a re-usable software framework that can used in wide range application domain. While producing a software framework, different enterprise integration patterns are researched and used; Message types, message routers, message dispatchers, Pipes and Filters are main patterns that are used to design and implement the software framework.

Rota Framework is a software framework that can be used to develop a software application. Rota Framework uses specialized components and messaging patterns to communicate between layers of applications. It can be used with wide variety of platforms: Web applications, desktop application, web services and etc…

Keywords: Software frameworks, Integration patterns, N-Tiered applications and architectures

MESAJLAŞMA SİSTEMLERİ / YAZILIM UYGULAMALARI İÇİN MESAJLAŞMA ALTYAPILARININ TASARLANMASI VE GERÇEKLEŞTİRİLMESİ

ÖZ

Yazılım uygulamalarında genel olarak mimari anlamında çok katmanlı mimariler kullanılır. Çok katmanlı mimariler uygulamanın ara yüz katmanını, iş bileşenlerini (iş katmanını) ve veri katmanını birbirinden ayırır ve genellikle en az üç katmandan oluşurlar.

Yazılım ana çatıları karmaşık uygulamaları gerçekleştirmek için kullanılan bir uygulamadır ve yeniden kullanılabilirlik mantığı ile tasarlanırlar. Aynı alanda bulunan bütün yazılım uygulamaları için kullanılabilirler. Yazılım geliştiriciler yazılım ana çatıları kullanarak uygulama geliştirdiklerinde daha az kod yazarlar ve uygulama geliştirimi sırasında ihtiyaçların toplanması ve analiz edilmesi için harcanan zaman, gerçekleştirim için harcanan zamandan daha uzun olabilir.

Tezin başlıca amacı, yazılım ana çatılarını, uygulama mimarilerini "özellikle çok katmanı mimarileri", kurumsal bütünleşme desenlerini araştırmak ve bu desenleri gerçekleştirerek geniş uygulama alanlarında kullanılabilen bir yazılım ana çatısı oluşturmaktır. Yazılım ana çatısı oluştururken farklı kurumsal bütünleşme desenleri kullanılmaktadır; Mesaj tipleri, mesaj yönelticileri, mesaj göndericileri ve boru ve filtreler yazılım ana çatısını gerçekleştirmek için kullanılan desenlerdir.

Rota Ana Çatısı uygulama gerçekleştirmek için kullanılan bir yazılım ana çatısıdır. Rota ana çatısı kendi içerisinde özelleşmiş bileşenler içerir ve uygulamanın farklı katmanları arasındaki iletişim sağlamak amacı ile mesajlaşma desenlerini kullanmaktadır. Web uygulamaları, masaüstü uygulamaları ve web servisleri vs. gibi farklı platformlardan kullanılabilirler.

Anahtar Sözcükler: Yazılım Ana çatıları, Bütünleşme desenleri, Çok katmanlı uygulamalar ve mimariler

CONTENTS

Page

CHAPTER ONE

INTRODUCTION

Software frameworks are applications that are not executable but are used to implement a software application in re-usable fashion. By using software frameworks, developers can spend much more time to have requirements of an application and analysis of these requirements than developing the application. With this manner, software frameworks are used to develop a software application.

Software Frameworks must support wide range of application domain and can be usable for different requirements of application domain. To achieve this, software frameworks must have flexible points called *hot spots* to meet different requirements of application domain. Hot spots of software framework can be applied to different part of application; logging, security, transaction managing and exception handling, etc… Especially, software frameworks consist of hot spots as abstract classes. To use the software framework, abstract classes of framework must be implemented. Software frameworks must also have unchangeable points called *frozen spots.* Frozen spots produce kernel of frameworks and the kernel of frameworks does not change application to application.(Markiewicz & Lucena, 2000)

Software frameworks can be categorized as white box frameworks, black box frameworks and grey box frameworks. Users of white box frameworks must know the kernel and internal structure of the framework in order to develop an application with software framework. In white box frameworks, if enhancement is wanted to be applied to the framework according to business needs, inheritance is used. Using inheritance makes the framework tightly coupled. Because of this, black box frameworks are preferable than white box frameworks. It can be learned by application developers easily. It is formed by using components. If an enhancement is wanted to be applied to a black box framework, implementing interfaces, composition or delegation are used to do this.(Earles (n.d.); Markiewicz & Lucena, 2000; Ronalt, 1997.)

Using software frameworks to develop an application has advantages:

- Application that uses software framework becomes reliable, because software frameworks are tested more than one time before.
- Developers who use software frameworks develop applications faster than developers who do not use software frameworks.
- Applications that use software frameworks decrease the probability of failure.
- Maintaining, learning and supporting of application that uses software frameworks is easier that application that does not use framework. (Anonymous (n.d.))

In N-tiered application architectures, there are at least three tiers (layers); Presentation Tier, Business tier and Data Tier. Designing and implementing an application by using N-Tier architectures reduces the cost of application implementation and maintenance than using one layered application. (Chartier R., (n.d.))

Enterprise Integration Patterns are used to communicate and integrate different applications. There different types of integration concepts are: Messages, Pipes and Filters, Message Router, Message Dispatcher are some of the patterns of Enterprise Integration Patterns. Messages consist of data that will be transmitted to accomplish a common task and there are different kinds of messages. Command message is a message type that is used to invoke a method in another location, it contains a command. Document messages contain data that must be carried from one application to another. Event messages have event notification that will be passed between applications. Request and Reply messages are the combination of document messages and command messages. Request message is command message and reply is the document message. Pipes and Filters pattern is used to apply different tasks to messages in a common sequence. Logging, authorizing, auditing can be task that will be applied to request message before a process will be executed. Message Router is a modified version of Pipes and Filters pattern. In pipes and filters, a common sequence of tasks is applied to all messages but in some circumstances, some tasks may not need to apply some messages. Message Routers are a centralized decision making points that decide which tasks are applied to

different messages according to the content of the message. Message Dispatcher is a pattern that is applied at the receiver side of the message. At the receiver side, according to the content of the message the performer of the message is decided.(Hohpe & Woolf, 2004)

The main goal of this thesis is researching application software frameworks, application architectures especially N-tier application architectures, enterprise integration patterns and implementing enterprise integration patterns to produce a re-usable software framework that can used in wide range application domain.

This thesis is divided into 4 chapters. First chapter summarizes the subject of the thesis, explains the purposes and advantages of thesis. Chapter 2 explains software frameworks, categorizes software frameworks and briefly describes layered (N-tier) applications. In Chapter 3, Enterprise integrations patterns are explained in detailed. Chapter 4 explains specialized framework called Rota Framework that is designed by using Enterprise integration patterns that is explained in Chapter 3. Chapter 5 explains an application called Book Store Application that is developed by using Rota Framework that is explained in Chapter 4. Finally chapter 6 presents the conclusion of thesis.

CHAPTER TWO

SOFTWARE FRAMEWORKS

2.1 What is Software Framework?

A framework is an application that is used to develop complicated applications and it is designed with re-usable manner. Re-usable manner can be used in a specific part of software applications or entire of software applications. To accomplish the re-usability purpose, software frameworks are usually designed with object-oriented approach and especially contain interfaces, abstract classes and methods. In addition to object-oriented approach, messaging systems that is evolved with object-oriented approach can be used too.

Software frameworks consist of code libraries, extra software programs other that software application and scripts to make developing of software application easy.

Software frameworks must support all possible applications to develop them easily, in addition to this applications may have different requirement so software frameworks must be flexible to meet different requirements of applications. Flexible points of software frameworks generally are implemented with abstract classes and methods. In Software applications that will be developed, the abstract parts of software frameworks are implemented. These points of software framework are called hot spots. (Markiewicz & Lucena, 2000)

In addition to flexible points of software frameworks, some features and part of software frameworks can not be changed and becomes permanent. They are called frozen spots. These parts are already implemented in software framework and are not requirement specific parts.

Figure 2.1 explains software frameworks' hot spots with a metaphor "engine". Software framework is an engine and hot spots of a software framework are power inlets. If the implementation of power inlets of engine is power.

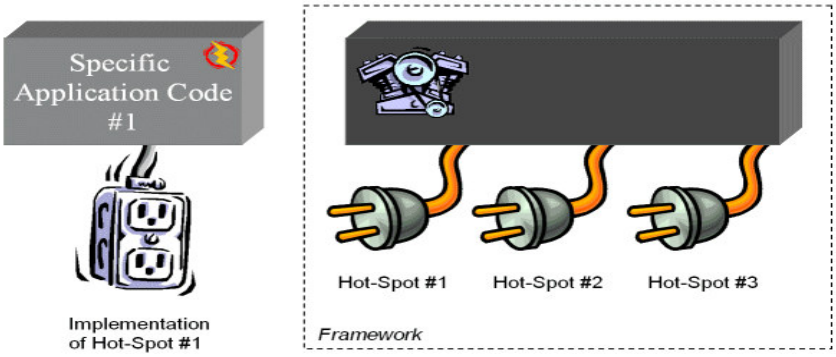

Figure 2.1 The metaphor for hot spots of software frameworks. (Markiewicz & Lucena, 2000)

2.2 White Box and Black Box Frameworks

Frameworks can be classified like white box frameworks and black box frameworks. It is determined with extensibility of framework. In White box frameworks, developers must understand the internal structure of framework to develop applications extend the abilities of framework. White box frameworks come with source code and developers implement classes by inheriting from frameworks. Model View Controller framework is an example of a white box framework. To create a new view, developers must create a new class that inherits from view class. These frameworks are also called architecture – driven frameworks.

In Black box frameworks, developers must not need to know the internal structure of framework to use and develop a software application. Application can be developed with configuration script or configurable components that are used to work with frameworks.(Figure 2.2) Black Box frameworks are also called Data-Driven frameworks. Developing an application with black-box frameworks is much easier than developing with white-box frameworks. (WAKE, W.C., 1994-2006) VisualWorks is an example of a black box framework. It is a modified version of model view controller that is suitable to be a black box framework. There is a generalized View class. To create a new view and modify it, configuring some properties of subclasses of generalized view class.

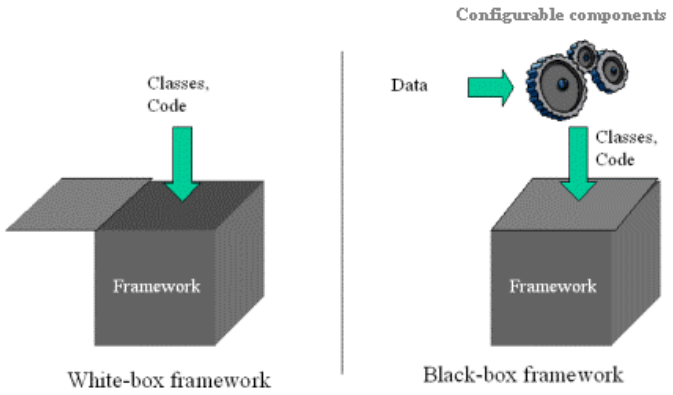

Figure 2.2 White and Black Box software frameworks. (Markiewicz & Lucena, 2000)

Frameworks change over their lifetime. When a framework is new, it behaves like a white-box framework. It can be changed creating new classes to implement a software application. When it begins evolving, it behaves like a black box framework. (Markiewicz & Lucena, 2000)

2.3 Layered Architectures

Each software application has several elements such as performance, scalability, usability or flexibility. To decide what kind of architecture will be used, these elements must be considered.

N-Tier (Multi Tier or Multi Layer) architecture is a software development method that divides the whole software application into a layered structure. As the usual definition, N-Tier architecture refers to the architecture of an application that has at least 3 "logical" layers that are separate. Each layer interacts with only the layer directly below, and has specific function that it is responsible for. Here the “N” stands for the number of layers at the architecture. It changes depending on the application.

Why is N-Tier architecture used? Because each layer can be located on physically different servers with only small code changes, they scale out and handle more server load. Also, each layer’s content is completely hidden to other layers and this makes it possible to change or update one layer without recompiling or modifying other layers. This brings flexibility and reusability to the application. As

an example, by separating data access code from the business logic code, when the database servers change you only needs to change the data access code. Because business logic code stays the same, the business logic code does not need to be modified or recompiled.

An n-Tier application usually has three tiers;

- Presentation tier (Presentation tier is the combination of Presantation Graphical User Interface -Presentation GUI- and Presentation Logic Tier.)
- Business tier
- Data tier

But here some other tiers will also be explained such as;

- Data access tier
- Presentation logic tier

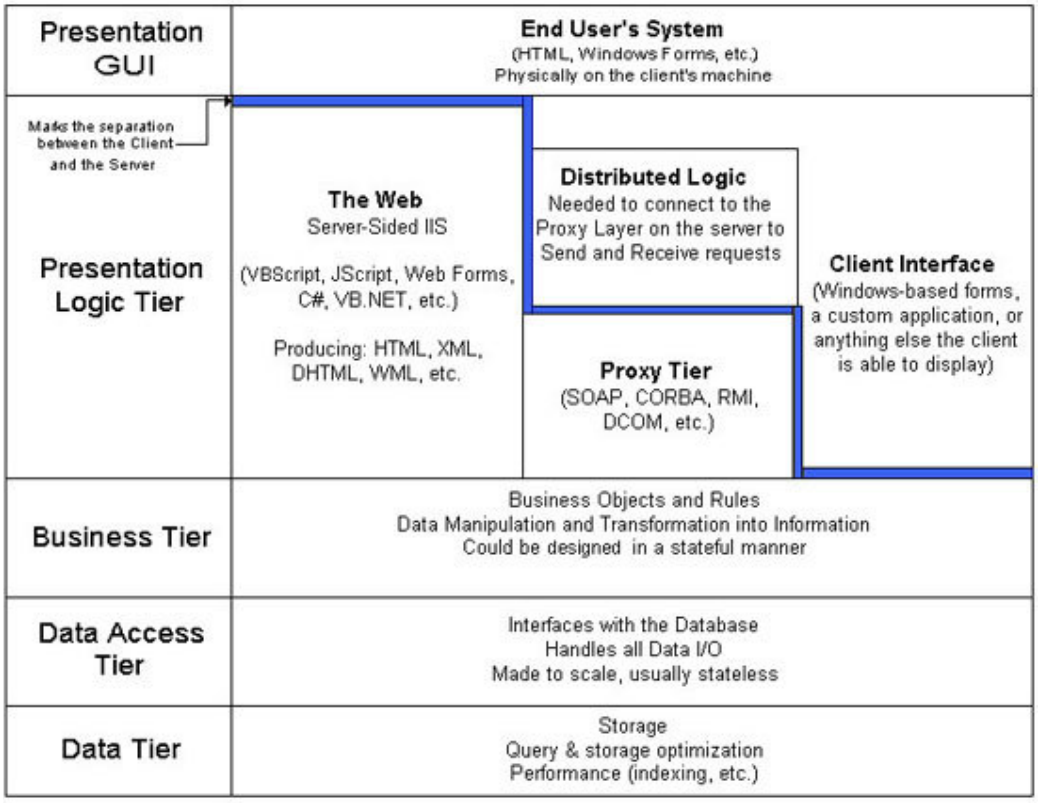

Figure 2.3 N-Tier Application Example. (Chartier (n.d.))

Figure 2.3 shows an example of n-tier architecture. There are five layers (tiers) and these are Presentation Graphical User Interface (Presentation GUI), Presentation Logic Tier, Business Tier, Data Access Tier and DataTier. The explainations of these tier are as follows;

2.3.1 Data Tier

Data Tier is the data source of the application. Usually it's the Database Management System or it can be text or binary files depending on the application. Task of this layer is dealing with the storage and retrieval of information. This layer must not contain any business logic issues. Depending on the DBMS capabilities there can be stored procedures, indexes or triggers can be defined in this layer.

2.3.2 Data Access Tier

This layer contains some generic methods to interface with the data. It creates or opens connections to the data source and handles all data I/O. This layer can be thought as a part of the business tier.

2.3.3 Business Tier

This layer is the main part of the application that implements the tasks. It contains business rules, data manipulations, class definitions, and application code itself. It's responsible for processing the data and sending it the presentation tier. Also it's responsible for getting and processing the information from presentation layer and sending it to the data layer using the insert, update or delete statements.

2.3.4 Presentation Logic Tier

This is the definition layer that defines how to present the application to the end user. It contains the user interface files like Web documents or Desktop application forms. This layer is responsible to handle the data transformation into something usable and readable by the end user.

2.3.5 Presentation GUI

It's the graphical user interface files physically located on the clients side. This layer is a part of the presentation layer. From the Figure above it can be said that presentation logic tier and presentation GUI tier together makes the presentation tier.

As mentioned in section 2.3, different tiers can exist in an application architecture. Different tiers can communicate with another by using enterprise integration patterns Enterprise Integration patterns are explained in Chapter 3.

CHAPTER THREE

ENTERPRISE INTERGRATION PATTERNS

Enterprises may have different application that are responsible to handle different business function. The reason of having different applications, so the reason of integration are;

1. Writing business applications is hard and writing a big application to run the complete business fıunction is impossible.
2. Dividing business functions into different applications can be usefull to obtain function specific and efficient solutions. An IT organization is not interested in single enterprise solution, interested in solution or solutions that cover the business requirements.
3. Users such as customers, internal users execute business functions, regardless of the how many internal systems the business function cuts across. For example a customer placing a new order may require the coordination of many systems.
4. The technologies that the applications are developed can be different and one application can need to communicate another. (Hohpe & Woolf, 2004)

3.1 Integration Problems

Different applications that are build different technologies and platforms imply integration but integrating applications also have different challenges. The disadvantages of integration are as follows:

- Networks are unreliable
- Networks are slow
- Applications are different
- Changes are costly

3.2 Approaches to Integration Problems

To avoid and decrease the implications of integration problems that are mentioned above, different approaches are developed over time. Different approaches solved different problems partially. The integration approaches are as follows:

3.2.1 File Transfer

File Transfer approach integrates systems by using files. File operations are supported by all operating system, programming languages and platforms The methods of implementing file operations are different but all systems have file operations. (Universal Method)

File transfer method transports data as a file between systems that will communicate each other. Figure 3.1 explains file transfer method. Application A exports data to a file and put it to a decided location. Application B reaches the location and imports the data.

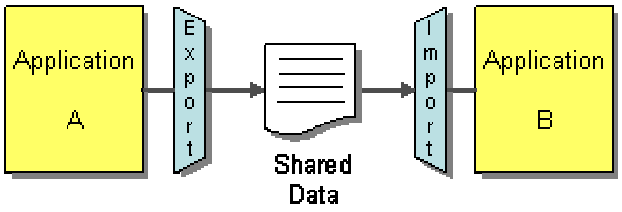

Figure 3.1 File Transfer Method (Hohpe & Woolf, 2004)

There are some important points the integrated systems must negatiate: File format that will be used, shared location and when to produce and consume the files. Very rarely, the output of one application fits the needs of another, so the application must process file to leave needless data within the file. Because of this, standard file formats are used between applications like XML and etc. File operations require certain amout of effort to produce and process files (data) so exporting data to a file and importing data from a file operations are scheduled like nightly, weekly and so on. In addition to these, the location of files that will be put by one application and consumed by another application must be negatiated.

By file transfer method applications become loosely-coupled, because they do not need to know the internals. The producer application provides the file. The file's

content, format and location are negotiated and the producer application does not require to know how the consumer application process and read file.

The disadvantage of file transfer method is data inconsistency. Providing and consuming files are organized by scheduled (nightly, weekly, ...) so one application may deal with latest data and the other application may deal with old data. This situation can cause inconsistency problem. (Hohpe & Woolf, 2004)

3.2.2 Shared Database

Another approach that is derived from file transfer method is shared database method. In file transfer method, data sharing with timeless manner causes data inconsistency. While one application are running with latest data the other one works with old data. This problem is very important for businesses.

In shared database method, applications store their data to a shared database and database schema. The schema covers the database requirements of all applications that are integrated with shared database method. Figure 3.2 explaines the shared database method.

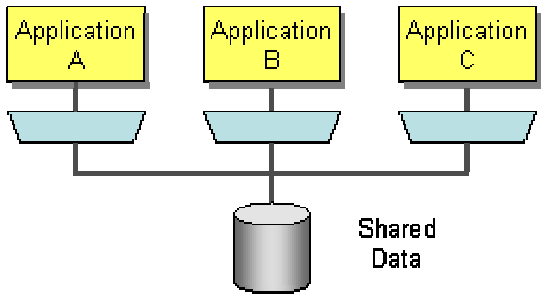

Figure 3.2 Shared Database Method(Hohpe & Woolf, 2004)

Shared database method resolves the problems of file transfer method, but it provides new problems. One of the biggest problem is constructing a unified database schema that meets the needs of multiple applications. The programmers find difficult to work with a unified schema, and database changes affect other applications that must not be affected.

Working with shared database and schema causes performance bottlenecks and deadlocks as one of the application reads the data, the other one wants to manipulate it. If the applications that are distributed multiple locations and accessing single database, they must reach the database over network. Accesing database accross the network and manipulating data are too slow operations for distributed applications. (Hohpe & Woolf, 2004)

3.2.3 Remote Procedure Invocation

While *File Transfer* and *Shared Database* methods implement data sharing, *Remote Procedure Invocation* method implements data and functionality sharing. Because sharing data between applications can cause actions to be taken accross applications. For example; changing address information of one customer can necessitate a registration operation.

Figure 3.3 explains working steps of remote procedure invocation method. Application B publishes the methods that are accomplished by the applications. This publication operation is done by interfaces. By interfaces, application A does not need to know the internal structure of Application B. It is enough to know the capabilities of Application B that are represented in interface of Application B (skeleton). Application A invokes a method of Application B and Application B returns a response that consists of result of related method invocation.

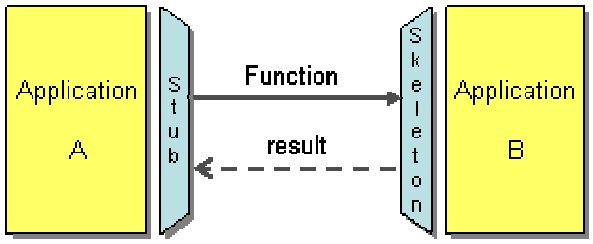

Figure 3.3 Remote Procedure Invocation (Hohpe & Woolf, 2004)

Remote Procedure Invocation applies encapsulation mechanism to data by using interfaces that is mentioned above. If one application needs data or information that is owned by another application, the application wants it directly by using the interface of another one or if an application want to modify the data of another, it call the related method by using interface of another again. By the encapsulation, the applications can change its internal structure or data without affecting the other application.

This method consists of performance imlications when many applications want to reach a method of one application. Application that deals out other application can be a deadlock beceuse of accessing multiple applications at the same time.

The Remote Procedure Invocation method is a synchronous method when one application is busy, the other application have to wait it to become available. (Hohpe & Woolf, 2004)

4.2 Messaging System

Messaging system is one of the integration style that performs asynchronous, high speed and reliable communication between different systems.

Applications that will send data to another applications put data with special format (message) to a message bus, and receiver applications consume formatted data (message) from the message bus. In the message bus, broadcasting approach (one message to all receivers) or routing approach (one message to a specialized receiver) can be used. (Figure 3.4) With this property, development and integration of the system decouples from each other.

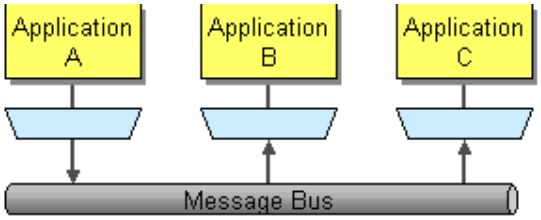

Figure 3.4 Messaging Systems (Hohpe & Woolf, 2004)

“Asynchronous messaging is fundamentally a pragmatic reaction to the problems of distributed systems. Sending a message does not require both systems to be up and ready at the same time. Furthermore, thinking about the communication in an asynchronous manner forces developers to recognize that working with a remote application is slower, which encourages design of components with high cohesion (lots of work locally) and low adhesion (selective work remotely).”

Messaging system loosely coupled solution and does not have data inconsistency problem like File transfer. Because message delivering operation is accomplished when the data change is occured. Messaging systems can perform data and function sharing properties like Remote Procedure Invocation(RPC)

method and work asynchronous manner different from RPC (RPC works synchronous manner) (Hohpe & Woolf, 2004)

In messaging systems, message transmission is achieved by five steps (Figure 3.5):

- Creating the message – data with special format
- Sending the message – sender put the message channel
- Deliver the message – channel sends the message to the receiver
- Receive the message – Receiver takes the message from the channel
- Process the message – The message is extracted and data is provided

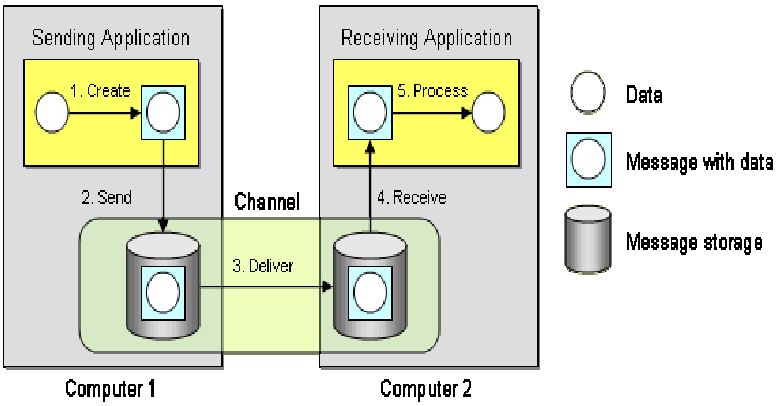

Figure 3.5 Messaging Transmission in Messaging Systems (Hohpe & Woolf, 2004)

4.2.1 Message Types

Data can be exchanged with messages that is wrapped, because message channels transmit data in message format. Messages varies because of the intent of the message. The message types are as follows:

4.1.1.1 Command Message

Some applications may need to execute a procedure that is located in other applications.This can be done with remote procedure call or with messaging systems. With remote procedure call (RPC) provides a method invocation mechanism that is executed synchronously. If the call can not be executed because

of a problem of network or a problem of remote process (not available function at the receiver), caller thread blocks. More appropriate solution is calling the remote process asynchronously like messaging systems. If the caller wants to execute the remote function asynchronously, the sytem keeps trying to execute the command until the remote function executes successfully.

One of the advantage of using messaging system to execute a remote function is using a *local procedure call* not *remote procedure call* across the network. With command message, the request of executing the remote procedure that is in another process is stored in the message as an object. The caller transmits the procedure's invocation as a message. In the receiver side, the command message request is interpreted and the local function is called. The parameters of the procedure that will be executed in receiver side, can be store in the command message too. (In the message state) (Hohpe & Woolf, 2004)

Command messages can be transmitted in Point -to-Point channel, so each request is excuted only one by a receiver.

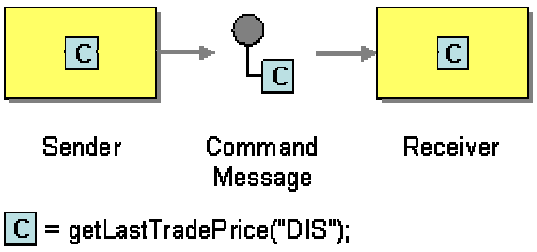

Figure 3.6 The structure of a command message. Command can be used to get last trade price information in an application. Hohpe & Woolf, 2004)

4.1.1.2 Document Message

Some applications may wants to transfer pure data to another without waiting a response or value from the receiver. Sender application does not concern with receiver what to do with that data.In Remote Procedure Call (RPC) and Command Message, caller sends a data and tells what to do with that data. Remote Procedure Call(RPC) also requires a two way communication which is not required with

sending data from one application to another.

Document message sends data to another application and does not concern what the receiver do with that data. The data that is send to receiver can be a single unit of data or a data structure that can be transmitted with more that one document message.

Document message is usually transmitted in Point-to-Point channel to transmit data from one application to another avoing more than one copies in the channel, sometimes document messages are tranmitting in Publish and Subcribe channel to broadcast data to more than one applications. In Publish and Subcribe channel, there are more than one copies of Document message and these must be read-only to avoid updating document message data and different versions of data.(Hohpe & Woolf, 2004)

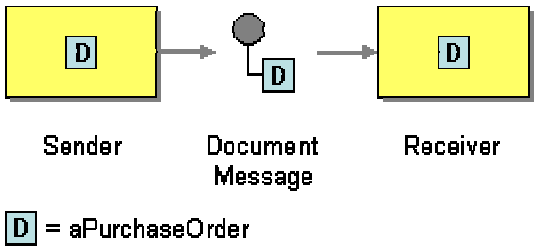

Figure 3.7 The structure of document message. Document message consists of data (information) to inform the receiver. (Hohpe & Woolf, 2004)

4.1.1.3 Event Message

Some application may need to notify another application about the changes. For example: a B2B system may want to notify another application about price changes and may want to send new prices catalog. Remote Procedure Call (RPC) can be used for notifying but in RPC, sender wants the receiver application accepts the event changes immediately.

Event Messages can be appropriate to notify applications asynchronously. With this way, senders sends the change events when they become ready and also receivers receives the change events when they become ready. When sender have changes, it creates an event object, wraps it as an event message and puts it in a

channel. To transmit event messages Publish and Subscribe channels can be used to broadcast messages to the receivers, there is not need to limit the messaging system with Point-to-Point channel.

Event messages are similar to document messages, the main difference is as follows: event message's content is not very important like document messages. Timing is very important for event messages, so message expiration can be helpful for event messages.

In some case, beside the event, information is important. For example, increasing price information is an event and the new price is information about this event. For these situations there two main models:

Push Model: The message is combined event and document message. This method is useful when all receivers needs the information about the event. If all receivers does not needs information, a large message is ignored by many receivers.

Pull Model: There are 3 types of messages in pull model;

1 - Update : Sender sends event messages to the observers about changes in the sender side.

2 - State Request : Observers sends a command message (state request) to have the details about the changes.

3 - State Reply : Sender sends the details of changes to the observers as a document message. (state reply)

Pull model has advantages and disadvantages. Update messages' size is small and only interested observers wants the details of the changes so there are not any needless messages. But on the other hand, additional channels are needed to transmit messages between sender and observer and one change is processed with three messages. (Hohpe & Woolf, 2004)

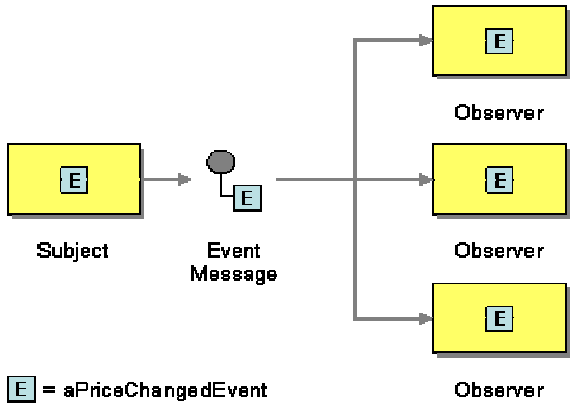

Figure 3.8 The structure of event message. Sender may want to notify receivers about an event, fort his situations event messages are used. (Hohpe & Woolf, 2004)

4.1.1.4 Request - Reply Message

Messaging provides one way communication, but sometimes applications may need to communicate with two way communication. For example: one application may want to invoke a function that belongs to another application, and receiver application returns resulting value to the sender or when an application wants to execute a query at the other application, it sends a command message that consists of query. Receiver side executes the query and returns the result as a document message to the sender. In these situations, request and reply messages are appropriate.

Requestors sends a request message to the receivers within one channel and and receivers responds the request message as a reply message within another channel. Both requestor and replier have seperate channels. Request message can be transmitted with publish and subscribe channel or point to point channel. With point to point channel, one message reaches from one reuqester to one replier and with publish and subcribe channel, one message reaches from one requester to multiple replier (broadcasting)

With Request and Reply architecture,

1. Remote Procedure Call can be implemented, the reqeust message is a

command message that consist of the method that will be invoked by replier and response message is a document message that is consist of the result value of the method invoke.

2. Messaging query can be implemented, the request message is a command message that consists of the query that will be executed in the replier part. Replier part executes the query and send the result of the query to the requestor part

3. Event notification and acknowledgement can be implemented with messaging. The requester sends a request message as a Event Message that consists of information about the event and replier sends a response message as a Document message to acknowledge the events or as Command message to request the other detailes of the event.

In the request and reply mechanism; the request message may consist of a return address to inform the replier about the location of the response message and the reponse message may consist of a correlation identifier to inform the requestor about the request message of the response message. (Hohpe & Woolf, 2004)

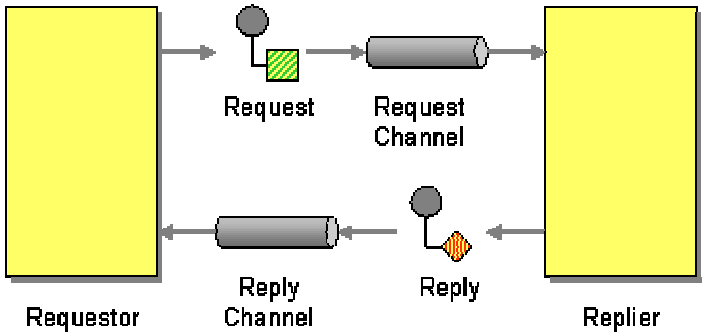

Figure 3.9 The structure of request and reply messages. Requester produces a request message to replier, replier send its reply to requester as a response message. (Hohpe & Woolf, 2004)

5.2 Pipes and Filters

In some cases, a sequence of processing steps must be applied to a message. For example a request message must be encrypted, after that must be authenticated and then must be logged. To accomplish these situations, two solutions can be implemented. The solutions are as follows:

1. Implementing one complete program that consists of all tasks one after other. But this solution is not a flexible one. If one of the messages in the system does not need to authenticate, the program must be changed. These situations will cause the program to change and this change operation cost-effective. This solution also eliminates re-use of components.

2. Dividing each task into separate independent components (filters) and applying them to the messages in a sequence (pipe). The tasks must not call other task to perform the sequence of tasks. With this situation, the tasks will be dependent each other and one task can not be worked by itself. To solve this problem, each task must have a generic interface; with this approach each task can be interchangeable and independent from each other.

In the interface, each filter has inbound port and outbound port. Filters gets messages from inbound port and, processes the message and produces the result, and publishes the result to the outbound port. Output ports and input ports are connected to pipes. Pipes connect to next filter in the sequence and send the output of previous filter to the next filter. Filters have common interface so new filters can be added or existing filters can be removed from the sequence. (Hohpe & Woolf, 2004)

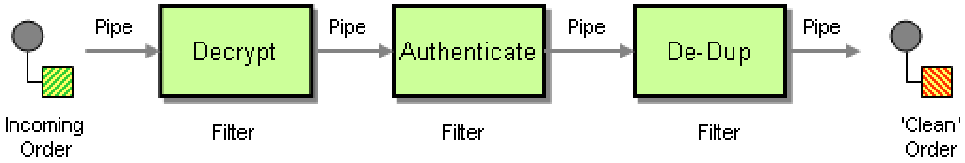

Figure 3.10 Pipes and Filters pattern. The tasks that are located in a sequence are applied to messages. (Hohpe & Woolf, 2004)

6.2 Message Router

The architecture that is used pipes and filters mechanism; filters are connected with fixed pipes with each other. This structure is appropriate if all messages require fixed functions. For example, it is suitable if all messages require firstly decryption operation, later authentication and later logging.

Figure 3.11 explains the structure of message router. It connects to an input channel, consumes messages from this channel and also connects to more than one output channel. According to message properties, Message Router puts message to the appropriate channel.

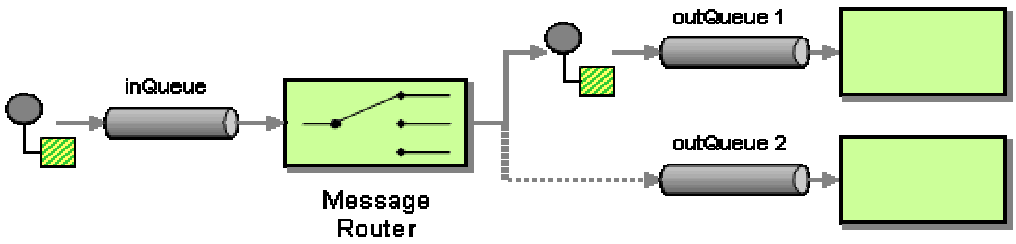

Figure 3.11 Message Router structure.

In message-based architectures messages may require a different set of functions that must be applied so there must be pipes between each combination of filters and each filter is responsible of deciding which filter will be used in the sequence. The figure 3.12 explains the problem:

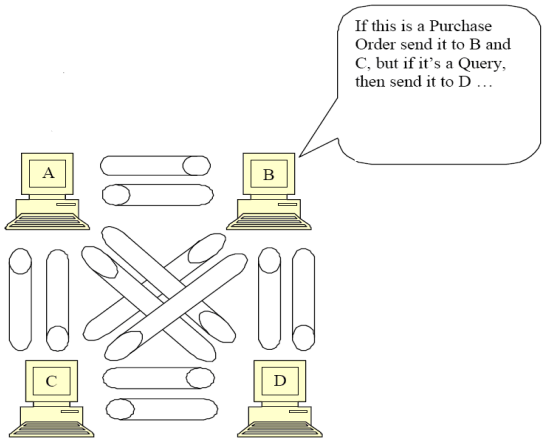

Figure 3.12 The filter that will be applied in the next step is filter's decision there must be combination of pipes between filters. (Hohpe, G, 2002)

To solve this problem, a filter must be located to a central control point to decide which message must be executed in which pipeline. This filter is called “Message Router”. Message router has the message from pipe and decides that the message will be forwarded which filter in the next step according to the message body and header information.

The main advantage of message router is, decision of which filter will be used after previous filter is done in a single point. If there is a change about deciding, it can be done in a single point. Changing only Message router is required because of the decision point is Message Router. (Hohpe, G, 2002)

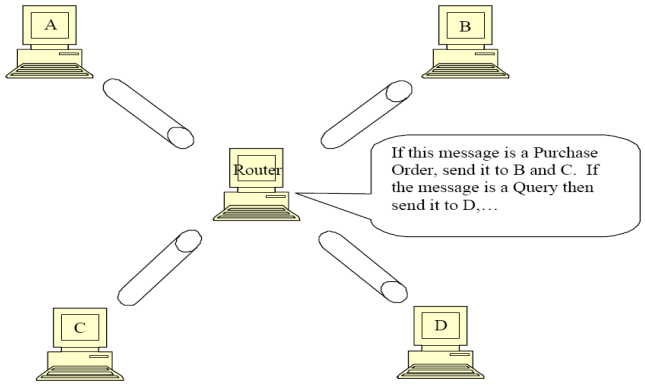

Figure 3.13 There is a central processing unit that decides which filter will be applied to message in the next sequence. (Hohpe, G, 2002)

7.2 Message Dispatcher

Message Dispatcher pattern is used when there is a sender that is sending messaging to a channel and there are multiple performers that are waiting messages to consume them.

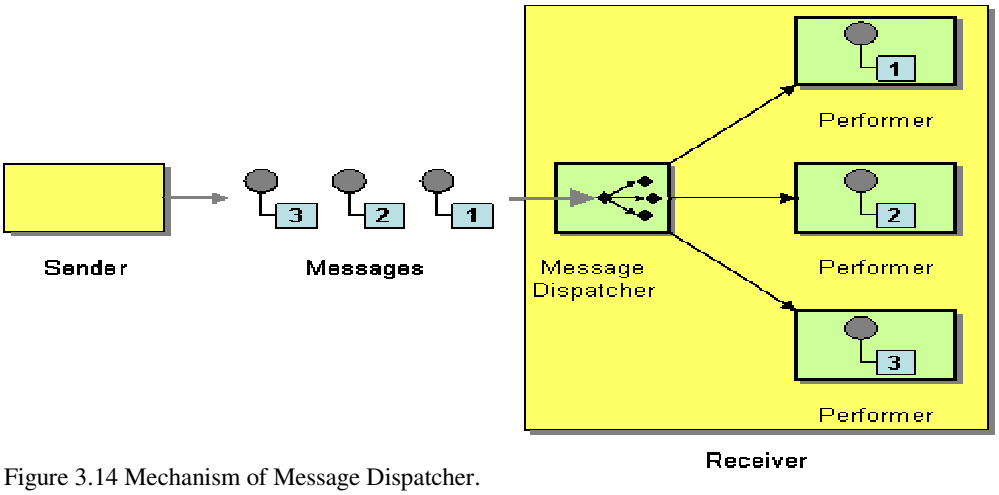

Figure 3.14 Mechanism of Message Dispatcher.

Figure 3.14 explaines the mechanism of message dispather pattern. Message dispather receives a message from the channel and according to message property, it delegates the message to appropriate performer to consume the message.

CHAPTER FOUR

ROTA FRAMEWORK

Rota Framework is a black box framework, because developers of applications that uses Rota Framework does not need to know the internal structure of Rota Framework. They can not extend the features of framework with inheriting classes of framework like white box frameworks. Application developers can extend framework features with standard interfae of Rota Framework. Pipelines can be formed by developers with implementing the interface IPipeline.

Rota Framework is designed by using messaging consepts. Request and Reply messaging type is used to perform the messages of framework. In layered applications that use Rota Framework, layers communicate between each other by using request and response messages so there is a common communication method. Developers know that if there is a request, to execute a service, request messages must be used and appropriate parameters must be assigned to the request message. The developers also know that response message has the result of the service and must be consumed to have the specified result. Transaction dispatcher is built by using Message Dispatcher pattern of Enterprise Integration Patterns. According to the parameters of TxnManager, appropriate service that will be executed is decided. Pipeline Router is designed by using Message Router pattern of Enterprise Integration pattern. According to the parameters of TxnManager, appropriate pipeline, consequently appropriate filters that will be applied is decided.

The components of the rota framework is as follows:

Presentation Layer Components : Web and Desktop Application Controls
Service Layer Components : Service management components and services

The main components and relations of the component are presented in Figure 4.1. Presentation layer components can be used in presentation tier of applications. Service layer components are used to decide which service will be used by using TxnManager and Transaction Dispatcher, which tasks will be applied to messages

by using pipelines, handlers and pipeline router.

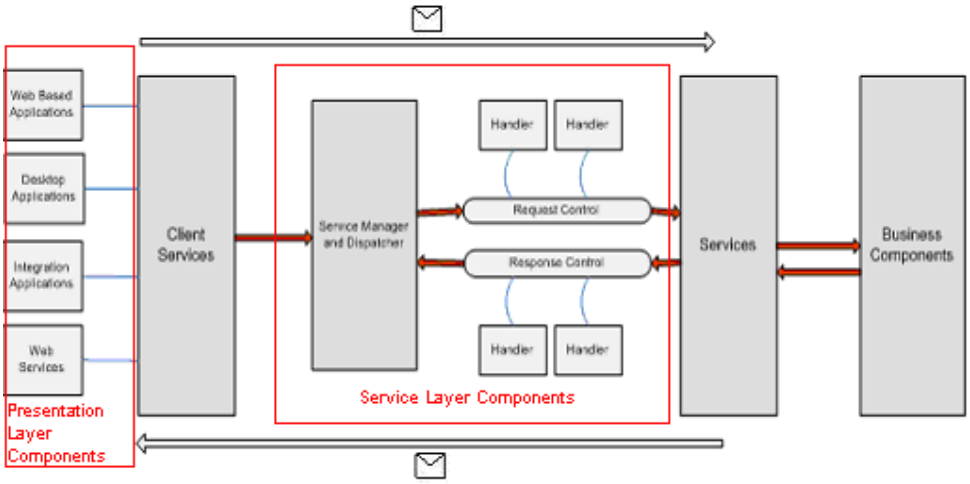

Figure 4.1 The architecture of a software application that is used Rota Framework.

4.1 ROTA Web and Desktop Application User Interface Components

To provide the communication between user and the appliation, the user interfaces are designed and developed in the applications. In different applications, the requirements of users or framework can differ and the developers have to write much code to satify the requirements. To re-use the properties of user interface controls, it is inherited from basic interface that have optimized and common proporties but controls must also have different customized properties too, so the controls are inherited from a basic class and specialized to satify other properties. Figure 4.2 shows IRotaWebControl interface that is used to implement specialized web controls.

```
public interface IRotaWebControl
{
    /// <summary>
    /// web control ünün Request Message da map edeceði yer.
    /// </summary>
    string RequestMessageFieldName
    {
        get;
        set;
    }
    /// <summary>
    /// web control ünün içeriðini Response Message dan alacaðý yer.
    /// </summary>
    string ResponseMessageFieldName
    {
        get;
        set;
    }
    /// <summary>
    /// Web control e girilen data nýn tipi.
    /// </summary>
    DataTypeEnum DataType
    {
        get;
        set;
    }
    /// <summary>
    /// Web control ün otomatik temizlenmesi
    /// </summary>
    bool ClearValueWhenRequested
    {
        get;
        set;
    }
   /// <summary>
   /// Map edilen deðeri control e atar.
   /// </summary>
   /// <param name="mappedValue">Map edilen deðer.</param>
   void SetMappedValue(object mappedValue);
   /// <summary>
   /// Map edilen deðeri control den okur.
   /// </summary>
   /// <returns>Map edilen deðer.</returns>
   string GetMappedValue();

}
```

Figure 4.2 An interface of Rota Web Controls (IRotaWebControl). All of the control that is special for Rota Frameworks implements this interface.

```
namespace Rota.WebControlsBase
{
    /// <summary>
    /// DropDownList kontrolünün Rota Framework için geniþletilmiþ halidir.
    /// IAutoFillableWebControl interface ini implement eder.
    /// </summary>
    public class RotaDropDownList : DropDownList, IRotaWebControl, IAutoFillableWebControl, ISortableWebC
    {
        /// <summary>
        /// otomatik erisilebilirlik degistirme iþlemine dahil olsun mu
        /// </summary>
        private bool changeAccessibilityWhenRequested = true;
        /// <summary>
        /// otomatik temizleme iþlemine dahil olsun mu
        /// </summary>
        private bool clearValueWhenRequested = true;
        /// <summary>
        /// web control ünün Request Message da map edeceði yeri saklar.
        /// </summary>
        private string requestMessageFieldName;
        /// <summary>
        /// web control ünün içeriðini Response Message dan alacaðý yeri saklar.
        /// </summary>
        private string responseMessageFieldName;
        /// <summary>
        /// GetMappedValue metodunun null dönmesine izin verir. Bu þekilde isAssigned false yapýlabilir.
        /// </summary>
        private bool allowNullGetMappedValue;
        [Category("Rota"), Description("change enabled value on automatic setcontrolaccessibilities"), DefaultValue(true)]
        public bool ChangeAccessibilityWhenRequested
        {
            get { return changeAccessibilityWhenRequested; }
            set { changeAccessibilityWhenRequested = value; }
        }
        /// <summary>
        /// control ün içeriði otomatik olarak temizlensin mi
        /// </summary>
        [Category("Rota"), Description("Clear value on automatic clean-up"), DefaultValue(true)]
        public bool ClearValueWhenRequested
        {
            get { return clearValueWhenRequested; }
            set { clearValueWhenRequested = value; }
        }
        /// <summary>
        /// web control ünün Request Message da map edeceði yer.
        /// </summary>
        [Category("Rota"), Description("Request Mapping Parameter")]
        public string RequestMessageFieldName
        {
            get { return requestMessageFieldName; }
            set { requestMessageFieldName = value; }
        }
        public void SetMappedValue(object mappedValue)
        {
            if (mappedValue != null)
            {
                SelectedValueSafe = Convert.ToString(mappedValue);
            }
        }
        /// <summary>
        /// Map edilen deðeri control den okur.
        /// </summary>
        /// <returns>Map edilen deðer.</returns>
        public string GetMappedValue()
        {
            if (AllowNullGetMappedValue && SelectedValueSafe.Length == 0)
                return null;
            else
                return SelectedValueSafe;
        }
```

class System.Convert
(The documentation cache is still being constructed. Please try again in a few seconds.)

Figure 4.3 Sample web control that implements IRotaWebControl. Sample is a specialized DropDownList control called “RotaDropDownList”

In the sample above, dropdownlist control is specialized to accomplish certain tasks to make developers life easy. RequestMessageFieldName property,

GetMappedValue method are used to map data from dropdownlist control to the message automatically and ResponseMessageFieldName property, SetMappedValue method are used to map data from message to the dropdownlist control automatically without spending any extra effort by developers. Developers set only RequestMessageFieldName and ResponseMessageFieldName properties from user interface.

4.1.1 Form manager

Mapping data between messages and control's of the form is belong to the framework's component called form manager. **Form manager** is responsible from the following subjects:

1. Mapping data that is entered by user to the request message and to accomplish this, it has MapControlToRequestMessage method.

2. Mapping data from response message to the form controls to show the result of the request to the user and to accomplish this, it has MapResponseMessageToControl method.

3. Displaying notifications that is the result of service to the user and it has DisplayNotifications method to accomplish this.

The reponsibilities can be accomplished by Form Manager component if develeopers of the application use control in the user interface that are inherited from IRotaWebControl that is mentioned in section 4.1. If not automatic data mapping between user interface controls and messages can not be provided.

```
private void MapControlToRequestMessage(Control control, ref RequestMessage requestMessage)
{
    if (control is IRotaWebControl)
    {
        IRotaWebControl rotaWebControl = (IRotaWebControl)control;

        if (rotaWebControl.RequestMessageFieldName != null)
        {
            if (rotaWebControl.RequestMessageFieldName.Length > 0)
            {
                string value;

                value = rotaWebControl.GetMappedValue();

                if (value != null)
                {

                    SetPropertyOrFieldValue(requestMessage, rotaWebControl.RequestMessageFieldName, value);

                }
            }
        }
    }

    if (control.Controls.Count > 0)
    {
        foreach (Control childControl in control.Controls)
        {
            MapControlToRequestMessage(childControl, ref requestMessage);
        }
    }
}

private void MapResponseMessageToControl(ref ResponseMessage responseMessage, Control control)
{
    if (control is IRotaWebControl)
    {
        IRotaWebControl rotaWebControl = (IRotaWebControl)control;

        if (rotaWebControl.ResponseMessageFieldName != null)
        {
            if (rotaWebControl.ResponseMessageFieldName.Length > 0)
            {
                object value = null;
                value = GetPropertyOrFieldValue(responseMessage, rotaWebControl.ResponseMessageFieldName);

                if (value != null)
                {
                    rotaWebControl.SetMappedValue(value);
                }
            }
        }
    }
    if (control.Controls.Count > 0)
    {
        foreach (Control childControl in control.Controls)
        {
            MapResponseMessageToControl(ref responseMessage, childControl);
        }
    }
}
```

Figure 4.4 The implementation of automatic mapping from controls to message and message to control.

4.2 Rota Messages

The main property of rota framework is using messages to communicate between layers so messages are very important in rota framework.

Messages are created with Request-and-Reply principle that is mentioned in the Message Types section of this document. In the request and reply pattern, requestor sends a request message to the receiver to have a response message.

Receiver accepts the request message of the requestor and accomplishes the specified task and returns the response to the requestor.

Figure 4.5 explaines main structure of the messages that is used in Rota Framework. Request and Response message are inherited from MessageBase and every message has these properties.

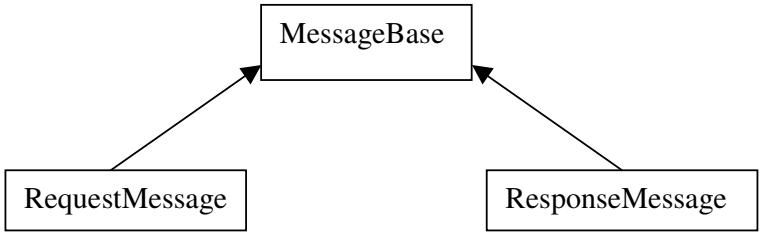

Figure 4.5 Message structure of Rota Framework

4.2.1 Message Base

MessageBase has properties that are common in request and response messages. Every message (both request and response message) has unique identifier called messageId, the property that is used to decide whether the message will be logged or not called LogMessage and service name that will be processed or that was processed called TransactionName. MessageBase properties are shown is the following figure:

Figure 4.6 The properties of Message Base. Request and Response message are inherited from MessageBase and every message has these properties.

4.2.2 Request Message

RequestMessage is inherited from MessageBase so it has all properties of MessageBase, but it also has additional properties;

- ***UserId :*** UserId property is used to specify the user that uses the service. This information is very important in the application because if the operation is not a valid, the user that does this operation can be found easily

- ***Channel:*** Channel property of request message is used to specify the origin of the request. Channel property can have the value of Channel type enumaration value. Channel type enumaration is consists of Web, WebService, WindowsApplication values.

- ***PipelineType:*** Pipeline type property is used to specify the pipeline type of the request message. According the value of pipeline type, the specific operations can be done to the request message. For example in the DefaultPipeline, autorization is applied to the user that is located in the request message but in the FetchPipeline, autorization is not applied.

The properties of *RequestMessage* is shown in the following figure:

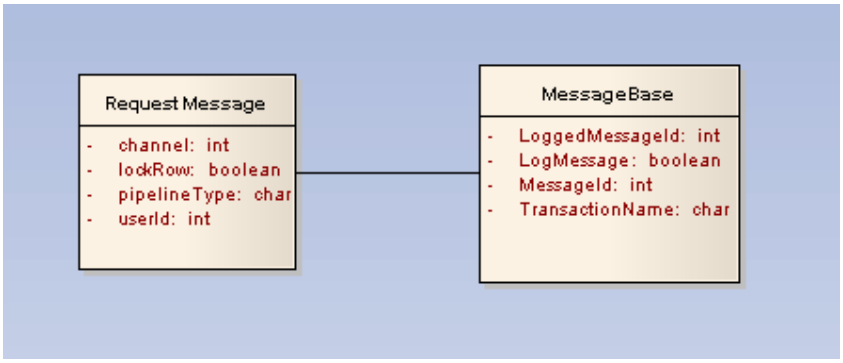

Figure 4.7 The structure of Request Message. Request message has properties that are shown in the figure additional to MessageBase properties.

4.2.3 Response Message

ResponseMessage is inherited from MessageBase like RequestMessage and

also has additional properties too.

- ***ResponseStatus:*** ResponseStatus property of the response message is used to provide the status of the operation at the service. (transaction) It can have one of the values of the ResponseStatus enumaration. The possible values of the ResponseStatus enumaration are Successful, Error, CatastrophicError.

- ***Notifications:*** Notifications property of the ResponseMessage is consists cf notifications that are produced after the service execution. Notifications in the notifications property of the response message can be info, error or warning notifications and these notifications are shown to user at the user interfaces.

- ***RequestMessageId:*** This property of response message specifies which request message belongs to response message. It has the value of unique identifier of request message.

- ***RedirectionUrl:*** This property is used to specify which url the application will be directed when an event occurs. For example the application sends a request to execute a service (transaction) in the service layer, it is executed and an exception occurs in the service layer. For this situation, an error notification is added to the notifications property of response message and RedirectionUrl property is set to the ErrorPage of the application to inform the user expactedly.

The properties of *ResponseMessage* is shown in the following figure:

```
public enum ResponseStatus
{
    Successful = 0,
    Error = 1,
    CatastrophicError = 2
}

public abstract class ResponseMessage : MessageBase
{
    private NotificationList notifications;
    private ResponseStatus status = ResponseStatus.Successful;
    private Guid requestMessageId;
    private string redirectionUrl;

    public ResponseMessage()
    {
        notifications = new NotificationList();
    }

    public NotificationList Notifications...

    public ResponseStatus Status...

    public string RequestMessageId...

    public string RedirectionUrl...

    public void SetSuccess()...

}
```

Figure 4.8 The implementation of Response message. Response message has properties that are shown in the figure additional to MessageBase.

RequestMessage and ResponseMessage is a template messages for the they are the header part of the application specific messages. Application specific messages are inherited from Request and Response message of Rota Framework. While developing an application, data that is necessary to execute the service (transaction) is located to the body part of the message.

4.3 Rota Service Management

In applications, appropriate solution is designing independent services that are responsible with implementing business work. Services must have the following features:

- Services do not need to know which application or which interface (form) consume it.
- Services must be loosely coupled and must not use another service.

In Rota Framework, services are called “Transactions”. Transactions that are responsible to accomplish the business work in application are managed and executed by rota framework with a specialized component “*TxnManager*”.

4.3.1 TxnManager

TxnManager is one of the Rota Framework component that is responsible being a bridge between the Presentation and Service Layer of an application. TxnManager is very important component for applications that use Rota Framework.

TxnManager has properties, events and methods. TransactionGroupName, TransactionModuleName and TransactionName properties are necessary to execute the required service (Section 3.5) and AutomapRequestMessage, AutoMapResponseMessage, AutoDisplayNotifications properties are used to specify mapping type (manual or automatic) MakeEventBasedRequestMapping and MakeEventBasedRequestMapping events are used to map request and response message manually.

The features of "TxnManager" component are as follows:

- ***Validating Parameters:*** TxnManager is responsible to validate the paramenters to execute the service. One of the parameter that must be validated by TxnManager is TransactionName parameter. This parameter represents the service name that will be executed and it must not be empty.

- ***Creating Request Message:*** Creating request message is loading the assembly of the application specific message and producing an object of the specified message.

As mentioned earlier, TxnManager is one of the component that is a bridge between user interface and service layer of an application. It gets the needed parameters from interface, validates and sends them to the TransactionsDispatcher that is infactly responsible creating the appropriate repuest message object according the parameters of TxnManager.

- ***Preparing Request Message:*** After creating request message object, TxnManager prepares the request message. There is two way to prepare the request message:
  - *Automapping the request message.*
  - *Making event based request message mapping*

The values of parameters that is needed to execute the appropriate service can be automatically mapped by FormManager component of Rota Framework (Section 4.1.1) or it can be mapped manually by using the mapping events of TxnManager component. Mapping event of the TxnManager is as follows:

```
public void PrepareRequestMessage(RequestMessage requestMessage)
{

    if (AutoMapRequestMessage)
    {
        if (formManager != "")
        {
            IFormManager formManagerObject = (IFormManager)this.FindControl(formManager);
            formManagerObject.MapControlToMessage(Page, ref requestMessage);
        }
        else
        {
            IFormManager formManagerObject = new FormManager();
            formManagerObject.MapAllToMessage(Page, ref requestMessage);
        }
    }

    FillRotaHeader(requestMessage);

    MakeEventBasedRequestMapping(requestMessage);
}
```

Figure 4.9 “TxnManager” prepares the request message and the figure shows the implementation of this. Automatic mapping (by using for manager) or event based mapping can be used to prepare the request message.

At the user interface of an applications to map the request message manually appropriate method is written. If AutoMapRequestMessage property of TxnManager is set to false value, MakeEventBasedRequestMapping event is fired by RotaFramework and the method that is responsible to map the request message manually works.

- ***Filling Header Information of Message***: While preparing request message, TxnManger component maps data that is consumed from user interface controls by using FormManager component. After mapping body of request message, TxnManager fills the header information of request message. Active user information, PipelineType and ChannelType properties are header part of request

message. Active user information is mapped to request message from Session and other properties are filled from user interface by developer.

- ***Executing Service:*** Executing required service is loading application spesific service assembly and invoking appropriate method of that service.

- TxnManager takes the required parameter from user interface and redirects them to TransactionDispatcher and then Transaction Dispather loads service layer assembly and executes the service.

- ***Consuming Response Message:*** After executing the service, a response message is produced and that must be redirected to the user interface to show result of the service to user. Like preparing request message, consuming response message can be done with two method: Mapping response message to the controls (user interface controls) automatically or manually. Again automatic mapping is accomplished by FormManager and manual mapping is done with event of TxnManager component.

```
public void ConsumeResponseMessage(ResponseMessage responseMessage)
{
    if (responseMessage.Status == ResponseStatus.Successful)
    {
        if (AutoMapResponseMessage)
        {
            if (formManager != "")
            {
                IFormManager formManagerObject = (IFormManager)this.FindControl(formManager);
                formManagerObject.MapMessageToControl(ref responseMessage, Page);
            }
            else
            {
                IFormManager formManagerObject = new FormManager();
                formManagerObject.MapMessageToAll(ref responseMessage, Page);
            }
        }

        MakeEventBasedResponseMapping(responseMessage);
    }
}
```

Figure 4.10 After executing the appropriate service (transaction), "TxnManager" consumes the response message and the figure shows the implementation of this. Consuming response message can be done with automatic mapping the response message by using form manager component or with event based mapping.

At the user interface of an applications to map the response message manually appropriate method is written. If AutoMapResponseMessage property of

TxnManager is set to false value, MakeEventBasedResponseMapping event is fired by RotaFramework and the method that is responsible to map the response message manually works.

- ***Displaying Notifications:*** After executing the appropriate service, a response message consists of the result of the service. While excuting the service an exception can occur or an info or warning message can be produced. Error, warning and info messages are stored in notifications property of response message (Section 4.2.3) . Notifications that are stored in the response message must be shown to user to inform about the service execution status. This is accomplished by TxnManager component by using Form manager component.

4.3.2 Transaction Dispatcher

"TransactionDispather" is responsible the managing the services in RotaFramework. TxnManager component is a bridge between user interface and service layer as mentioned section 4.3.1, TxnManager redirects the parameters to TransactionDispatcher that are necessary to create request message and execute services.

After having appropriate parameters, TransactionDispatcher loads application specific request message assembly to create request message object and and loads application specific service(transaction) assembly and invokes specified method to execute the service and it also handles the exception that is occurred while the service executing.

TransactionDispatcher can create request message or execute specified service if there are some rule while developing an application. The rules must be as follows:

- **Rule 1.** There must be a naming convention in messages and transactions project. The name of the messages project must be ended with the same value of ***MessagesDefaultGroupName*** property and name of the transactions project must be ended with the same value of ***TransactionDefaultGroupName*** property. There

can be a prefix of the applications module name, if the application is not consist of modules, it can be project's name too.

If TransactionDispatcher's MessagesDefaultGroupName property is equal to ".Messages" and TransactionDefaultGroupName property is equal to ".Transactions", the names of Message and Transaction projects must as follows: Locations**.Messages**, Locations**.Transactions**

- **Rule 2.** There must be a naming convention when creating Request, Response messages and transactions. The name of Request message must be ended with the same value of RequestMessageSuffix property, the name of response message must be ended with the same value of ResponseMessageSuffix property and the name of transactions must be ended with the same value of TransactionSuffix property of TransactionDispatcher.

If TransactionDispatcher's RequestMessageSuffix property is equal to "Request" and ResponseMessageSuffix is equal to "Response" and TransactionSuffix is equal to "Transaction", the following names are valid for request, response messages and transactions:

AddLocation**Request,** AddLocation**Response,** AddLocation**Transaction**

- **Rule 3.** Service name and its request response message names must have the same value. For example the request, response and transaction names that are used to list customers must be as follows: **ListCustomers**RequestMessage, **ListCustomers**ResponseMessage, **ListCustomers**Transaction

The properties of TransactionDispatcher are as follows:

- ***MessagesDefaultGroupName:*** This property is used to load the request message assembly to create request and response message. Generally, it has "Messages" value.

- ***TransactionDefaultGroupName:*** This property is used to load the service project assembly to execute a service. Generally it has “Transactions” value.

- ***TransactionExecutingMethodName:*** This property is used to execute a service. After loading the assembly of service project, a method that is specified with *TransactionExecutingMethodNameis* invoked to execute the service. Generally it has “Execute” value.

- ***RequestMessageSuffix:*** This propety is used to specify the name of the request message to create it. TxnManager redirects the name of the service that will be executed and rule 3 says that “*Service name and its request response message names must have the same value*”. So the request message’s class name is provided by the name of transaction and *RequestMessageSuffix* property value. If the name of the transaction is “AddCustomer” and the suffix is “Request”, the name of the request message is “AddCustomerRequest”

- ***ResponseMessageSuffix:*** This propety is used to specify the name of the response message to create it. As a rusult of transaction name is the same of its messages, response message’s class name is provided by the name of transaction and *ResponseMessageSuffix* property value. If the name of the transaction is “AddCustomer” and the suffix is “Response”, the name of the response message is “AddCustomerResponse”

- ***TransactionSuffix:*** This propety is used to specify the name of service that will be executed by TransactionDispatcher. The name of the transaction that will be redirected by TxnManager and the value *TransactionSuffix* property provides the class name of the trasaction that will be executed. If that name of the transaction is AddCustomer and the suffix is “Transaction”, valid name of the service is “AddCustomerTransaction”.

While creating request and response message and executing transactions, TransactionDispatcher uses “***Reflection***”. TransactionDispatcher creates request

message, creates transaction to execute and creates response message that is the result of executing transaction with reflection .

```
private static ResponseMessage Execute(string transactionModuleName, string transactionGroupName, string transactionName,RequestMessage requestMes
{
    ResponseMessage responseMessage = CreateResponseMessage(transactionModuleName, transactionGroupName, transactionName, requestMessage.MessageId
    PipelineRouter pipelineRouter = new PipelineRouter();
    try
    {
        pipelineRouter.ExecutePre(requestMessage, responseMessage);
        ExecuteTransaction(transactionModuleName, transactionGroupName, transactionName, requestMessage, responseMessage);
    }
    catch (Exception ex)
    {
        if (ex is IRotaException)
        {
            HandleError(responseMessage, (IRotaException)ex);
        }
        else if (ex.InnerException is IRotaException)
        {
            HandleError(responseMessage, (IRotaException)ex.InnerException);
        }
        else if (ex.InnerException != null)
        {
            if (ex.InnerException.Message.Contains("ORA-02292"))
            {
                HandleFKConstraintError(responseMessage, ex);
            }
            else
            {
                responseMessage.Status = ResponseStatus.CatastrophicError;
                ExceptionHelper.SendExceptionAsMail(requestMessage.ToString(), responseMessage.ToString(), ex);
                throw;
            }
        }
        else
        {
            responseMessage.Status = ResponseStatus.CatastrophicError;
            ExceptionHelper.SendExceptionAsMail(requestMessage.ToString(), responseMessage.ToString(), ex);
            throw;
        }
    }
    finally
    {
        pipelineRouter.ExecutePost(requestMessage, responseMessage);
    }

    return responseMessage;
}
```

Figure 4.11 Execution of a service

As shown in the figure 4.10, before executing transaction that is specified with transactionName, some operations must be applied to request message. For example request message must be autorized before executing. These operations are done with a specialized structure "*PipelineRouter*".

4.3.3 PipelineRouter and Pipelines

Pipelines are structures that are implemented to control request and response message in a sequence of specific tasks. Before executing the service, the request message must be controlled or after executing the service some tasks must be applied to response message.

Pipelines has different handlers that have different tasks. Logging, Autorization, Auditing, Locking, Exception Handling operations have separate handlers and Pipelines can have these handlers in different sequence and existence.

There are different pipelines for different requirements of applications. In some situations, some request messages may need to be autorized but some request messages may not. For these situations, there are different pipelines.

Every handler can implement its task in its context and can be thougth as a closed box. It does its work by itself and produces required result. It autorizes the request message and if the request is autorized pipeline passes another handler in the sequence, if the request is not autorized, handler throws appropriate exception.

Pipelines have following tasks :

1. Controlling Request message before execution of a service. This is accomplished by ExecutePre method.

2. Controlling response message that is produce after execution of a service. This is accomplished by ExecutePost method.

These methods are common method for all pipeline and these are represented with a common interface called IPipeline. Every pipelines that are specialized for requirements of applications must implement IPipeline interface.

Figure 4.12 shows a sample pipeline implementation. To control the request message ExecutePre method is used and Authorization, Logging, Auditing and locking control are applied to request message. To control response message, ExecutePost method is used and exception handling, integration handling, logging and e-mail sending controls are applied to response message.

```
namespace Rota.Pipelines
{
    public interface IPipeline
    {
        void ExecutePre(RequestMessage requestMessage, ResponseMessage responseMessage);

        void ExecutePost(RequestMessage requestMessage, ResponseMessage responseMessage);
    }
}
```

```
class DefaultPipeline : IPipeline
{
    private void Handle(IMessageHandler messageHandler, RequestMessage requestMessage, ResponseMessage responseMessage)
    {
        messageHandler.HandleMessage(requestMessage, responseMessage);
    }
    public void ExecutePre(RequestMessage requestMessage, ResponseMessage responseMessage)
    {
        Handle(new RequestAuthorizationHandler(), requestMessage, responseMessage);
        Handle(new RequestLoggingHandler(), requestMessage, responseMessage);
        Handle(new AuditHandler(), requestMessage, responseMessage);
        Handle(new LockingHandler(), requestMessage, responseMessage);
    }
    public void ExecutePost(RequestMessage requestMessage, ResponseMessage responseMessage)
    {
        Handle(new OracleExceptionHandler(), requestMessage, responseMessage);
        Handle(new IntegrationHandler(), requestMessage, responseMessage);
        Handle(new ResponseLoggingHandler(), requestMessage, responseMessage);
        Handle(new EmailSendingHandler(), requestMessage, responseMessage);
    }
}
```

Figure 4.12 A sample pipeline implementation.

According to PipelineType field of request and response message, pipeline router decides which request must be process in which pipeline. (Section 2.3) MessageRouter performs deciding appropriate pipeline type with reflection. It loads assembly that consists of different pipelines, it uses PipelineType property to select appropriate pipeline. After selecting a pipeline, message has processed by handler that are locate d in the selected pipeline.

Figure 4.13 explains the interaction between transaction dispatcher, pipeline router and pipelines. Before executing the appropriate transaction, transaction dispatcher calls the ExecutePre method of Pipeline Router to control the request message. According to the request message's PipelineType property, appropriate pipeline is provided by PipelineRouter and appropriate pipeline controls the request message. After this, transaction is executed and produces a response message. Response message is also controlled by calling PipelineRouter's ExecutePost method.

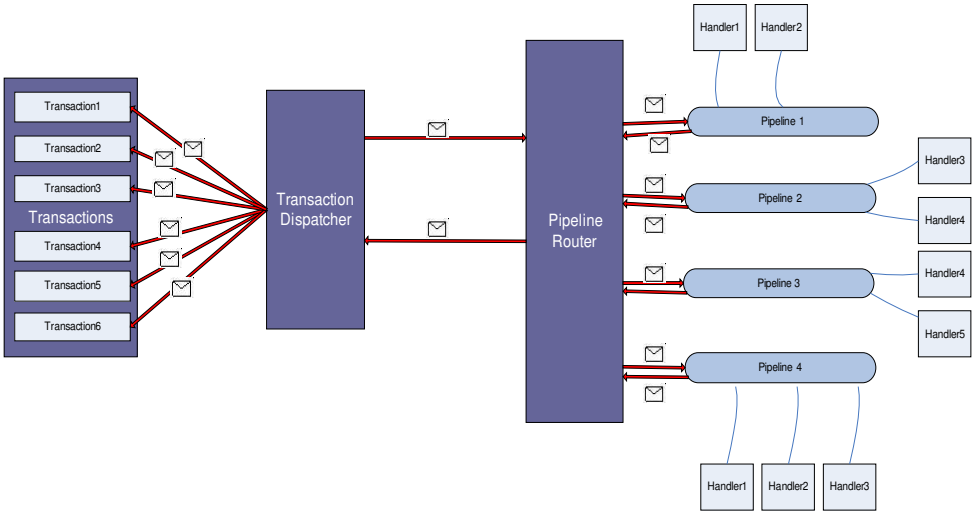

Figure 4.13 The interaction between transaction dispatcher, pipeline router and pipelines.

In the PipelineRouter, GetPipeline method firstly loads the application specific Pipeline assembly that consists of application specific pipelines. After loading assembly according to the value of PipelineType property, appropriate pipeline is invoked by pipeline router. Figure 4.14 shows these operations.

```
public  class PipelineRouter
{
    private IPipeline GetPipeline(RequestMessage requestMessage, ResponseMessage responseMessage)
    {
        PipelineSettings pipelineSettings = new PipelineSettings();
        Assembly pipelineAssembly = Assembly.Load(new AssemblyName(pipelineSettings.AssemblyName));
        if (pipelineAssembly == null)
        {
            throw new PipelineCreationException(String.Format("{0} isimli mesaj assembly bulunamadý. Projeye referans verdiðinizden emin olun",
        }
        string pipelineTypeName=String.Format("{0}.{1}", new object[] { pipelineSettings.AssemblyName, requestMessage.PipelineType});
        Type pipelineType = pipelineAssembly.GetType(pipelineTypeName);
        if (pipelineType == null)
        {
            throw new PipelineCreationException(String.Format("Pipeline tipi {0} bulunamadý, lütfen pipeline isminin doðru olduðundan emin olu
        }
        Object pipeline = pipelineType.InvokeMember(null,
            BindingFlags.DeclaredOnly | BindingFlags.Public | BindingFlags.Instance |
            BindingFlags.CreateInstance, null, null, null);
        if (pipeline == null)
        {
            throw new PipelineCreationException(String.Format("Pipeline tipi {0} türünde yeni bir instance yaratýlamadý", pipelineTypeName));
        }
        return (IPipeline)pipeline;
    }
```

Figure 4.14 The implementation of GetPipeline method of PipelineRouter. It uses “reflection” to specify the pipeline.

CHAPTER FIVE

BOOK STORE APPLICATION

Book Store is an application that is developed by using Rota Framework. Rota Web Application User Interface components are used to implement user interfaces of Book Store application. Especially RotaTextBox, RotaDropDownList, RotaLabel(that are a special edition of asp.net control) and TxnManager controls take a place in Book Store application.

5.1 Responsibilities of Book Store Application

The responsibilities of Book Store application is as follows:

- Searching a publication by using categories of publications or by using a detailed search. In detailed search; book author, book publisher, book category, name, price, edition and printing date are some of the searching criteria of detailed search.

- Performing administrative operations. In this scope; administrator can make the following tasks:
  - Adding, deleting, updating and searching a publisher
  - Adding, deleting, updating and searching an author
  - Adding, deleting, updating and searching a publication
  - Adding, deleting, updating and searching a category
  - Manipulating user operations.

Figure 5.1 shows the main page of Book Store application. Figure 5.2 shows one of the administrative option; Publisher operations, figure 5.3 shows author operations and figure 5.4 shows publication operations. Adding, deleting, updating and listing publishers can be done with this interface.

Figure 5.1 The main appearance of online book store.

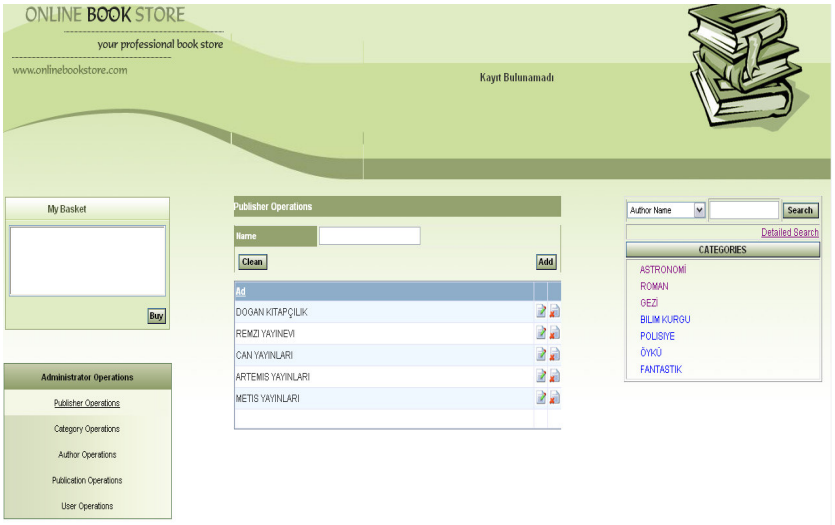

Figure 5.2 Publisher operations.

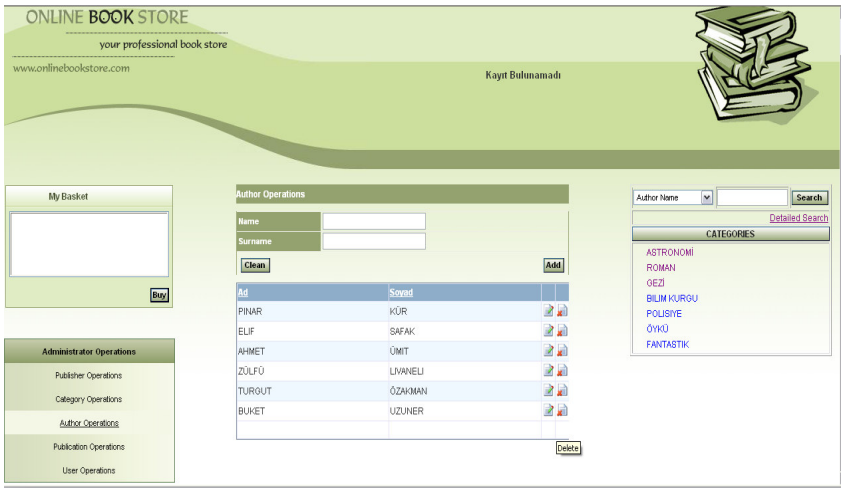

Figure 5.3 One of the administrative option; Author operations. Adding, deleting, updating and listing authors can be done with this interface.

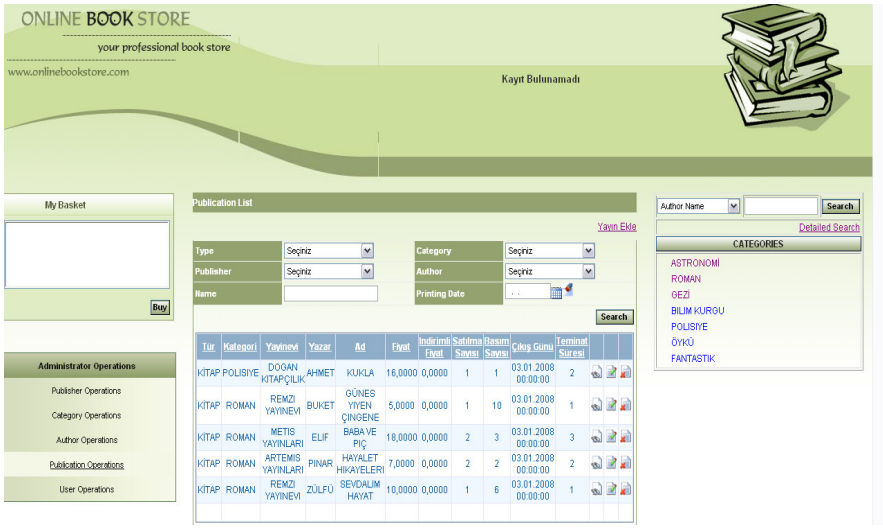

Figure 5.4 One of the administrative option; Publication operations. Adding, deleting, updating and listing publications can be done with this interface.

5.2 The Architecture of Book Store Application

There are three layers in the architecture of BookStore application(Figure 5.5) Presentation Layer that contains user interface of book store application, Business Layer that contains business logic and validation operations, Data Access Layer that contains operations to achieve database operations.

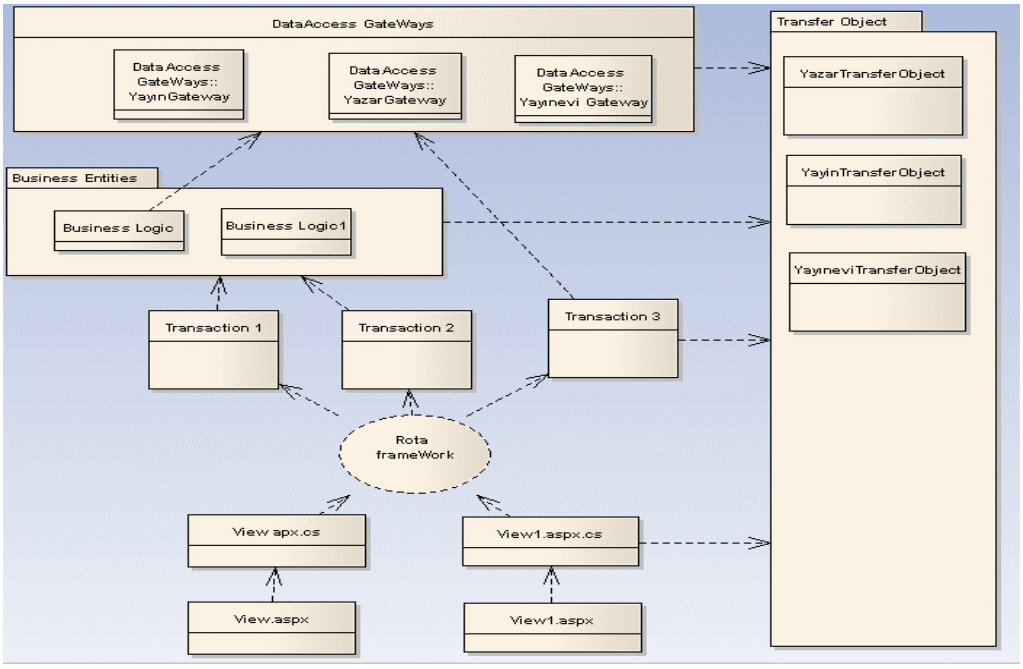

Figure 5.5 The architecture of Book Store Application

In Figure 5.5; there are four layers that are separately responsible with different tasks.

1. Presentation Layer of Book Store Application

In the presentation layer, user interface operations are accomplished. The user of Book Store application makes a searching book request and the inner layers of Book Store application replies the request of the user and presentation layer shows the results of the request (Books) to the user.

2. Business(Transaction) Layer of Book Store Application

In the service layer, Book Store application's business logic and validation operations are implemented. Listing books, adding a selected book to a basket of the user, adding a publisher to the book store system are sample business logic of the system. To execute some of business logics validations must be applied, for example to add a book to a basket user must be enter to the book store system and must select a book and to add a publisher to the book store system a name of a publisher must be entered by administrator. These validations are applied at the business logic of Book Store application.

Business logic of Book Store application formed as a Transactions project that is mentioned in Figure 5.6.

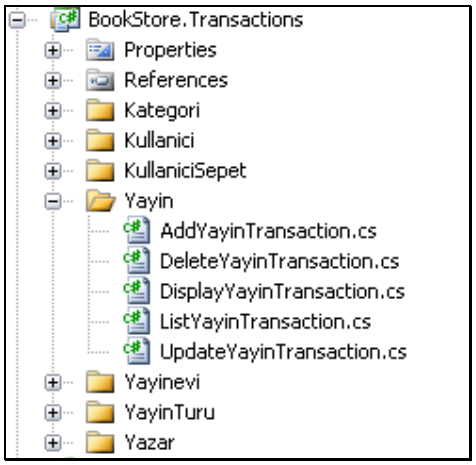

Figure 5.6 Business Layer of Book Store application.

```
public class AddYayinTransaction
{
    public void Execute(AddYayinRequest addYayinRequest, AddYayinResponse addYayinResponse)
    {
        YayinGateway yayinGateway = new YayinGateway();
        yayinGateway.InsertYayin(addYayinRequest.YayinTransfer);

        addYayinResponse.Notifications.Add(NotificationType.Info, "İşleminiz Başarı ile Gerçekleştirilmiştir.
    }
}
```

Figure 5.7 One of the business layer objects of Book Store application. "AddYayinTransaction" class of Business Layer of Book Store application. This class accomplishes adding publication operation.

Book Store application Business Layer communicates with presentation layer and data access layer with request and reply messages that are mentioned in 3.1.4. section of this document.

Request and reply messages store header information and application specific information as a class called "…TransferObject". There is a transfer object for each object in application. There is one for publication, one for publisher and one for author so on.

Transfer objects are filled with information that is entered by user that uses presentation layer and located to an appropriate message to transmit it to business layer. Business layer gets the message and parses transfer object from the message and uses it to accomplish the specified operation.

3. Data Access Layer of Book Store Application

In stead of putting data access operations into the presentation layer, data access layer and presentation layer is separated from each other in the Book Store Application. Consequently, this structures tightly couples presentation layer and data access layer.

Data access layer contains data "Gateways" that are responsible from doing basic data operations like retrieving, inserting, updating, deleting data and also contains "TransferObjects" that represent the tables of database. Gateways perform the appropriate operation and return a TransferObejct or a list of TransferObjects.

The structure of an example gateway is as follows;

```
public class KategoriGateway
{
    private SqlConnection kategoriConnection;
    private KategoriTransferObject _kategoriTransferObject;

    Constructors

    #region ReturnKategoris
    public List<KategoriTransferObject> ReturnKategori(KategoriTransferObject kategoriTransferObject)...

    private List<KategoriTransferObject> CreateAndReturnKategoriList(DataTable kategoriDataTable)...

    #endregion

    #region InsertAKategori

    public void InsertKategori(KategoriTransferObject kategoriTransferObject)...

    #endregion

    #region UpdateAKategori

    public void UpdateKategori(KategoriTransferObject kategoriTransferObject)...

    #endregion

    #region DeleteAKategori

    public void DeleteKategori(KategoriTransferObject kategoriTransferObject)...

    #endregion
```

Figure 5.8 One of the Data Access layer objects of Book Store application. "KategoryGateway" class has main database operation to insert, update, delete and list categories.

In a gateway, there is a connection to connect the database and a transfer object that represents the corresponding database table. Insert, update and delete methods of gateway accept transfer object as a parameter to keep the information to insert, to update and to delete.

Data access layer returns the response of the requested operation to the appropriate business layer object and business layer object responses to the presentation layer as a response message.

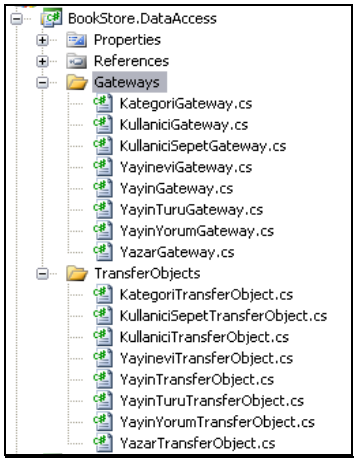

Figure 5.9 Data Access Layer of Book Store application.

5.3 Implementation of Book Store Application

At the implementation stage of Book Store application, the architecture that is mentioned in section 5.2 is implemented and Rota Framework that is explained in Chapter 4 is used.

In Book Store application, Microsoft SQL Server 2005 is used as a Database Management System, Microsoft Visual Studio 2005 is used to develop application.

5.3.1 *Web Application User Interface Components Implemantation*

ROTA Web Application User Interface Components are used in the Book Store application. RotaTextBox, RotaDropDownList and RotaLabel are some of the controls that are used to implement user interfaces of Book Store application.

Automatic mapping, (from controls to messages or from messages to controls) and filling controls properties are used while implementing Book Store application. Figure 5.10 shows “RotaTextBox” control usage, automatic mapping can be done with RotaTextBox control. RequestMessageFieldName property is used to map data from controls to request message. In the following figure, RequestMessageFieldName property has “YayinTransfer.Ad” value where “YayinTransfer” is the transfer object of YayinEkleRequestMessage. ResponseMessageFieldName property is used to map data from response message to web controls. In the following figure, “Ad” field of “YayinTransfer” object of response message (DisplayYayinResponse) is automaticaly mapped from response message to “YayinAdTextBox” control.

Figure 5.11 shows “RotaDropDownList” control usage. Like “RotaTextBox” control automatic mapping is achieved by using “RequestMessageFieldName” and “ResponseMessageFieldName” properties. Additionaly; automatic filling operation can be accomplished by “RotaDropDownList” control. “AutoFill” property is used to specify whether automatic filling will be used or not. “AutoFillTransactionGroupName”, “AutoFillTransactionModuleName” and “AutoFillTransactionName” properties are used to locate and find the transaction

that will be executed to fill “CategoryDropDownList” control. “AutoFillResponseMessageListName” is used to specify which field of response message will be mapped to the control to fill it. “AutoFillListFieldForText” and “AutoFillListFieldForValue” properties are used to represent data that is filled to control automaticaly.

```
<td class="tdinput" align="left" style="width: 30%">
    <rota:RotaTextBox runat="server" ID="YayinAdTextBox" RequestMessageFieldName="YayinTransfer.Ad"
        ResponseMessageFieldName="YayinTransfer.Ad" CssClass="input" Width="70%"></rota:RotaTextBox>
</td>
```

Figure 5.10 TextBox web control that is called “RotaFramework” and is specialized in Rota Framework. With this control automatic mapping can be done.

```
<td class="tdlabel" align="left" style="width: 20%">
    <asp:Label runat="server" ID="CategoryLabel" Text="Category"></asp:Label>
</td>
<td class="tdinput" align="left" style="width: 30%">
    <rota:RotaDropDownList ID="CategoryDropDownList" AutoFill="true" AutoFillResponseMessageListName="KategoriList"
        AutoFillListFieldForText="Ad" AutoFillListFieldForValue="Id" AutoFillTransactionGroupName="Kategori"
        AutoFillTransactionModuleName="BookStore" AutoFillTransactionName="ListKategori"
        Width="70%" CssClass="input" runat="server">
    </rota:RotaDropDownList>
</td>
```

Figure 5.11 DropDownList web control that is called “RotaDropDownList” and is specialized in Rota Framework. Automatic mapping and filling control operations can be done with this control.

5.3.2 *Message Implementation*

Request and reply messages are used to develop Book Store application. Application specific messages are inherited from frameworks based messages. (RequestMessage and ResponseMessage) Framework based messages consist of header information that is mentioned in section 4.3.1 of this document. Application specific message that is explained in Figure 5.12 consist of application spesific data that is used to accomplish spesific task like “ListYayinevi”.

```
using System;
using System.Collections.Generic;
using System.Text;
using Rota.MessagesBase;
using BookStore.DataAccess.TransferObjects;
using System.Data;

namespace BookStore.Messages.Yayinevi
{
    [Serializable]
    public class ListYayineviRequest : RequestMessage
    {
        public YayineviTransferObject YayineviTransfer;
    }

    [Serializable]
    public class ListYayineviResponse : ResponseMessage
    {
        public List<YayineviTransferObject> YayineviList;
    }
}
```

Figure 5.12 Messages that are special for Book Store application. There two kinds of messages; "ListYayineviRequest" and "ListYayineviResponse". These messages store application specific data within them.

5.3.3 *Service Management Implementation*

To execute transaction in business logic and to achieve common tasks special controls called TxnManagers are used. (TxnManagers are explained in section 4.4 in this document.)

"TransactionModuleName", "TransactionGroupName" and "TransactionName" properties of TxnManager are used to specify the required transaction to execute.

While executing a transaction, doing automatic mapping operation is decided with "AutomapRequestMessage" and "AutomapResponseMessage" properties. If these properties are set to true value and controls of the interface has appropriate properties set the appropriate values, automatic mapping is done at the framework. If these properties are set to false, developer of application sets the values of controls to the request message or response message to controls manually. Figure 5.14 shows the properties of "TxnManager" and figure 5.14 shows how to manually map data from controls to request message.

```
<rota:TxnManager ID="AddYayineviTxnManager" runat="server" PipelineType="FetchPipeline"
    AutoDisplayNotifications="false" AutoMapRequestMessage="false" AutoMapResponseMessage="false"
    NotifyUserForUnauthorizedTransaction="false" TransactionModuleName="BookStore"
    OnMappingRequestMessage="AddYayineviTxnManager_MappingRequestMessage"
    TransactionGroupName="Yayinevi" TransactionName="AddYayinevi">
</rota:TxnManager>
```

Figure 5.13 Structure of "TxnManager" that is ready to map controls to messages and messages to controls manually. AutomapRequestMessage and AutomapResponseMessage properties are set to false.

```
protected void AddYayineviTxnManager_MappingRequestMessage(object sender, Rota.MessagesBase.RequestMessage requestMessage)
{
    YayineviTransferObject yayineviTransfer = new YayineviTransferObject();
    yayineviTransfer.Ad = YayineviAdTextBox.Text;

    ((AddYayineviRequest)requestMessage).YayineviTransfer = yayineviTransfer;
}
```

Figure 5.14 Manually mapping from controls to request message is shown in the figure.

To display notifications automatically that is produced while executing appropriate transaction "AutoDisplayNotifications" property is set to true.

After setting appropriate parameters to “TxnManager” control, it is executed with “Execute” method that triggers transaction dispatcher of Rota Framework that is mentioned in section 4.5. Figure 5.15 shows the execution of TxnManager.

```
void YayineviAddButtonEkleGuncelleSil_AddButtonClicked(object sender, EventArgs e)
{
    AddYayineviResponse addyayineviResponse = (AddYayineviResponse)AddYayineviTxnManager.Execute();
    if (addyayineviResponse.Status == Rota.MessagesBase.ResponseStatus.Successful)
    {
        ClearControls(this);
        ListYayinevi();
    }
}
```

Figure 5.15 This figure shows the executing of “TxnManager” control. While the user presses the appropriate button, “Execute” method of TxnManager is executed.

CHAPTER SIX

CONCLUSION

Rota Framework is a framework that is suitable to develop multi-layer applications with re-usability manner. Rota Framework has frozen spots to accomplish its task. Specialized component “TxnManager” that is responsible for being a bridge between presentation layer and business layer, a service manager called “TransactionDispatcher” and other specialized components that are generally used in presentation layers of applications are frozen points of Rota Framework. This frozen point can be changed by the designers and developers of framework but can not be changed and implemented by developers of applications. With this manner, developers of software applications knows that there is only one way to accomplish their jobs. Once a developer learns to develop an application with Rota Framework, he/she applies it to applications that are in different domains. Rota framework also has hot spots that can be developed according to different requirements of applications. Pipelines of Rota framework are hot spots and can be developed by developers of applications according to the needs of an application.

Rota framework framework is useful for developers to write less code and have much more time to have and analyze the requirements of applications. By using rota framework, maintenance and development cost of applications can be reduced too.

Rota Framework is a black box framework. A developer does not need to know the internal structure of framework so he/she develops large domain of applications with rota framework. Rota framework has configurable parts like “TxnManager”, so application developers learn only the properties of configurable parts’ properties.

Rota Framework is used to develop n-tier applications. In n-tier application architectures each layer’s content is completely hidden to other layers and tiers are loosely coupled from each other. So, it is easy to change any layer without affecting any other layers. Through this benefit, applications become more flexible and more reusable.

Rota Framework is used in presentation and business layer of applications. It has presentation layer components that are responsible from accomplishing user interface tasks and business layer components that are used to execute appropriate services and performing specialized operations like logging, autorizing and etc. In the future, components that are used in data access layer of application can be developed in Rota Framework.

REFERENCES

Anonymous (n.d.). *Software Frameworks*, Retrieved December 18, 2007, from http://www.chl.chalmers.se/~j/complex_systems/articles/31.pdf.

Chartier R. (n.d.). *Application architecture: An N-Tier Approach,* Retrieved April 5, 2007, from http://www.15seconds.com/issue/011023.htm.

Çok katmanlı mimari (N-Tier architecture), (n.d.), Retrieved January 2, 2007, from http://www.15seconds.com/issue/011023.htm.

Earles J. (n.d.) *Framework Evolution! One Box, Two Box, White Box, Black Box,* Retrieved January 1, 2008, from http://www.cbd-hq.com/articles/2000/000401je_frameworks2.asp

Fayad M., Johnson R., Schmidt D. C. (1999). *Implementing Application Frameworks: Object-Oriented Frameworks at Work* (1thEd.). NY: Wiley & Sons.

Hohpe, G. & Woolf, B. (2004*). Enterprise integration patterns* (5th Ed.). NY: Addison Wesley.

Hohpe, G. (2002). *Enterprise integration patterns*. (8-11), http://www.enterpriseintegrationpatterns.com/docs/Enterprise%20Integration%20Patterns%20-%20PLoP%20Final%20Draft%203.pdf.

Jezierski, E.A. (2002), *Application architecture for .Net: Designing applications and services* (1thEd.). NY: Microsoft

Maekiewicz, M.E. & ucena C.J.P. (2000). *Understanding Object-Oriented Framework Engineering.* (1-2), http://www.acm.org/crossroads/xrds7-4/frameworks.html.

Software framework, (n.d.), Retrieved May 15, 2007, from http://en.wikipedia.org/wiki/Software_framework#Types_of_software_frameworks

Trowbridge, D., Roxburgh, U., .Hohpe, G., Manolescu, D., Nadhan, E.G. *(2002). Integration patterns* (1th Ed.). NY: Microsoft

Ronalt M. (1997). *White-box frameworks vs. Black-Box frameworks*., Retrieved December 15, 2007, from http://swt.cs.tu-berlin.de/~ron/diplom/node35.html.

Wake, W.C. (n.d). *White box and black box frameworks*. Retrieved February 12, 2007, from xp123.com/wwake/fw/ch12-bb.htm.

What is N-tier Architecture, (n.d.), Retrieved January 2, 2007, from http://www.developerfusion.co.uk/show/3058/2/

Woolf, B. & Brown K. (2002). *Patterns of System Integration with Enterprise Messaging.* PLoP 2002 conference, (15-47), members.aol.com/messagingpattern/woolf-brown2.pdf