A Distributed Architecture for U-Messaging Systems

Dissertation

Submitted in Partial Fulfillment
Of the Requirements for the Degree of
Doctor of Professional Studies in Computing

At

Pace University

School of Computer Science & Information Systems

By
Hevel Jean-Baptiste

Advisors
Professor Fred Grossman
Professor Paul Dantzig
Professor Chuck Tappert

September 2002

ABSTRACT

This research describes a *Distributed Architecture for U-Messaging Systems* (DAUMS). With the explosion of the Internet and the increasing complexity of business today, Information Technology (IT) developers and project managers have to work harder and harder to find ways to develop a Universally Unified Messaging System efficiently. Obviously it is a challenge to create a well-defined architecture for such a system. DAUMS is an architecture that is created because of the need for a Universally Unified Method Invocation (UUMI). As the need for a universal portal has grown considerably, it has also created the need for connecting all the communication media in a seamless fashion. Users want to listen to their phone and email messages in one phone call. They want to read their emails and their faxes on the same Web browser. In summary, they want their email, faxes, phone messages, video conferences, and real time messages connected together. Not only will DAUMS provide a solution to these problems, it will create a standard that can be shared by all developers.

DAUMS architecture will support clients such as: browser, User Interface (UI), telephone, wireless and so on. It will be a hybrid between Java 2 Enterprise Edition (J2EE) and Windows DNA, and it will handle synchronous and asynchronous message mechanisms. As we create this architecture, we will look at all the pros and cons of other emerging distributed architectures for Web Services. Also, a Universal Messaging Application Framework will be created using this architecture to show how the DAUMS Architecture can be used to develop an application system.

This work can be used as a tool for IT managers, developers, project leaders, programmers and researchers. In addition, the framework can provide a starting point for developers and project managers; they do not have to start from scratch when building a Universally Unified Messaging System, which we will call U-Messaging system. The framework will work jointly with Universal Description Discovery and Integration (UDDI) to provide unlimited Web Services access to clients.

Acknowledgements

I thank Prof. Fred Grossman for his guidance, support and patience. This work would not have been possible without his advice and support. I am grateful to the members of my thesis committee Profs. Paul Dantzing and Chuck Tappert for their time and advice. I want to thank each one of my advisors for their help. I want to thank Prof. Paul Dantzing who always cut his lunch short to meet with me. He always directed me to the right document. Also, he always gave me a list of documents that are in the line of my research and asked me if I have read these documents already. I want thank Prof. Tappert who made me fall in love with emerging technology. Again, I want to thank Prof. Grossman who always asked me what is my contribution, or what is new that my research is going to bring to the Information Technology. These types of questions have helped me think deeper and deeper. It was a long journey for me. However, today would never have been possible without the help of all the faculty members of this University. I want to thank Prof. Maude Meisel who helped me with my grammar.

To Habtamua Jean-Baptiste, my wife, for her love and faith in me, words are inadequate to express my gratitude.

Table of Contents

List of Figures…… vii
List of Tables…… vii

Chapter 1 Distributed U-Messaging
1.0 Introduction……..1
1.1 What is U-Messaging…….. 1
1.2 What is Unified Messaging?…… 2
1.3 Relationship Representation of Unified Messaging…… 5
1.4 What Is Universal Messaging?…… 6
1.5 Relationship Representation of Universal Messaging…..…… 7
1.6 Universally Unified Messaging (U-Messaging)…...……..……. 8
1.7 Well Known Distributed Architectures – J2EE and Windows DNA…….…. 9
1.7.1 Java2 Enterprise Edition (J2EE0 …… 10
1.7.2 Windows DNA Architecture …….. 11
1.7.3 What is DAUMS and What will it Add to the Unified Messaging….. 12
1.7.4 The Purpose of the DAUMS Architecture…… 14
1.7.5 Development Effort With J2EE or Windows DNA……. 14
1.8 JMS and UUMI Examples …… 14
1.8.1 JMS Deficiencies …… 15
1.8.2 UUMI Solutions For JMS Deficiencies ……... 16
1.8.3 Using JMS To send a Synchronous Message ……...16
1.8.4 Using UUMI To Send a Synchronous Message ……..… 18
1.8.5 Using JMS To send an Asynchronous Message …… 20
1.8.6 Using UUMI To Send an Asynchronous Message…… 22

1.9 Messaging API of J2EE and Windows DNA and DAUMS…… 24
1.9.1 Missing Functionality in JMS…….. 25
1.9.2 Missing Functionality in MSMQ…… 25
1.10 The DAUMS Architecture For U-Messaging…… 25

Chapter 2 Distributed Architecture For U-Messaging Systems (DAUMS)
2.0 General View of the DAUMS Architecture …… 27
2.1 The Tiers of the DAUMS Architecture ……..…… 28
2.2 Description of Each Section of the DAUMS Architecture.……. 28
2.2.1 The Client …… 29
2.2.2 The Application Logic.……..30
2.2.3 The Data …… 39
2.3 Application Logic of the DAUMS Architecture.…….. 41
2.4 Role of Each of the Components in the DAUMS Application Logic Layer…. 41
2.4.1 Web Server/ Video Server…… 42

2.4.2 Speech Server ……... 42
2.4.3 WAP Server.…… 42
2.4.4 Universal Translator.……. 42
2.4.5 Business Logic.……. 42

2.5 Navigation Through the DAUMS Architecture…….. 43
2.6 Web Services: The Next Horizon for e-business……. 44
2.7 Definition of Web Services……. 46
2.8 Advantages of DAUMS Architecture……. 46
2.8.1 Universally Unified Method Invocation Process and Web Services… 47
2.8.2 Scalable Component Flexibility……... 47
2.8.3 Configuration at Run Time Process…….. 47
2.8.4 Synchronous Messaging Mechanism.…… 47
2.8.5 Asynchronous Messaging Mechanism.…… 47
2.8.6 Peer-to-peer Communication Messaging……. 47
2.8.7 Instant Notification……... 47
2.8.8 Solid Framework……... 48
2.8.9 Real Time Notification……. 48

Chapter 3 DAUMS and Universally Unified Method Invocation (UUMI)
3.0 Interaction Of UUMI as a Web Service With other Components…….. 50
3.1.1 The Network …… 51
3.1.2 XML Messaging-Based Distributed Computing.……..52
3.2 Unifying Messaging Communication for E-Business Success…… 53
3.3 Services for Real e-business with UUMI …… 54
3.4 Security with UUMI ……... 55
3.5 The Meaning of Distributed Applications …… 57
3.6 Distributed Applications Architecture and Design……. 59
3.6.1 Distributed-subsystem Structure. ……. 60
3.6.2 Message Communication ……... 61
3.6.3 Distributed Configuration. …….. 63

Chapter 4 DAUMS Integration With Web Services
4.0 What is the essential of this research?……. 65
4.1 Topic Naming ……. 65
4.2 Hierarchical Topic Support ……66
4.3 Wire Protocol …….. 67
4.4 Broker Clustering …… 68
4.5 Administration Tools ……. 68
4.6 Multicasting Messaging …….. 68
4.7 Broadcasting Messaging ……..69
4.8 How UUMI will be integrated with (JMS, SOAP, WSDL, and WSFL) ……. 69
4.8.1 UUMI Log in Screen …….. 69
4.8.2 UUMI Main Screen ……. 70
4.8.3 UUMI Setting Screen ……. 73

4.9 How is UUMI, WSDL, SOAP, WSFL, and UUMI work?……. 75
4.10 A Summary of What This Study will bring to the IT World …… 85
4.10.1 Subscribe/Publish Mechanism …….. 86
4.10.2 Subscribe Mechanism with Retaining Messages …….. 86
4.10.3 Topic Naming …… 86
4.10.4 Administration Tools …… 86
4.11 Strengths and Weaknesses of Some Messages Systems……... 87
4.11.1 MQSeries Strenghts. …….. 87
4.11.2 MQSeries Weaknesses. …….. 87
4.11.5 UUMI Strengths……. 87
4.11.6 UUMI Weaknesses……. 87
4.11.7 MSMQ Strengths…… 87
4.11.8 MSMQ Weaknesses……87

Chapter 5 DAUMS Application Interface Programming (API)
5.0 UUMI Application Configuration…… 92
5.1 Error/Message Logging…….. 96
5.2 Common GUI Support…… 98
5.3 UUMI Client and UUMI Server Via UUMI Session Manager…….. 99
5.4 Common Layers on UUMI Client and UUMI Server…… 106
5.5 Fix Protocol Over HTTP for Client/Server Interface……. 108
5.6 Generic UUMIMessage Object…….. 113
5.7 UUMI Client…….. 117
5.8 UUMI Server…….…. 127
5.9 Heartbeat…… 135

Chapter 6 Conclusion Of The DAUMS Architecture Research

Conclusion…… 137

Appendix A. ……. 140
Appendix B…….. 141
Appendix C…… 147
Appendix D…… 159

Glossary…… 165

References…… …….. 177

Figures and Tables

Figure 1: A Diagram For An Example of Unified Messaging
Figure 2: Unified Messaging Relationship Representation
Figure 3: An Example of Universal Messaging
Figure 4: Universal Messaging Relationship Representation
Figure 5: The U-Messaging Relationship
Figure 6: The Java 2 Enterprise Edition (J2EE) Architecture
Figure 7: The Windows DNA Architecture
Figure 8: Distributed Architecture For U-Messaging Systems (DAUMS)
Figure 9a: The General Architecture of the DAUMS
Figure 9b: The General Architecture of the DAUMS With Detail On UUMI
Figure 10: Example of a User Who wants to receive Emails By Telephone
Figure 11: Example Of a user Who Want to receive Emails On Fax Machine
Figure 12: Integration of The Universal Translator With The Business Logic
Figure 13: The Application Tier of the DAUMS
Figure 14: Navigation Through The DAUMS Architecture
Figure 15: Universally Unified Method Invocation (UUMI) Diagram
Figure 16: Example of a Hierarchical Events
Figure 17: Info Services System Log in Screen
Figure 18: UUMI Main Screen
Figure 19: UUMI User Profile
Figure 20: Integration Of UUMI With JMS, MQSeries and UDDI
Figure 21: Integration of UUMI With MQSeries and UDDI
Figure 22: Integration of UUMI With UDDI
Figure 23: Integration Of UUMI as a standalone Messaging System
Figure 24: UUMIClient and UUMIServer Data Flow and Connectivity Overview
Figure 25: UMIClientPeer Interaction
Figure 26: UUMISessionServlet Connectivity

Table 1: Interaction of UUMI With Other Components

Chapter 1

U-Messaging

1.0 Introduction

Today's communications environment is still characterized by the usage of several services. Each of them needs to be accessed in a specific way. For example, users have to use two different devices if they want to make a phone-call and to send a fax. The more services a user wants to employ, the more devices or applications he has to operate [28]. A lot of effort is been spent on trying to make devices communicate. Software companies and hardware manufactures are working very hard to provide a solution for Unified Messaging. Web Services provides a Universal Discovery, Description and Integration Protocol (UDDI) as a means for facilitating the development of Unified Messaging Systems. These solutions aim to fulfill the vision of integrating different communication protocols. This is why we have proposed a *Distributed Architecture for U-Messaging Systems* (DAUMS).

1.1 What is U-Messaging?

U-Messaging is a combination of Unified Messaging and Universal Messaging. It includes the three relationships -- one to many, many to one and many to many -- which we have found respectively in Unified Messaging, Universal Messaging, and Universally Unified Messaging, which we call U-Messaging (UM).

1.2 What is Unified Messaging?

Unified Messaging is the ability to access all types of messages using the device or terminal of your choice, e.g., telephone or computer. Imagine being a systems administrator in an international firm, and you have to oversee the global technology operation of the company. In this case you have to deal with people, network systems, applications, and processes that can be in different countries, time zones, platforms, and languages. That type of responsibility can place you in a position where you can receive hundreds of emails, phone calls, pages, and faxes a day. Businesses have tried to wrestle with this issue for years. Some of them have tried to develop their own proprietary Application Programming Interfaces (API) to get content from different systems and combine them into a single Unified Messaging System.

Sometimes these proprietary wrappers do not work very well with other external systems. The communication protocol can become a bottleneck for developers. According to Kundan Singh and Henning Schulzrinne, " *A unified Messaging System must be compatible with existing Internet multimedia protocols, and should be designed with little or no modification to the current infrastructure"*. [13]

In thinking about Unified Messaging, one has to consider that people need to communicate using many modes. Therefore the messaging devices should be built in a unified way. They should be able to support keyboard and non-keyboard access. Users should be able to use their voices to communicate through email and fax naturally. Voice and data should be able to be transferred to the Internet and get to their destination with

no loss of content or meaning. There do exist Unified Messaging Systems that will transmit voicemail messages over the Internet. [25] However, Unified Messaging should be more than just receiving email and voice mail in a single device. It should make email look like voice mail and voice mail look like email. Unified Messaging is not a single product, or even a group of products, rather it is a set of capabilities that can be delivered in addition to basic e-mail. Unified Messaging can be defined as offering access to any message, at anytime, anywhere via any device. [34]

For example, suppose that User A sends an email to User 1 and User B sends a voice mail to User 1. Without a Unified Messaging System, User 1 can only receive email using device X, e.g., a PC, and voice mail using device Y, e.g., a telephone. However, with Unified Messaging, User 1 could receive email and voice mail using either device X or device Y, i.e., both on the same device. It is important to mention that Unified Messaging is defined in the context of the receiver. Therefore a user can receive messages from more that one source and be able to retrieve them in one device or multiple devices. Figure 1 is a graphical representation of utilization of unified messaging.

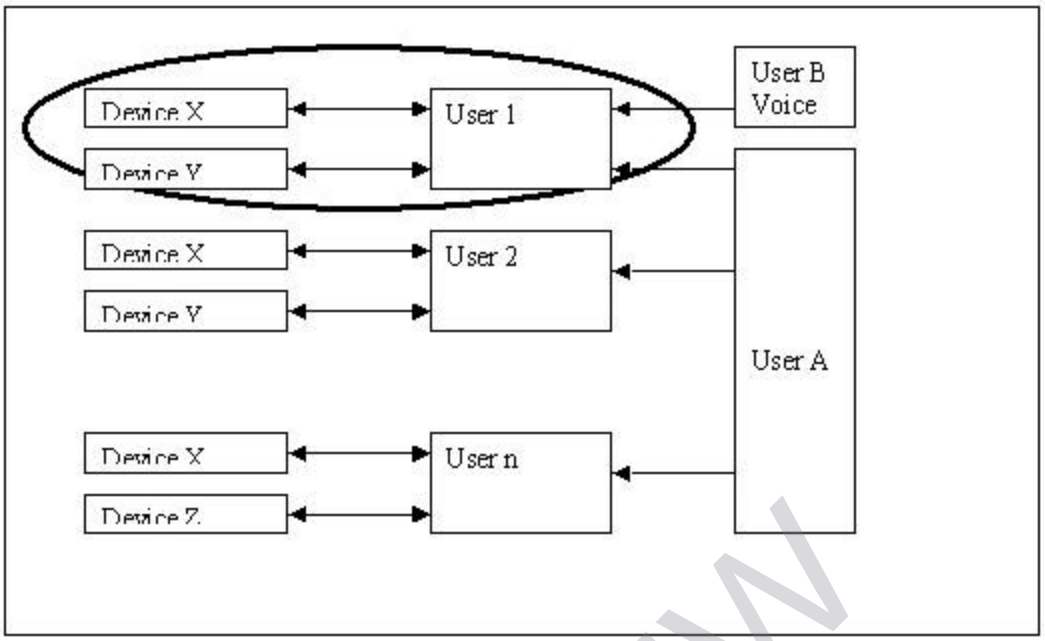

Figure 1: A Diagram For An Example of Unified Messaging

The message context conveys information about the way that the user expects to interact with the message. For example, a message may be e-mail, voice mail, fax, etc. A smart user agent may have specialized behavior based on the context of the message. [3]

The concept of Unified Messaging is very simple to explain, but the problem is to understand where and how all the pieces fit together to provide the Unified Messaging services. Traditional telephony services, such as call forwarding, transfer, and 800 number services can be enhanced by interaction with email, Web, and directory services. Additional media types, like video and interactive chat, can be added as well. One of the challenges in providing these services is how to effectively develop the necessary software. Programming these services requires decisions regarding where the code executes, how it interfaces with the protocols that deliver the services, and what level of

control the code has. [11] This is important when it comes to getting all the pieces to work together provide a unified access.

1.3 Relationship Representation of Unified Messaging

The relationship in Unified Messaging is many to one, where from one device a user can access all his messages. This relationship is represented in Figure 2.

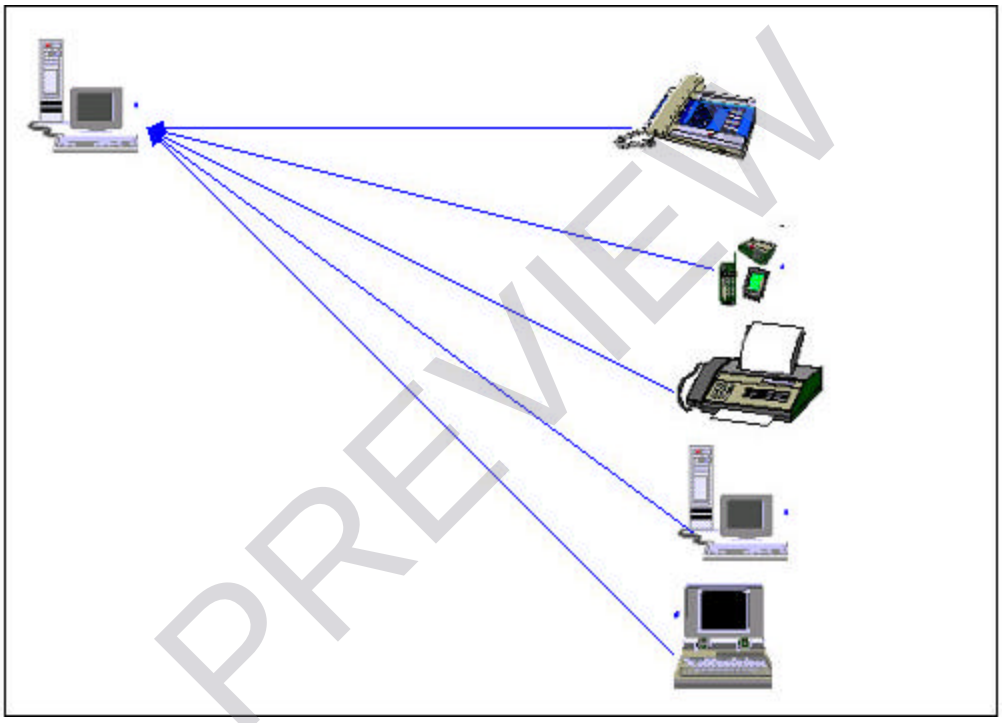

Figure 2: Unified Messaging Relationship Representation

It is important to mention that in the Figure 2 the user uses a computer to access his messages. However, the user could use a phone, a PDA and so on to access his messages as well. A user has the freedom to use whatever device is available in a true Unified Messaging system.

1.4 What Is Universal Messaging?

Universal Messaging is the ability to distribute a single message to different recipients no matter the type of access device and network the originator or recipient might choose to use [34]. Assume that a manager wants to notify a group of fifty salespeople about a sale that has just closed. The manager can send one email and all the salespeople that are in the field anywhere can receive it in their own device. These devices can be wireless phone, Palm pilot, pager, email and so on. That is the power of Universal messaging. This will allow users to receive messages in their own devices without thinking about the sending device.

For example, suppose User A wants to send a message to a list of users and all these users should get the message in their own device. As shown in Figure 3, User A sent one message and all the users will receive that message in their own device. The source can be defined as Universal Messaging because it reaches more than one destination from one message sender. It is important to mention that the source does not have to worry about the type of receiving device.

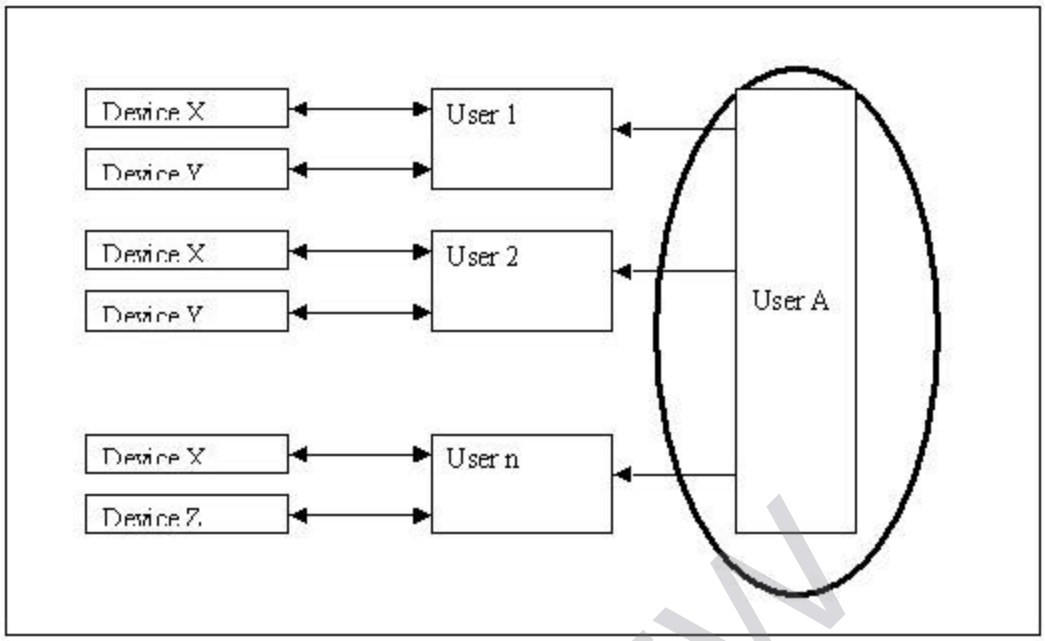

Figure 3: An Example of Universal Messaging

1.5 Relationship Representation of Universal Messaging

In Contrast to Unified Messaging, Universal Messaging represents a one-to-many relationship. From one device a user will be able to reach a variety of others, where the device at the destination can be independent of the originator device. Figure 4 is an illustration of Universal Messaging.

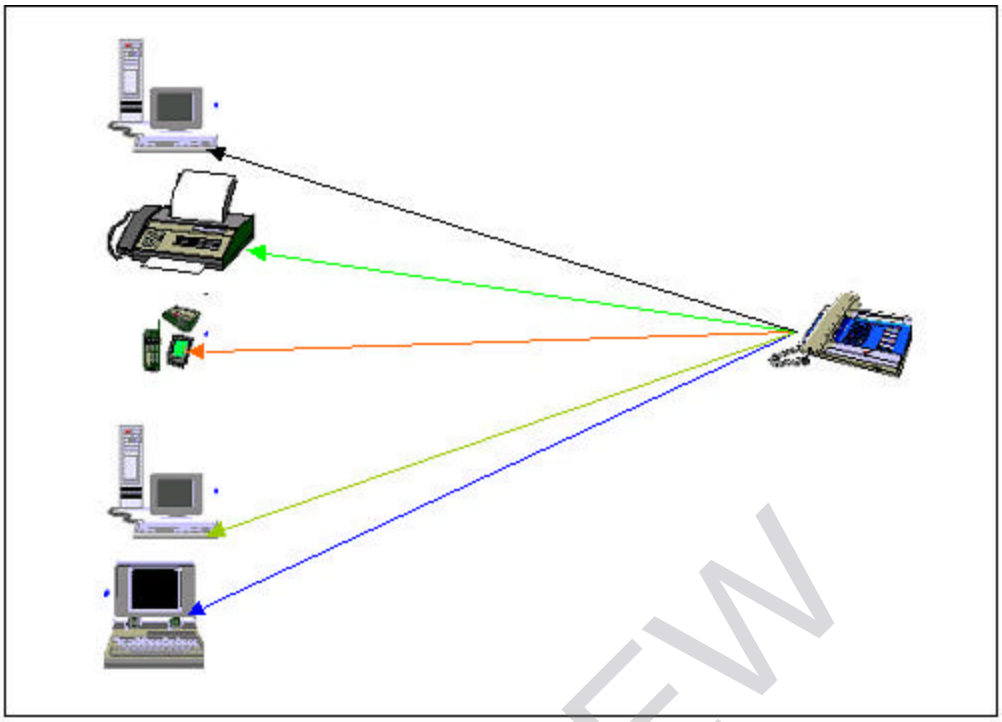

Figure 4: Universal Messaging Relationship Representation

In this case the user who sends the message uses a phone to send it, and everybody that is supposed to receive the message gets it on his own device. In a true Universal Messaging System a user could send a message from any type of device, and the message will be delivered to its destinations.

1.6 Universally Unified Messaging (U-Messaging)

The relationship between these two mechanisms creates a third relationship that is many to many. Therefore, from more than one device a user can reach other users that are using the same or a different device. This is a representation of a relationship between Unified Messaging and Universal Messaging that we call Universally Unified or U-Messaging. This is illustrated in Figure 5

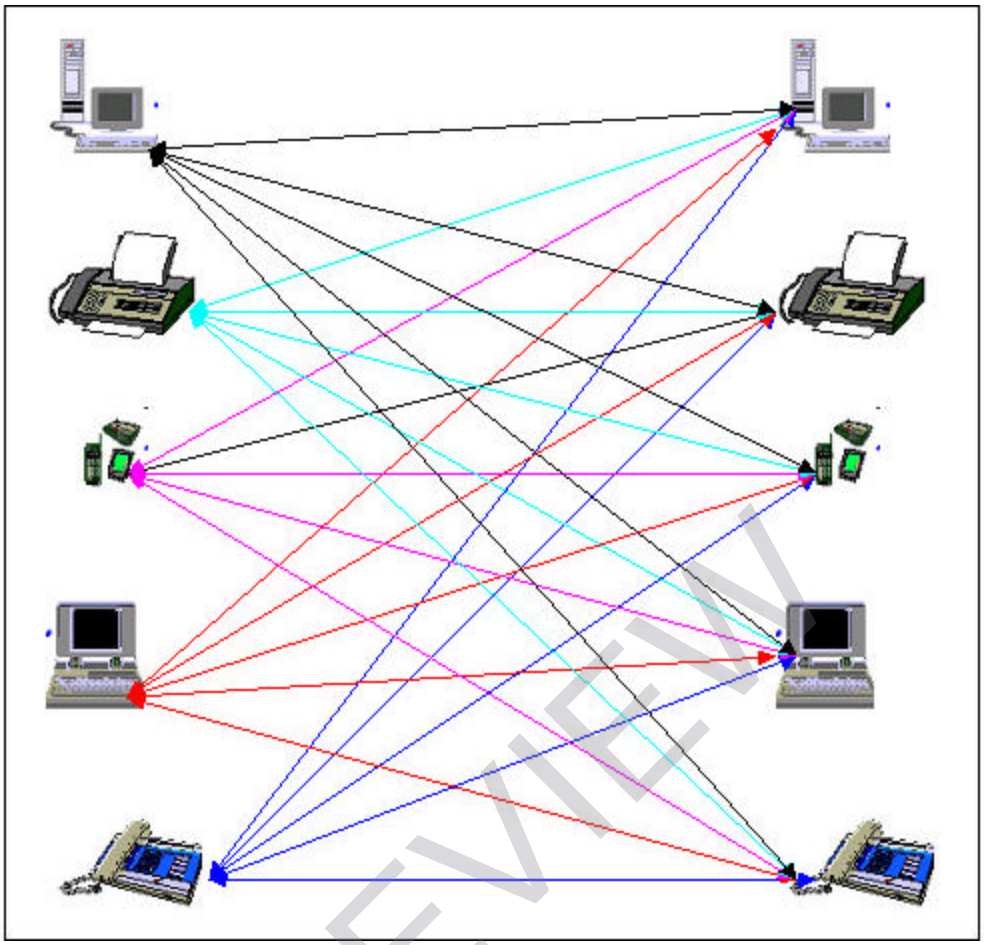

Figure 5 : The U-Messaging Relationship

From any device a user can send a message to reach many recipients regardless of what type of devices are used to send or receive the message.

1.7 Well Known Distributed Architectures – J2EE and Windows DNA

The purpose of these two distributed architectures is to provide a means for developing multi-tiered applications. Therefore different parts of a system can be running in different machines or servers independent from each other. That flexibility helps developers change one part a system without touching another part.

1.7.1 Java 2 Enterprise Edition (J2EE) [42]

This is a Sun MicroSystems Architecture. It is an open architecture that provides integration flexibility to developers of third party software. It is very important to know the flexibility that an architecture can provide. Because businesses do not want to be tied to a particular architecture or application, they are always looking for a flexible application that can be integrated with others easily.

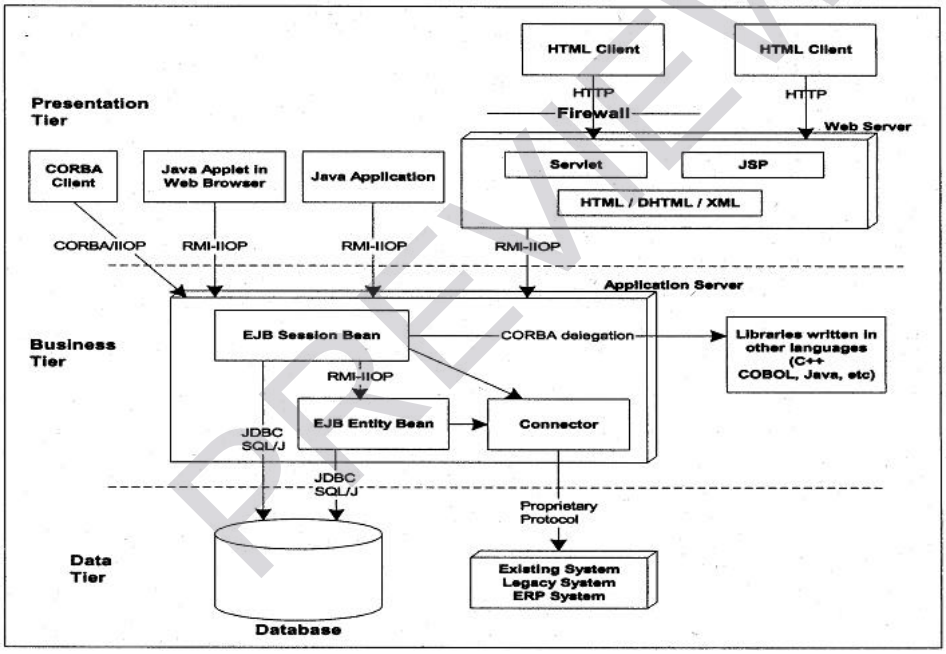

Figure 6: The Java 2 Enterprise Edition (J2EE) Architecture

This architecture is already well defined in the J2EE specification [42]. As you can see it has three layers: the Presentation Tier, the Business Tier and the Data Tier. Each of these layers works together through some protocols. For example, between the Presentation Tier and the Business Tier the following protocols have been used: HTTP, RMI-IIOP and

CORBA. Also the Business Tier and the Data Tier work with the JDBC and other proprietary protocols.

Each layer is separate and can run in a different location and on a different platform. However, using the appropriate protocol they can communicate with each other as if they were running in the same place or the same environment.

1.7.2 Windows DNA Architecture [38]

This is a Microsoft architecture. It comes with components that can aid developers in the development of a Unified Messaging Application. However, some of the components are proprietary to Microsoft. Therefore, if you use this architecture, you have to use Microsoft components.

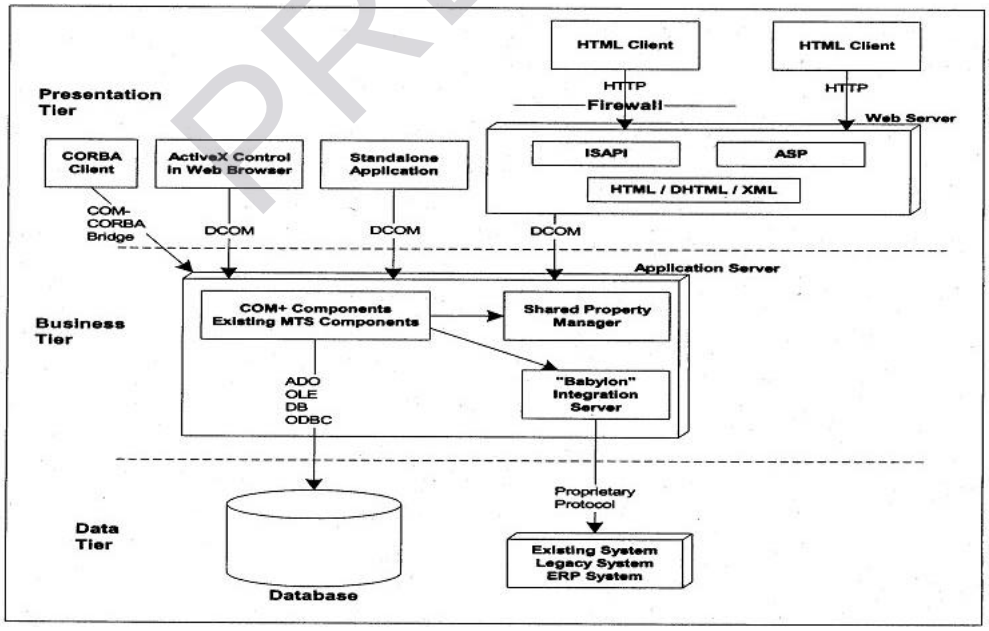

Figure 7: The Windows DNA Architecture

This architecture is defined in the Windows DNA architecture specification [38]. Like J2EE architecture, it has the Presentation Tier, the Business Tier and the Data Tier. Also, each of those tiers communicates with the others through some protocols. However, the windows DNA protocols are different from the J2EE protocols. Between the Presentation Tier and the Business Tier, Windows DNA Architecture uses the following protocols: HTTP, COM-CORBA Bridge, and DCOM. Between the Business Tier and the Data Tier, the following drivers and protocols are used: ADO, OLE, ODBC.

In the Presentation Tier, a client can be an HHTP browser behind a firewall or an HHTP client without a firewall. Also you can have the following clients: Standalone Application, ActiveX Control in Web browser, CORBA Client. However, it does not matter which client requests information, the Business Tier responds to the client needs. The same thing is true for the relationship between the Business tier and the Data Tier. When a request is made to the Data Tier, this layer responds to the Business Tier with no idea about the type of client that the data are going to. In short, the Business Tier is a client for the Data Tier just as the Presentation Tier is a client for the Business Tier.

1.7.3 **Introduction to the Distributed Architecture for U-Messaging Systems (DAUMS)**

Briefly, DAUMS can be defined as an architecture that provides a unique framework for Universal Messaging and Unified Messaging. Because so many problems exist in the development of distributed applications today, it has become even more complicated to

face the challenge of developing a distributed application for universally-unified messaging (U-Messaging).

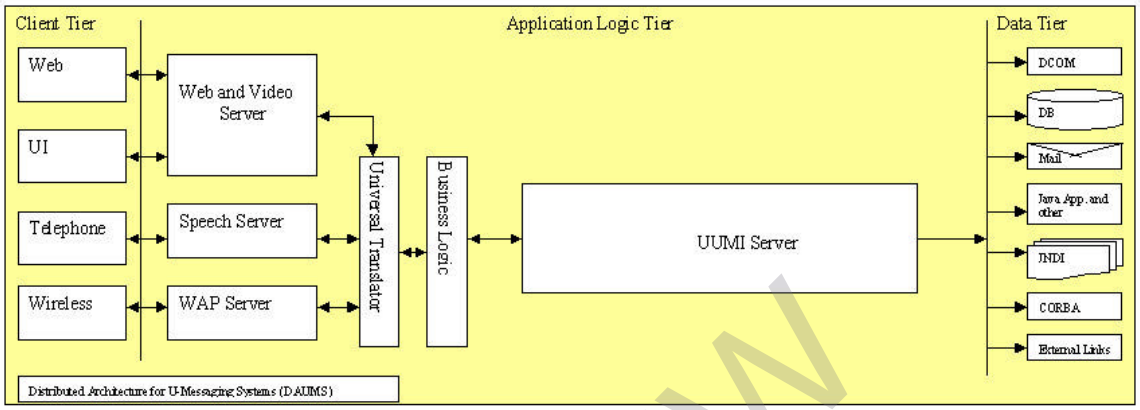

Figure 8: Distributed Architecture For U-Messaging Systems (DAUMS)

DAUMS provides a framework that a developer can use to support and facilitate the development of a messaging system. This architecture will give a significant advantage to a developer. Developers can concentrate on deploying their applications and developing business rules. It is important to mention that a lot of different tools and protocols can be used in this architecture, such as HTTP, Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), Web Services Flow Languages (WSFL), and Universal Description Discovery and Integration (UDDI). [7]

DAUMS will work with Java 2 Enterprise Edition (J2EE), Windows DNA, and IBM Software Group Web Services, specifically Universal Description, Discovery and Integration (UDDI) [1]. DAUMS will be the infrastructure for the next generation of Universally Unified Messaging Systems.

1.7.4 The Purpose of the DAUMS Architecture

The purpose of the DAUMS is to provide an architecture for a Universally Unified Messaging System. Therefore developers can customize the DAUMS API to fit their business need or use the specification of this architecture to develop good messaging systems.

1.7.5 Development Effort of A Messaging Application with J2EE or Windows DNA

These architectures are not too difficult to understand, as illustrated in Figures 6 and 7. However, using this architecture to develop a messaging system often requires a large effort by skilled developers. Each one of these architectures comes with a messaging API. However there are a number of issues that are not addressed in these APIs that make it very difficult to develop a good Unified Messaging System. That is the reason why DAUMS was developed.

1.8 JMS and UUMI Examples

JMS provides a common way for Java programs to create, send, receive and read enterprise messaging system's messages. UUMI provides a unified way for application

programs to create, send, receive and read an enterprise messaging system's messages. However, UUMI provides some functionalities that are not implemented in JMS.

1.8.1 JMS Deficiencies

JMS does not address the following functionality:

- Load Balancing/Fault Tolerance - Many products provide support for multiple, cooperating clients implementing a critical service. The JMS API does not specify how such clients cooperate to appear to be a single, unified service.
- Error/Advisory Notification - Most messaging products define system messages that provide asynchronous notification of problems or system events to clients. JMS does not attempt to standardize these messages. By following the guidelines defined by JMS, clients can avoid using these messages and thus prevent the portability problems their use introduces.
- Administration - JMS does not define an API for administering messaging products.
- Security - JMS does not specify an API for controlling the privacy and integrity of messages. It also does not specify how digital signatures or keys are distributed to clients. Security is considered to be a JMS provider-specific feature that is configured by an administrator rather than controlled via the JMS API by clients.
- Wire Protocol - JMS does not define a wire protocol for messaging.
- Message Type Repository - JMS does not define a repository for storing message type definitions and it does not define a language for creating message type definitions.