Fall 2005

Emanager - Cdo Made Simple

Solomon Vedaprakash
Regis University

Regis University
School for Professional Studies Graduate Programs
Final Project/Thesis

Disclaimer

eManager – CDO Made Simple

A Thesis submitted

to the Graduate School

Regis University

in partial fulfillment of requirements for the degree of

MASTERS OF SCIENCE

in the Department of Masters of Science in Computer Information Technology

October 2005

Solomon Vedaprakash

Acknowledgements

It is with great gratification and joy I find myself writing this section. There were times during the project I didn't think that I would make this project a success, but here I am. The development of the eManager application provided many benefits, including the opportunity to cultivate the knowledge in programming with CDO, ADO and other Windows artifacts such as events, processes and threads using VC++.

"We create a world for ourselves by what we speak. Words have power, and we can speak life into a situation". I take pleasure to show my gratitude to Prof. Dan Likarish, Assistant Professor, Regis University for his sincere, dedication and commitment he showed in encouraging me throughout this project. There were times I almost gave up on certain areas of the project, but Dan helped me with wonderful suggestions cheerfully even if it was after office hours. It was amazing how professionally he organized the online conference calls and engaged everyone involved, though some of us were connected from thousands of miles away. His encouragements and guidance were not only limited to technical and professional project but I had the privilege of enjoying his great advice on pursuing my PhD program. Dan, I will never forget those conversations and your words of caution on my decisions; you will always be my loving professor, so dear to me.

I would like to convey my sincere and genuine thanks to Mr. Mohan Reddy, President, RedySoft Inc (www.redysoft.net) for readily offering this project to be developed and

acknowledging the success of the eManager application development. His encouragements and willingness to impart technological advancements were definitely the motivating factor for the successful completion of this project. I personally thank him for providing the infrastructures such as software, hardware and support throughout. It was a pleasure working with you.

I would like to extend my special thanks to Mr. Rajan for providing the valuable technical wealth of experience and sharing it with me. I thank him for all his technical references and guidance throughout. I promise to bombard you with more technical questions!

Before I move on, my special thanks to my dad, mom, brothers, sister and their families for their persistent encouragement. Today, I can boldly admit, without you, I wouldn't have made this possible. I thank you for all your prayers, support, incredible patience, dedication, flexibility, support and commitment throughout. "A praying family is God's power" and your prayers released the power to do Gods will. Magnificent products such as eManager do not make perfect without your love, affection, care and guidance. I have so much to say and thank each and every one of you and not enough time and space left to say it all.

Even though this is the last part of this section, I pray that this is just the beginning of the blessings of the LORD towards me. May HIS Blessings and HIS Seeds of faith deeply planted in prayer grow in me into following HIS heart and will.

October 8, 2005

Table of Content

Acknowledgements **1**
Table of Content **4**
List of Figures **5**
Abstract **7**
Summary **8**
Introduction **10**
Project History **13**
Systems Analysis and Design **15**
Systems Analysis 15
Functional Requirement 16
Usability Requirement 17
Reliability Requirement 17
Supportability Requirement 18
Performance Requirement 18
Development Methodologies **19**
Functional Modeling 19
Entity Class Modeling 21
Dynamic Modeling 22
Overview of Technology **24**
Active Directory 25
LDAP 26
ADSI 27
CDO for Exchange 27
COM Component Design 28
Application Design **32**
TAB Control 33
Custom ComboBox Control 37
CMListCtrl class: 38
CMListCtrlWnd class: 38
CMButtonCtrl class: 39
Windows Messages 40
ON_MESSAGE: 41
User-Defined Messages/WM_APP constant Messages: 42
Registered Windows Messages: 42
Process and Threads 44
Inter Process Communications 46
Accessing Web Storage System using SQL 47
CDO for Exchange 50
Application Enhancements **55**
Obstacles, Pitfalls and Suggestions **56**
Conclusion **59**
Appendix A: Use Cases **61**

Appendix B: CRC Cards **92**
Appendix C: List of Figures **93**
References **113**
Glossary **117**

List of Figures

Figure 1: eManager UC1 [Use Case] 93
Figure 2: Read Items UC-RI00 [Use Case] 94
Figure 3: Read Items UC-RI01 [Use Case] 95
Figure 4: Read Items UC-RI02 [Use Case] 95
Figure 5: Read Items UC-RI03 [Use Case] 96
Figure 6: Read Items UC-RI04 [Use Case] 96
Figure 7: Read Items UC-RI05 [Use Case] 97
Figure 8: Read Items UC-RI06 [Use Case] 97
Figure 9: CrNew Items UC-NI00 [Use Case] 98
Figure 10: CrNew Items UC-NI01 [Use Case] 98
Figure 11: CrNew Items UC-NI02 [Use Case] 99
Figure 12: CrNew Items UC-NI03 [Use Case] 99
Figure 13: CrNew Items UC-NI04 [Use Case] 99
Figure 14: MtClient Process UC-MT00 [Use Case] 100
Figure 15: eManager Context Diagram [Use Case] 100
Figure 16: Application accessing Directory Service using LDAP wire protocol [CMN00] 101
Figure 17: The relationship of ADSI and ADO to Active Directory [CMN00] 101
Figure 18: CDO to access Active Directory [CMN00] 102
Figure 19: Proxy / Stub Overview [JSW00] 102
Figure 20: Connection Points and Sinks [JSW00] 103
Figure 21: eManager Application Architecture 103
Figure 22: eManager Contacts Tab Control Dialog 104
Figure 23: Custom ComboBox Control Class Diagram 105
Figure 24: eManager Contacts window with custom ComboBox shows the Business 2 item selected 106
Figure 25: eManager application attachments processing using XML document and Stream object 106
Figure 26: eManager Data Communications through XML 107
Figure 27: eManager application class diagram 107
Figure 28: eManager application Logon/Server Event 108
Figure 29: eManager application Send Mail 108
Figure 30: eManager application Read Items 109
Figure 31: eManager application Save Draft 110
Figure 32: eManager application Move Item 110

Figure 33: eManager application Delete Item..... 111
Figure 34: eManager application Create New Message State Diagram 111
Figure 35: eManager application Component Diagram..... 112
Figure 36: eManager application Deployment Diagram..... 112

Abstract

"eManager – CDO Made Simple" by Solomon Vedaprakash

The eManager application integrates the existing enforcement systems and MS Exchange Server 2003. The application exploits the benefits of CDO, ADO, LDAP, ADSI, XML and COM technologies. The application explicates the integration and utilization of CDO using VC++ programming language. It illustrates how CDO can be implemented effectively in various circumstances while maintaining uncomplicated application design and solutions. It demonstrates how CDO and ADO complements each other and interacts with other Windows artifacts such as ADSI, LDAP and Active Directory. The application implements the COM methodology that establishes the COM events using connection points and sinks. The integration and management of process and threads using events is the prim benefit for application programmers.

Keywords:

CDO	ADO	LDAP	ADSI
COM	Events	Sink	Process
Thread	Active Directory	Connection Point	

Summary

The current system supporting Redysoft Technologies necessitates modernization in providing a successful enforcement programs with instant messages and critical data updates needed by the Law Enforcement agencies. The eManager application integrates the existing enforcement applications and MS Exchange Server 2003. The application is an advanced e-mail software program that allows users to send and receive e-mail. In addition to e-mail, eManager has a personal calendar, group scheduling, personal contacts, personal tasks, the ability to collaborate and schedule with other users. The eManager authorizes administrators to execute user and group administration effectively with friendly user interface. The application exploits the benefits of CDO, ADO, LDAP, ADSI, XML and COM technologies. The application divulges the benefits of utilizing these technologies and imparts a valuable source of example for incorporating CDO using VC++. The application supports and imparts the benefits and implementations of callback methodologies incorporating COM connection points and sinks to initiate instant alerts to the clients. The application can also be perceived as a viable example for uploading attachments using binary stream assimilating XML methodology. The eManager application utilizes the effective implementation of multi-process and threads.

It is a suitable exemplar in explaining how custom controls can be dispensed using simple regular MFC controls, and how events can be handled effectively in providing a competent application that can manage all of its processes and threads while providing uncomplicated simple application. The eManager application was developed employing the Object Oriented Analysis methodologies. The application enhanced the functionality, usability, reliability, portability, supportability and upgradeability of the current enforcement system while integrating user and group administration by system administrators.

Introduction

Microsoft Exchange 2003 is a messaging and collaboration platform that provides e-mail, scheduling, online forms, and tools for custom messaging and collaborative applications. A successful Exchange 2003 development hinges on many aspects; these include a strong dependency on Windows 2003 Active Directory (AD), Windows 2003 Internet Information Server (IIS), a properly configured DNS (Domain Name Service) infrastructure, sufficient and reliable hardware and good operational practices. The core of Exchange development paradigm is the Web Storage System. The Web Storage System combines the functionality of an intranet Web server, a database server and a collaboration server. One of the most significant enhancements in Exchange 2003 Server is the wide variety of data access options available. Most importantly, the OLE DB provider ExOLEDB, is high performance server-side provider that interacts with folders, items, and files in a Web Storage System. ADO and CDO for Exchange use ExOLEDB to interact with Exchange on the server. For years, earlier versions of ADO have been used to access data stores, such as MS Access and SQL Server. ADO 2.8 presents a high-level, easy-to-use interface and a low-level, high-performance interface to practically any data store available using the ExOLEDB provider. CDO for Windows 2003 is one in a suite of collaborative COM components referred to collectively as CDO and unlike

previous CDO components; CDO for Windows 2003 is not MAPI-based. To send messages using CDO, you must have network or local access to an SMTP or NNTP service. These restrictions on CDO make it challenging for application programmers. The fast changing technology, growing need for applications developed with record-breaking time constraints, forced almost all of the support available on CDO to VBA and VB programming languages. Though VC++ can effectively and efficiently deliver light weighted applications using MFC, COM and ATL, there aren't much published help available. The main objectives of the eManager application development are,

- Provide comprehensive aid with examples for the development of custom applications using CDO
- Expose the MS Exchange CDO collaboration capabilities to applications on the server
- Demonstrate how CDO for Exchange complements ADO, rather than competing with it
- Demonstrate how ADO for Exchange can be used to create new resources, delete unwanted resources, copy and move resources, and even query a Web Storage System.

The eManager application is designed to deliberate the effective implementation of user administration and user management using Windows 2000 Active Directory. The application configurations enable the system administrators to perform Microsoft Exchange 2000/2003 Server user and group administration in synchronization with Active Directory from within the "eManager' application. The application also

establishes and imparts one of the indispensable methods of implementing the Tab Control in the client area rather than presenting it as a property page using easy to adapt methodology in establishing the Tab Control.

One of the most important means of communication in windows is Messages. The Windows concept is different than traditional programming. The way programs in windows responding to events are called messages. The messages signal many events, caused by user, operating system, or other programs/applications. The eManager application is a viable exemplar to inter-process windows message processing. The inter-process communications enable the application to be independent of other applications while reinforcing the established behavior of the application to specific windows messages.

eManager is an advanced e-mail software program that allows users to send and receive e-mail. In addition to e-mail, eManager has a personal calendar, group scheduling, personal contacts, personal tasks, the ability to collaborate and schedule with other users. The eManager authorizes administrators to execute user and group administration effectively with friendly user interface.

Project History

RedySoft Inc is an outstanding IT Systems Consultants and is staffed with high-tech professionals with ten to twenty years of experience with proven consulting methodology, contribution to cost effectiveness in implementing new technologies, exceptional network team to analyze and to recommend purchase, configure and install hardware and software and reduce the operating costs by 60% through improved performance with uncompromising security features in place.

Redysoft was developing advanced applications that were capable of providing operational information they needed to provide an effective and modernized application that was capable of rendering instant messages to Law Enforcement agencies, and other State and Federal agencies. The instant messaging system with tracing features is the growing need of the hour. The modernized system will dominate the supremacy in instant messaging system in accordance with State and Federal regulations and guidelines.

I have been developing applications in IBM Mainframe platform with COBOL programming language for sometime. The introduction of Microsoft Visual C++

definitely changed my view of application development. The eManager application system is designed to provide enough effective samples that outline the process of working with CDO for Microsoft Exchange. As a result, the application is built to explore and expose the facts of working with CDO using VC++, while effectively implementing the strengths of MFC.

The development of eManager system was unique and involved a degree of uncertainty. The system was designed and developed by only one member and involved analysis and design, project plan development, leading and managing the different parts of the project management including establishing direction, motivating and inspiring client and communicating with clients and other project stakeholders. The project negotiations and their outcome, and agreements were documented in detail. The negotiation and decision-making includes analyzing the problems, their probable cause, choice solutions and recommendations. Once the decision is made, the decision is documented and implemented. The eManager system development lifecycle has been a very powerful tool to me in understanding the mechanics of power and politics in application development and implementation.

Systems Analysis and Design

The eManager system is designed and developed by way of Object-Oriented analysis and Iterative development process using Unified Process methodologies. The Object-Oriented Analysis methodology proffer and outstand in providing methodical approach in dealing with projects that are constrained by ruthless time constraints through Iterative development process. OOA also provides the maintainability through simplified mapping to the real world, which provides for less analysis effort, less complexity in system design, and easier verification by the user; reusability of the analysis artifacts which saves time and costs; and depending on the analysis method and programming language, productivity gains through direct mapping to features of Object-Oriented Programming Languages [BDN96].

Systems Analysis

The eManager system development requires a short initial step expose the vision and business case with feasibility examination. The system requirements, capabilities and

conditions the system supports are of prim challenge. The system requirements were classified into different categories as shown in table below.

Artifact	Comment
Vision and Business Case	Implement the email subsystem providing instant messaging with message tracking capability
Use-Case model	The functional requirements and related non-functional requirements are documented
Supplementary Specification	Windows 2000 Operating System and Microsoft Exchange 2003
Risk and Risk Management	Delay in instant messaging system implementation due to technological training required for application development team members
Conceptual Proofs and Prototypes	The entire design section provides innumerable examples and prototypes that provide in depth realization of concepts required

Functional Requirement

In product development, it is useful to distinguish between the baseline functionality necessary for any system to compete in that product domain, and features that differentiate the system from competitors' products, and from variants. What makes the product lines part of a family, are some common elements of functionality and identity. These strategies have important implications for software architecture. In

particular, it is not just the functional requirements of the first product or release that must be supported by the architecture.

Usability Requirement

The user requirements effectiveness, efficiency and satisfaction for the user are considered. The acceptable task time and optimum target, well-disposed contingency plans, the understandability, learnability, operability and attractiveness are analyzed. The understandability provided the interface elements such as menus that are easy and catchy. The system provides a gradual but steady increase in learning curve while maintaining the simple context sensitive help and effective documentation. The system actions and elements are consistent with most email applications. The system screen layout and color layout are appealing and customizable. [SER01]

Reliability Requirement

Most organizations are concerned with the reliability of their products; many will not develop good reliability specifications. While this is once again an admirable aim, if there is no numerical measure of "customer expectations," the requirement is essentially useless. Ideally, customer measurement programs and input should be used to develop reliability goals, but in a more quantitative manner. Another common pitfall when it comes to specifying product reliability is the use of the mean life or MTTF (mean time to failure) metric. We have seen that non-quantitative statements and MTTF values are not adequate for reliability specification. [REL01] The reliability requirements should adequately address the following three components:

- A specified reliability
- A time associated with the specified reliability
- A desired confidence level

Supportability Requirement

Developers of modern systems face challenging ownership and availability requirements, and need cost effective techniques for satisfying them. These measures are used because customer satisfaction is determined by system performance, cost, and ability to perform when needed (or operational availability). During allocation, a portion of the cost of ownership and availability (or more accurately down-time) is apportioned to each lower level element of the system. [PBS97]

Performance Requirement

Performance specifications leave out unnecessary "how to" or detail and give the manufacturer latitude to determine how to best meet our stated needs. The specification enumerates the interface requirements necessary to allow maintenance at the appropriate level, but it must not impose a design solution beyond that necessary to ensure a proper interface.

Development Methodologies

The Unified Process is use-case driven. During the analysis the use cases were described in terms of the classes of the software product. The UP does not describe how classes are to be extracted in OOAD. The entity class extraction process consists of three steps that are carried out interactively and incrementally:

1. Functional modeling
2. Entity class modeling
3. Dynamic modeling

Functional Modeling

The Unified Process defines the Use-Case Model within the requirements discipline describes the functionality and environment of the system. The Use-Cases provide a mechanism that help stakeholders understand the system indicating what the system will do. The UML defines a use case diagram to illustrate the names of use cases and actors and their relationships. The Use-Cases are written in different formats, depending on the needs. Use Cases that describe system responsibilities specific to what the system must do (functional requirements) without specifying how it will do it (the

design) are known as black-box use cases. Use cases that describe system responsibilities that specify how the requirements are fulfilled are known as white-box use cases.

The system is perceived as having high quality in meeting the user needs. The users of the system and the tasks they must undertake with the system were identified. The actors are identified as a user of the system in a particular role. The use cases are recognized as, tasks the actor needs to perform with the help of the system. The system boundary is clearly identified. The system boundary is clarified to identify the system responsibilities. The **Figure 15** shows the context diagram of the eManager system. People who need assistance from DS contact the enforcement workers, who then identify the needs of the client and their eligibility for certain specific programs. The enforcement workers also identify and discover if there is a need to initiate criminal enforcement activities against the other parties involved. In case of legal proceedings the enforcement attorneys are informed to initiate any legal actions required. The above process identifies the actors and their needs the eManager system has to perform.

The actors and their primary goals in terms of UP artifacts are identified and documented as shown in **Figure 1**. The use case embodies a, possible complex, set of requirements on the system, which starts to emerge during the initial requirements capture and are refined as the system is developed. The possible scenarios and what determines which of them applies in any given set of circumstances are analyzed and documented.

Entity Class Modeling

The entity class modeling determines the entity classes and their attributes. The interrelationships and interactions between the entity classes are identified. The attributes not methods of an entity class are determined. There are two different ways of determining the class model.

1. Noun Extraction
2. CRC Cards

The eManager system was developed with extensive use of the CRC cards method as part of the UP/Object-Oriented analysis workflow [WBW90] The CRC model (**Appendix B: CRC Cards**) was developed by filling in a card showing the current name of the class, its functionality (responsibility), and a list of the other classes it invokes to achieve that functionality (collaboration).

The class diagram that describes the entity classes was identified in the use case analysis and their relationships. The CRC cards for the eManager system were developed by creating one CRC Card for each of the entity class identified in the Functional Modeling (Use Case analysis). Each of the CRC Cards was named with the entity classes and their name written <http://c2.com/doc/crc/draw.html>(May 19, 2005 16:25). The responsibilities of each of the entity classes were determined by examining the class diagram. [RMD] The collaboration details of each of the classes were determined as the interaction of individual class with other classes in the domain. [KBWC] The analysis using CRC Cards brought in the design reviews and a framework for implementation into the center stage by providing an informal notation. [OOA96], [CAC]

Dynamic Modeling

The dynamic modeling determines the operations performed by or to each entity class or sub-class. The dynamic modeling produces a statechart, a description of the target product and is not a complete representation of the product to be built but consists of a set of transition rules of the form,

Current state and **event** and **predicate** => **next state** [SCH05]

The eManager state diagrams were designed to capture the stimuli, responses, and actions of the classes identified using use cases and CRC Cards. The state diagram shows the events that cause a transition from one state of a class to another state. It is a network of states and events, and records the dynamic behavior of important classes in the system. The state diagram was developed just for the MESSAGE class based on various events. The state diagram clearly identifies the extent to which the current state of the class will continue and when will it change.

The State diagram shows the events that cause a transition of a class from one state to another state. It is a network of states and events. It captures the class's received stimuli, responses, and actions. Each state receives one or more events, at which time the class transitions to the next state. The next state depends on the current state as well as the events. Modeling a state diagram is useful for understanding the dynamic behavior of important classes in the system. The state diagram is build for the classes in the system that necessities further understanding or communicate its dynamic behavior in response to events.

The state diagram shows the lifecycle of an object; what events it experiences, its transitions, and the states it is in between these events. It does not illustrate every possible event; if an event arises that is not represented in the diagram, the event is ignored as far as the state diagram is concerned. This helps create state diagram that describes the lifecycle of an object at arbitrarily simple or complex levels of detail, depending on the requirements. [SDR05] The state diagram help designer methodically develop a design that ensures correct system event order. The diagram (**Figure 34**) describes the different states realized by the message class. The diagram delineates how the object reacts to various events extensively.

Overview of Technology

Building a system that requires a great deal of communications between both windows applications and web applications is demanding. Although CDO is widely used it is considered to be a lower level of programming when developing with MFC using VC++. The underlying application artifacts are provided by MFC and Microsoft XML technologies.

The MFC provides the framework to make Windows objects such as windows, dialog boxes, and controls behave like C++ objects and help create self contained and highly reusable classes that can respond to its own events. The MFC provides an object-oriented interface to the Windows operating system that supports reusability, self-containment, and other tenets of OOP. MFC also provides these without imposing undue overhead on the system or unnecessarily adding to an application's memory requirements.

XML has become the standard for exchanging data in many organizations. However, the Microsoft Internet Explorer (IE) XMLDOM via COM (Component Object

Model) is used in order to use XML from MFC. Using the XMLDOM doesn't compare to the very elegant and powerful set of .NET XML classes. As a result, the .NET Base Class Library (BCL) is used with MFC applications to provide ultimately more productive and marketable applications.

"A collaborative application facilitates information sharing and management, allowing groups to work together across an organization."[MSD05] Many types of collaborative applications can be developed on the Exchange store, such as Messaging applications, Workflow applications, and applications that allow users to communicate in real time. The CDO and CDO Rendering Libraries are used for building collaborative Web server applications on Microsoft® Exchange Server. The CDO Library can be used to build both client and server applications, but the CDO Rendering Library can be used only for server applications. Server applications integrated with Active Server Pages and a browser can provide Web access to the client, namely mailboxes, and public folders. This library lets you add the ability to send and receive mail messages and to interact with folders and address books from client applications. The eManager application design enables the user to interact with the Exchange store through WEB and client server architecture.

Active Directory

In Windows 2000, Active Directory manages all the user and group information in a Windows domain. Exchange 2000 turns over the user and mailbox management responsibilities to Windows 2000 and Active Directory. Windows 2000 manages the

domain accounts, user mailboxes along with the distribution groups for exchange. Client applications can communicate with Active Directory using Active Directory Service Interface (ADSI) and Lightweight Directory Access Protocol (LDAP). [CMN00]

LDAP

The Lightweight Directory Access Protocol (LDAP) is a directory service protocol that runs directly over the TCP/IP stack and permits low-level access to a directory. LDAP supports C and C++ programming languages and is applicable to directory management and browser applications that do not have directory service support as their primary function. The LDAP cannot create directories or stipulate how a directory service operates. A directory entry in LDAP is represented by its entry name, or relative distinguished name (RDN), and by its distinguished name (DN). The LDAP API provides functions that allow LDAP client applications to search for and retrieve information from an LDAP directory server, as well as functions for modifying directory entries, where such modifications are permitted. Microsoft also provides the Active Directory Service Interfaces (ADSI) for developing client-side directory service applications using COM interfaces that enable applications access the network directory services for Windows Operating system. The ADSI uses LDAP to communicate with the Active Directory [CMN00]. The diagram (**Figure 16)** explains configurations of LDAP and ADSI.

ADSI

The Active Directory Service Interfaces (ADSI) are Component Object Model (COM) interfaces that provide an abstraction layer for manipulating resources stored in a directory service. ADSI bolster access and maintenance of different network providers in a distributed network environment. The ADSI accomplishes common network administrative tasks, such as adding new users, managing printers, and locating resources in a distributed computing environment unchallenging. [CMN00] The ADSI requires components to reside on both the server and the client. In addition, ADSI supplies its own OLE DB provider, so that any client already using OLE DB, including those using ActiveX® Data Objects, can query directory services directly [MSD052]. The diagram (**Figure 17)** illustrates the communication between client and server using ADSI through LDAP wire protocol. Here the client requests the appropriate directory service through the application that issues the ADSI requests. The ADSI then distributes the application request to the LDAP. The LDAP services the request from ADSI in accordance with the Active Directory in the server and the results are returned to the ADSI, which in sequence returns the results back to the application. [CMN00]

CDO for Exchange

CDO for Windows 2000 is one in a suite of collaborative COM components referred to collectively as CDO. CDO provides a number of objects and interfaces for managing users and mailboxes. Unlike ADSI, CDO for Exchange cannot be installed on a client and it requires the Exchange OLE DB (ExOLEDB) provider that can only be installed on the Exchange 2000 Server. Any application that requires CDO to interact

with Active Directory is forced to run on the Exchange Server. CDO consists of three object models, each of which serves a unique purpose. CDOEX provides the fundamental interfaces and Component Object Model (COM) classes that are used to manage most types of items in the Exchange store. The Collaboration Data Objects (CDO) Workflow Objects for Exchange and the CDO for Exchange Management (CDOEXM) COM components extend this core component to provide additional functionality. The diagram (**Figure 18**) enlightens how the client application can interact with the Active Directory using CDO through LDAP. CDO and ADO enhance one another rather than compete and both use the ExOLEDB. The ADO is optimized for navigating, searching, and setting properties within the Web Storage system and specializes in generic resource manipulation tasks such as copying, moving records. CDO is optimized for creating messaging, calendaring, and contact-management systems. CDO can also be used to build server-management tools and complex routing systems. The above elucidation is the basis for the eManager application to exclusively incorporate both CDO and ADO. The eManager system also exploits the preeminent features of ADSI and LDAP wherever applicable. [CMN00]

COM Component Design

COM is a system API that allows the applications to access the functions and data in another application (EXE) or a dynamically linked library (DLL). COM methods can be called from C++ with no different than calling any other C++ methods. Creating COM object in C++ is achieved by specifying what file the object is in and what object inside needs to be created. [JSW00] The actual API appears like the following:

```
STDAPI CoCreateInstance(
  REFCLSID rclsid,
  LPUNKNOWN pUnkOuter,
  DWORD dwClsContext,
  REFIID riid,
  LPVOID * ppv
);

Example:
CoCreateInstance(__uuidof(DOMDocument40)
, NULL
, CLSCTX_INPROC_SERVER
, IID_IXMLDOMDocument
, (void**)&pIXmlDomDocPtr_eManager);
```

The diagram (**Figure 19)** explains how C++ communicates with COM. Initially the C++ calls the go-between function called proxy. This proxy actually serializes the calling arguments into the standard COM client/server protocol and sticks it onto the channel to the COM EXE where another function called stub unpacks it and finally places the call to the method. Hence, when the CoCreateInstance() is called to create the object starts the COM EXE, creates the object and return the pointer generated by the proxy. The system registry helps in identifying the EXE/DLL that is associated with interface requested by CoCreateInstance method.

It is vital for the client to notify the user of any event that interests the user. The modern design and requirements outline the growing need for the server to notify the client of an event than the client waiting for some events to occur at the server. COM provides two standard ways for a server to communicate with its client that uses both of those solutions.

In Win32, when one function passes an address to another function to call when some event occurs, that address is called callback address. In COM such events are known as connection point and sink where one or more clients can give a server a callback address that the server will call when something happens The address the server calls in a client is considered a sink and a server that can do this is considered to have a connection point. COM also implements early binding and late binding interfaces analogous to early and late binding interfaces. [JSW00]

MFC and ATL (Active Template Library) support early binding connection points and sinks. This can be implemented by creating a COM interface project for that client with its own COM class and methods (**Figure 20)**, IDL file and proxy/stub DLL. A client class derived from CCmdTarget – such as the main window (CMainFrame or CDialog) – and sticks the same MFC macros in its .h file that can be used as a regular MFC COM server to implement a COM class described below.

```
DECLARE_INTERFACE_MAP()
      BEGIN_INTERFACE_PART(SinkCls, IID_sink)
            STDMETHOD_(HRESULT, Callback)(long);
      END_INTERFACE_PART(SinkCls)
```

In the example, IID_sink is the name of the client's COM class interface, SinkCls is the implementation of that COM class interface and Callback is the method of that COM class. The early bindings calls COM methods directly and are efficient. [XYL01]. The eManager application implements early binding interface methodology to provide

real-time messaging to notify the user of events such as, the arrival of new email, appointment reminders and other critical events.

Application Design

The eManager application is a working demonstration of delivering a successful application developed and implemented on time within budget using Object Oriented Analysis and Design with Unified Process. The application analysis and design, and technological artifacts described previously contribute extensively in the development of the eManager application.

Real-world enterprise applications are seldom single, monolithic systems and most enterprise applications must cooperate with multiple data sources and enterprise information systems. The eManager application separate modules while maintaining the functional requirements of the business need. The result is a decoupled enterprise architecture that can interoperate with existing Microsoft Exchange Server 2003 data store. [SADI]

As discussed earlier, the exchange stores information in both web storage systems and active directory. The exchange uses the web storage system to store resources such as folders, items, and files, and it uses the active directory to store and manage data about

exchange mailboxes. The dual information source means the application often needs to access data from both web storage system and active directory. The eManager development application architecture accesses the web storage system using ADO, CDO, XML, and optionally using HTTP. The eManager application uses CDO to access the active directory, and assimilates extensive use of COM methodologies while utilizing the .NET features. As a result, the .NET Base Class Library (BCL) is used to provide ultimately more productive and robust application.

TAB Control

The tabbed dialog boxes containing pages of controls that the user can switch among with mouse clicks are one of the most appreciated windows programming controls. It is not difficult to add property sheet to MFC applications as MFC encapsulates the property sheet in CPropertySheet and CPropertyPage classes. The CPropertySheet represents the property sheet itself and is derived from CWnd. The CPropertyPage represents a page in a property sheet and is derived from CDialog. The general procedures for creating a modal property sheet are:

1. Create a dialog template for each page in property sheet defining the page contents and characteristics and set the dialog title
2. For each page in property sheet, derive a dialog-like class from CPropertyPage that includes public data members linked to the pages controls via DDX or DDV
3. Derive a property sheet class from CPropertySheet. Instantiate the property sheet class and property sheet page classes defined in step 2
4. Call the property sheets DoModal function to display it on the screen.

The above-described method is well documented and there are plenty of examples and worked solutions available. The advantages of the Tab controls are far beyond just property sheets and property pages. One of the classic and widely utilized examples is Microsoft Outlook client application. The MS Outlook implements the tab control very effectively to display/edit the Contacts details. The foremost difference between implementing the tab controls using property sheets and property page and the method used to implement the tab control in MS Outlook are,

- The tab control is employed as part of the entire client area as part of CDialog class and not in the form of property sheet and property pages.
- Implementing tab control as part of CDialog class extends the MFC framework support to the tab control being implemented. The application developer is encouraged to utilize some of the features such as support for printing, etc with no effort provided by the MFC framework. This lessens the work of the application developer in spending more time on reinventing the same functionalities provided freely by the MFC framework.

Designing applications similar to MS Outlook Contacts dialog appear to be very simple and I always thought it is one of the easiest design decisions to implement tab control similar to MS Outlook. Actually, the implementation needs little more tricky approach to accomplish the tab control that fulfills the requirements in its entirety in providing the easy look a like approach. There are many different solutions to software

development and there is no wrong approach, but some are elegant than others. The methodology adapted to implement the tab control in eManager application is very flexible and is somewhat variation to the solution provided in the article “CTabCtrlSSL – An easy to use, flexible extended tab control” by Derek Lakin <http://thecodeproject.com/tabctrl/ctabctrlssl.asp>. The eManager application implements tab controls using accepted MFC CTabCtrl class in following steps.

1. Design the dialog resource for each of the property pages in the tab control.
2. Add each of the dialog resources to a class derived from CDialog.
3. Add empty methods to handle the default OK and Cancel events for each of the dialog resource designed.
4. Create a new class derived from CTabCtrl class.
5. Add an integer array member to the CTabCtrl class. The size of the array is equal to number of tab control dialogs designed.
6. Initialize the integer array with the resource Ids of the tab control dialogs.
7. In the InitDialog handler for the dialog class create the tab control class generated and add pages to the tab control.
8. Set the current selected tab dialog.

The steps described support most requirements and is the simplest solution to implement the tab control using CDialog. The above implementation has its own anomaly. Events handling and responding to events that transpire from various controls that occupy each of the tab control dialogs are not new to Windows application

development. Responding to list box control events was one of the requirements of the eManager application. As the tab control dialogs were implemented as described above, it became evident that the framework failed to instantiate the classes described for each of the tab control dialogs. This caused an undocumented extraordinary situation. There were enormous amount of research performed on this topic and almost all were found not fertile. It was after much efforts and research eManager design was slightly altered so those specific tab control dialogs that require specific event handling were designed dynamically. Each and every control on those specific tab controls were created, and their properties and event handling specified dynamically at run time. Implementing the tab control dialog dynamically deciphered the predicament. It was one of the situations where an easy and simple solution provided at the appropriate place and time helped the eManager application thrive.

```
void CContactsTABCls::OnSize(UINT nType, int cx, int cy)
{
	int i=(int)cx, j=(int)cy;
	CRect pRect;
	this->GetParentFrame()->GetClientRect(&pRect);
	pRect.top=0;
	pRect.right=pRect.right-2L;
	this->AdjustRect(true, pRect);
	pRect.top=0;
	this->MoveWindow(pRect, true);
	this->Invalidate(true);
	this->GetParentFrame()->Invalidate(true);
}
```

Tab Control Dialog Re-Positioning

The discussion above provided the approach of integrating the tab control to dialog window. The tab control dialogs attached to the dialog window now occupy the default window position in the client area. This default window position may not

necessarily be the appropriate place for the tab control dialogs and in most cases the dialogs are ungallant and are not user friendly. Research proved that this conundrum could be resolved by tweaking the tab control dialog position in accordance with the client area of the tab control itself in the handle for OnSize() for the class derived from CTabCtrl control class. This technique provided the much-desired outlook for the GUI.

Custom ComboBox Control

The ComboBox control to provide the list of obtainable contact's details such as home address, business address, home phone number, business phone number, mobile phone number, home fax number, business fax number, primary and optional email addresses. The standard ComboBox control provided by framework offered a strange appearance on the application GUI. It was far from the appearance provided by MS Outlook and it was decided to initiate a customized control that can provide satisfying appearance and boost the UI of the eManager application. The methodology adapted to implement the custom ComboBox control in eManager application is very flexible and somewhat variant to the solution provided in the article "A Custom Group Combo Box" by Brett R. Mitchell <http://thecodeproject.com/combobox/customcombo.asp>.

The requirements of the custom combobox control was analyzed and studied thoroughly. The underlying parts of the control were documented as shown in the class diagram (**Figure 23**). The composite relationship between CMButtonCtrl class and CMListCtrlWnd class establishes a special kind of aggregation, which does impose some

further restrictions. In a composition association, the whole strongly owns its parts: if the whole object is copied or deleted its parts are copied or deleted with it. The multiplicity at the whole end of the composition association must be 1 or 0..1 – a part cannot be part of more than one whole by composition. The composition association is shown just as aggregation is, except that the diamond is filled in. [PLY00] The ComboBox control can be implemented by using the steps described below.

CMListCtrl class:

1. Define a class derived from CListBox class.
2. Implement the Create method.
3. Implement DrawItem method. This method draws the list item with specific text color, text background color and selected item background color.
4. Implement the MeasureItem method. This method measures the width, height of text using current font setting.
5. Implement a method to add text items to the list.
6. Implement a handle for selected item changed event. (This method also sends corresponding message to the view class of the application so the contact information is displayed appropriately.)

CMListCtrlWnd class:

1. Define a class derived from CWnd class.

2. Implement the Create method. As part of the composition association specified in the class diagram, this method also creates an object of the class CMListCtrl class.
3. Implement the OnSize handle. As the size of this class changed, reposition the CMListCtrl accordingly.
4. Implement the OnShowWindow handle. This method should enforce the redrawing of the parent window when the control goes off focus.

CMButtonCtrl class:

1. Define a class derived from CButton class.
2. Implement the Create method. As part of the composition association specified in the class diagram, this method also creates an object of the class CMListWnd class.
3. Implement the DrawItem method. This method draws the control text with specified text color, text background color and the control background color and control border.
4. Implement the OnMeasureItem handle. This method measures the width, height of text using current font setting.
5. Implement the DrawArrow method. This method measures the rectangle size of the CMButtonCtrl control and draws the arrow at the right of the text item.
6. Implement the handle for Left Button click. When the CMListCtrlWnd class object is not visible, determine the total number of list items added and determine

the size of the rectangle that occupies the list control. Reposition the CMListCtrlWnd class object and display the list control window.

The CMButtonCtrl class object is utilized to implement the custom ComboBox control. The initial ComboBox control text is specified when creating the CMButtonCtrl class object. The window is displayed to the user. When the ComboBox control item is selected or changed, the text displayed on the ComboBox button is changed respectively. This technique provided the much-desired outlook for the GUI using the custom ComboBox class control. The discussion above provided the approach of integrating the custom ComboBox control to dialog window.

Windows Messages

The Windows Operating system and the applications that run generate messages for every event that occurs in Windows. Messages are fundamentally important to the value of Windows as a multitasking operating system. Windows generates messages for every hardware event that occurs and it then passes each message to the appropriate message queue. Occasionally the system generates several copies of a message that it simultaneously places in multiple message queues. A message queue is a place in memory, which stores messages that are transferred between applications. [JMS02] Windows has default message handler for almost all messages providing a default behavior. As discussed above, Message handling is one of the prim aspects of the Windows Operating System. The motivation for this discussion is to highlight some of the established methods in which messages can be effectively implemented within

applications without impinging the functioning of the Windows OS. This discussion does not address the MFC message-handling methods using message handlers provided by classwizards. It is implied that this discussion is meant to deal with something beyond and to highlight the message handling that are not supported by the classwizards. The discussion deals with widely used three distinctive message-handling techniques,

1. Message-handling using ON_MESSAGE macro
2. User-Defined Messages/WM_APP constant Messages
3. Registered Windows Messages

ON_MESSAGE:

MFC provides a generic macro feature that can be suited for any special requirements using ON_MESSAGE. The prototype of the message ON_MESSAGE handler is as shown. Here ID_ON_CUSTOM_EVENT is the message and OnIDCustomEvent is the method, which is invoked on the message.

```
afxmsg LRESULT OnMessage(WPARAM wParam, LPARAM lParam);


Example:
BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
      ON_MESSAGE(ID_CUSTOM_EVENT, OnIDCustomEvent)
END_MESSAGE_MAP()
```

User-Defined Messages/WM_APP constant Messages:

It is sometimes discernible to implement message handling between two different windows of an application or even more so between two different applications as well. In situations like these the User-Defined messages can be exploited. The Windows OS predefines all standard messages and their number identifies the messages. The user-defined messages are defined by just using one of such numbers from WM_APP through 0xBFFF. [WMH200], [WMH100]

```
#define WM_INQUIRE WM_APP + 0x101

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
       ON_MESSAGE(WM_INQUIRE, OnInquire)
END_MESSAGE_MAP()
```

Registered Windows Messages:

The RegiserWindowMesage provides the potential to communicate between cooperating applications by defining a new window message that is guaranteed to be unique throughout the system. [MSD053] The registered window messages can be used to post or send messages between application boundaries and WM_APP constant messages described earlier can be used within the same application.

```
// CContactProcView1 class
static const UINT CDLG_CLOSE=RegisterWindowMessage("CDLG_CLOSE_{B7A832D1-
8B2D-4312-8A15-874284D868F2}");

void CContactProcView1::OnClose()
{
	// TODO: Add your message handler code here and/or call default

	CListView::OnClose();

	SendMessage(CDLG_CLOSE, 0, 0);
}

// CMainFrame class
static const UINT CDLG_CLOSE=RegisterWindowMessage("CDLG_CLOSE_{B7A832D1-
8B2D-4312-8A15-874284D868F2}");

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd)
	ON_REGISTERED_MESSAGE(CDLG_CLOSE, OnCloseCDlgBar)
END_MESSAGE_MAP()
```

In the example, the CContactProcView1 class uses the registered window message CDLG_CLOSE. The message is registered using RegisterWindowMessage and to make the message unique across systems a GUID is attached to the definition as shown. When the CContactProcView1 class sends the message, the CMainFrame class receives it where the same event is registered to provide the handle. The eManager application utilizes the entire message handling techniques described extensively to provide elegant method of message handling, while maintaining the source code that is precise but uncomplicated for the system maintenance.

Process and Threads

A process can be defined as an executing program. A process may contain one or more threads. A thread is a path of execution within a process. Each time Windows initializes a new instance of a process, the OS creates a new primary thread for that process. The primary thread starts when Windows loads the program. The thread intern calls WinMain function and continues to execute until WinMain function ends processing, and the program calls ExitProcess to end itself. However, process can create additional threads to help accomplish certain tasks. A new thread can be created using CreateThread method. The new process and the new thread handle inherits all the access rights from the process thread that created it and the CreateProcess method can also specify security attributes using SECURITY_ATTRIBUTES structure that determines whether the child processes can inherit the returned handle. When the function provides a security descriptor, the program performs an access check on all subsequent uses of the handle before it grants access. If the access check denies access, the requesting process is not able to use the handle to gain access to the thread. [JMS02]

The eManager application was designed to extend the benefits of Dynamic Link Libraries (DLLs) beyond their basic library support and to incorporate multitasking and multithreading. Multithreading using DLL led to very intensive research and analysis for a considerable period of time with no foreseeable and acceptable solutions available to fulfill within the time constraints. The application design was restructured and modified to incorporate multithreading/multitasking using multiprocessing technique. This led to the utilization of multiple processes while controlling them within the application domain

effectively using Windows messages discussed earlier. The processes were created using CreateProcess method.

The example below shows how eManager application creates a process when the user selects to create a new message. The CreateProcess invokes the MessageProc.exe application by passing the parameters “E EditMessageID”. This creates the thread with an initial stack whose size is described in the image header of the specified program’s executable file. The thread begins execution at the image’s entry point.

```
void CMainFrame::Newmailmessage()
{
STARTUPINFO si;
PROCESS_INFORMATION pi;

ZeroMemory( &si, sizeof(si) );
si.cb = sizeof(si);
ZeroMemory( &pi, sizeof(pi) );

if(!CreateProcess(NULL
, "MessageProc E EditMesageID"
, NULL
, NULL
, TRUE
, CREATE_NEW_PROCESS_GROUP
, NULL
, NULL
, &si
, &pi
))
{
AfxMessageBox(" New Message Creation Failed ", MB_OK);
return;
}

// Wait until child process exits.
//WaitForSingleObject( pi.hProcess, INFINITE );
// Close process and thread handles.

CloseHandle( pi.hProcess );
CloseHandle( pi.hThread );
}
```

Inter Process Communications

Communications between two different applications can be accomplished in many different techniques. The eManager application exploits the benefits of events and process and threads both collectively and individually. The discussions on events and process and threads delineate how they are achieved in eManager application. The eManager utilizes the integration of process, threads and events in a unique approach. There are situations wherein they demonstrate their effective implementation in applications. The CreateToolhelp32Snapshot function takes a snapshot of the specified processes in the system, as well as the heaps, modules, and threads used by these processes. The Process32First function retrieves information about the first process encountered in the snapshot and Process32Next function retrieves information about the next process encountered in the snapshot. Each process can be associated with multiple modules. The Module32First function retrieves information about the first module associated with a process or thread and Module32Next retrieves information about the next module associated with a process or thread. These methods can be used to retrieve information about the processes or thread and modules associated with them. eManager application utilizes these functions to obtain information about the processes or threads of specific interest. Once the process and their modules are identified the application then sends predetermined events to those processes. The eManager application effectively implements this method in the following areas to achieve specific outcome.

1. To effectively implement the unauthorized access to different modules and enforce the access to specific modules only through specific actions such as preventing access to different modules without login to eManager application.
2. Complete the “message delete” process to reflect the changes to the client instantaneously
3. Complete the “save message as draft” to reflect the changes immediately available to client
4. Complete the “send new message” to reflect the copy of the message available in sent items.
5. Do cleanup process when application ends or terminates.

Accessing Web Storage System using SQL

The Exchange provides a variety of data access application programming interfaces and protocols. This discussion highlights the Structured Query Statements (SQLs) that can be used to access the Web Storage System effectively. The GetChildren method of ADO Record Object get all the child records including items and folders but the SQL help restrict only the subset of resources or use only one of the properties of each of the resources returned is to be obtained. The Exchange supports several SQL commands including SELECT, WHERE, ORDER BY, CONTAINS and GROUP BY but does not support AVG, CONVERT (use CAST), COUNT, CREATE VIEW, DATASOURCE, DELETE, DROP INDEX, INSERT, JOINS, MAX, MIN, UPDATE, SET and SELECT DISTINCT. It is evident that SQL can be effectively used to query the exchange store. It is sometimes required to determine the total number of resources in the

solution set. The SQL does not support the COUNT but this can be achieved by using the RecordCount property of the RecordSet object and the DELETE can be achieved by using the DeleteRecord method of the RecordSet object.

The SQL uses the concept of tokens for search conditions and each token must be delimited with single quotes. It is important to establish the connection with the database explicitly using Connection object or Command object. The eManager application uses the SQL to determine the URLs of specific resources to further data manipulations using CDO. The SQL query results can also be exported to XML using Recordset object. The basic steps to query the exchange using SQL are,

1. Build the query
2. Establish Connection using Connection Object
3. Open the Recordset object
4. Get fields of the Recordset object using GetFields method.
5. Use GetItem method of Fields object to obtain specific field data

The example below shows how the SQL statement was built using string concatenation. The connection object establishes the connection to one of the default folders for the specified user. The Recordset object opens the SQL using Open method, and the fields and corresponding items are obtained.

```
// Build SQL
char tChrSel[]="Select ",
tChrFrm[]=" \"urn:schemas:httpmail:from\" ",
tChrTo[]=" \"urn:schemas:httpmail:to\" ",
tChrSub[]=" \"urn:schemas:httpmail:subject\" ",
tChrDtRcvd[]=" \"urn:schemas:httpmail:datereceived\" ",
tChrFrom[]=" From SCOPE('SHALLOW traversal of
\"http://Server.com/exchange/userID/Inbox/\"') ",
tChrWhere1[]=" Where \"DAV:iscollection\" = false ",
tChrWhere2[]=" And \"DAV:isfolder\" = false ",
tChrWhere3[]=" And \"DAV:ishidden\" = false ",
tChrWhere4[]=" And \"DAV:isstructureddocument\" = false ",
tChrOrder[]=" Order by \"urn:schemas:httpmail:datereceived\" DESC ";
char tChrDavHref[]=" \"DAV:href\" ",
tChrDavDateReceived[]=" \"urn:schemas:httpmail:datereceived\" ",
tChrComma[]=" , ",
tChr4096[4096];


// Build the SQL String
strcpy(tChr4096, tChrSel);
strcat(tChr4096, tChrDavHref);
strcat(tChr4096, tChrComma);
strcat(tChr4096, tChrDavDateReceived);
strcat(tChr4096, tChrFrom);
strcat(tChr4096, tChrWhere1);
strcat(tChr4096, tChrWhere2);
strcat(tChr4096, tChrWhere3);
strcat(tChr4096, tChrWhere4);
strcat(tChr4096, tChrOrder);

// Establish Connection
hr=pConn->Open(L"http://Server.com/exchange/userID/Inbox/", _bstr_t(),
_bstr_t(), adCmdUnspecified);

// Open Recordset
hr=pRecordSet->Open((_Command*)pCmd, vtMissing, adOpenForwardOnly,
adLockOptimistic, adCmdUnspecified);

// GetFields
pFlds=gRecordSet->GetFields();

// GetItem
tfName0=pFlds->GetItem(0)->GetName().GetBSTR();
```

CDO for Exchange

The CDO uses COM classes and interfaces. The discussions earlier outlined the fundamental ideas on how CDO interacts with other Windows OS artifacts. This discussion delineates the implementation of CDO interfaces and COM classes into the application programs. The CDO interfaces define an encapsulated set of properties, methods, some are specific to certain classes and others have common functionality. A default interface exposes the default functionality for a class and a default interface has the same name as the object it represents but it is prefixed with *I*. In addition to default interface, most classes use at least another interface. For example, Person and Message classes implement IDataSource interface and this helps application programmers achieve their requirements easily. CDO provides named constants for all the predefined schema properties. It uses easier approach than ADO by passing fully qualified schema property name of the associated CDO constant equivalent, if it exists.

```
hr=::CoCreateInstance(__uuidof(Message), NULL, CLSCTX_ALL,
__uuidof(IMessage), (void**)&pMessage);

hr=pMessage->DataSource-
>Open("http://domain.com/exchange/username/inbox/emailID.eml",
             variant_t( (IDispatch*)gConn, true),
             adModeRead,
             adFailIfNotExists,
             adOpenSource,
             _bstr_t(),
             _bstr_t()
             );


pTemp_DomNodePtr=pDoc->createElement("BCC");
pTemp_DomNodePtr->text=pMessage->BCC;
pOut_DomNodePtr=pDomNodePtr->appendChild(pTemp_DomNodePtr);
```

The example above shows how the BCC property of the IMessage interface is extracted using CDO Message class. The example above evidently demonstrates the reliance of CDO on the URLs and the ExOLEDB provider interacts with resources in a Web Storage System. The URL must be properly constructed with both the complete folder path and the display name of the resource being accessed. If opening an item such as contact or email message the .eml extension can be appended. It is a good practice to include a trailing slash mark on a URL that points to a folder [CMN00]. The eManager utilizes the Uniform Resource Locators (URLs) throughout the application to access resources precisely and uses trailing slash when accessing the folders. This standard is adhered throughout the application development to maintain consistency. The Web Storage resources can be accessed using URLs using the DataSource class. The Open method uses the ADO active connection details, access mode, resource open options, user ID and password. The example above shows the implicit authentication where the system enables the authentication based on the Windows domain access specifications. The URL of the resources is obtained using the SQL described earlier. [MSDN05]

The eManager application exploits the strength of XML in data transfer between client and server. The diagram (**Figure 26**) shows the communications between the client and server, employing XML documents. All communications between client and server is converted into XML and the XML document is transmitted and received. The conversion of fields accepted from applications are formatted and interpreted to incorporate binary data transfer. This process requires the conversion of the attachments be converted into

XML before transmitting it to the server. When server receives this XML document the server is then required to convert back the received XML document into corresponding fields and continue to process the data in correlation with their intended purpose and not loose their meaning or translate them on its own.

The IMessage interface of the Message class supports AddAttachment method. The AddAttachment method attaches the attachments within a BodyPart object using Uniform Resource Locator (URL). This attached message can be transmitted using the send method. Though sending messages using this method transmits the message, it falls short of meeting the requirements specifications. To achieve the utmost requirements specifications, eManager handles the email messages and attachments independently. The ADODB::_Stream structure facilitates the conversion of the email Message class object into a stream. The Istream object supports reading and writing data to stream objects. The stream objects contain a structured storage object where the object provides the structure. Methods in this IStream interface present the object data as contiguous sequence of bytes that can be read or written. The eManager application builds the email message using the IMessage interface of the Message class. The attachments are attached to the message as described earlier in this discussion using AddAttachments method of the IMessage interface. The AddAttachments method returns the interface IBodyPart. The GetStream of the IBodyPart interface returns the stream object, which is converted to the required format using the Type property of the stream object. The Read method of the stream object can be used to read the stream object data. This stream object data is converted into XML document and transmitted to the client to be uploaded as part of the email

message. The XML document received by the server is converted back to the original message using the raw_Open method of the stream object by specifying the respective conversion type the stream object data can be read back into the stream object. The IDataSource interface can be used to reinterpret the stream object back into the attachments using the IBodyPart interface. The diagram (**Figure 25**) shows the attachment converted into XML document as transmitted to the Server to be uploaded as attachment of the specified email message.

```
SendMail()
{
	CDO::IMessagePtr	pMessagePtr1;

// Create IDataSource object using Message class
	CDO::IDataSourcePtr pDSin;
	pDSin.CreateInstance(__uuidof(CDO::Message));

// Transmit the message
	hr=pMessagePtr1->Send();

// Initialize IDataSource object with message
	pDSin=pMessagePtr1;

// Save message in Folder
	hr=pDSin->SaveTo( pSrcURL
			, variant_t((IDispatch*)pConnPtr, true)
			, adModeReadWrite
			, adCreateNonCollection
			, (RecordOpenOptionsEnum)NULL
			, bstr_From
			, bstr_Passwd
			);

	pMessagePtr1=NULL;
	pDSin=NULL;
}
```

The new email message can be transmitted using the send method of the IMessage of the Message class. Transmitting the messages using this method establishes the connection between the exchange server and the server relays the message. While the server efficiently handles this, it failed to fulfill the required, widely accepted standard of saving a copy of the sent message in the SentItems folder. Research shows that this can be handled in many different ways. The eManager application utilizes the uncomplicated simple solution by saving the copy of the message in SentItems folder with SaveTo method of the IDataSource interface through Message class. The IDataSource interface enables access message data in other objects. [MSDN05]

Application Enhancements

The eManager application modeled the real world situation requiring the integration of mail capabilities to current application and is continuously changing according to the ever-changing requirements and technological changes. The phase 1 of eManager application integrates the mail capabilities including send/receive email, create personal contacts, delete items, integration of server events, process and threads and application events. The phase 1 lays the foundation for further application enhancements and explorations. The phase 2 of the application will include the integration of Active Directory in addition to e-mail, personal calendar, group scheduling, personal contacts, personal tasks, the ability to collaborate and schedule with other users. In addition to all of the above-mentioned strengthening additions, the eManager application will integrate the virus scan integration of the email messages before transmitting them and before delivering the messages to individual users. The current outward show of the application is adequate in providing and supporting the most pressing requirements, the application necessitates further improvements when integrating the different artifacts mentioned above.

Obstacles, Pitfalls and Suggestions

Saving the messages using the SavetoContainer method of the IDataSource interface was just enough in saving the messages. It was noticed that the messages were saved with a GUID as part of the message title to form the resource Uniform Resource Locator (URL). Research showed that this might create issues at a later stage of the system implementation, limiting the amount of messages that can be stored in the folder itself. This limitation could initiate serious predicaments at the maintenance phase of the application. A solution to this problem was worked out to accommodate the messages without any limitations enforced by the application as a default on the amount of messages that can be stored on users personal folders. The solution incorporates adaptation of the message title into the formation of the URL of the resource. While this unraveled the problem greatly, it created a situation where the system failed to save messages with matching titles. Further analysis on how this issue was handled by Microsoft led to the process of suffixing the URL of the resource with numbers when storing messages with matching title.

The discussion highlighted the importance of effective management of process and threads in fulfilling the requirements and their implementation. There were several

incidents in the development of eManager application where successful message handling between applications and application windows were of prim importance. While there were several approaches to handle inter-process communication messages, the critical need was to decide how these messages would be sent to the specific application/application window. Research showed and directed to utilize the message handling thru windows messages and other means where the message is passed thru Windows operating system. While this method was effective, a more robust implementation was required on specific circumstances. This was again handled by obtaining the process using the snapshot methods explained in the discussions earlier. Here, the process of specific interest is identified and distinguished from the snapshot and the desired message is sent to the application directly. This method provides the much-anticipated effective application performance.

Multithreading an application could fetch untold complications and untold impediments. Initial analysis requirements suggested utilizing multithreading different artifacts of the eManager applications to be implemented as part of DLLs. This fabricated a situation where the multithreading could not be achieved. There were considerable amount of time and with no impending solution that can be achieved within the specified time constraints, it was decided to implement the application modules that require multithread processing as an independent application module managed and organized using effective inter-process communications management. This awards one of the areas for further research where multithreading applications are implemented using DLL.

Applications providing implicit authentication require recognizing the user details including the user name, Domain Name, fully qualified Domain Name, network identification details, etc. Research proved that these requirements could be fulfilled using Secure32.lib of the windows system library. This helps in identifying the user credentials for auditing and tracing purposes.

In the world of Web applications, today, it is hard to locate an application that does not support hyperlink controls and hyperlink capabilities. The eManager application was designed to incorporate the hyperlinks to enable the implementation the addition and removal of attachments. The hyperlink was incorporated and proved to be effective as an independent application distributed as part of the application main window. The eManager application requirements incorporate the hyperlink as part of the dialog bar integrating MFC CDialogBar. Generating the hyperlink class dynamically at runtime as part of the CDialogBar implementation led to difficulty in message handling. The issue was discussed with the client and it was acknowledged to avoid utilizing the hyperlinks as part of the phase 1 of the project due to time constraints. The implementation of the hyperlinks is to be implemented as part of the phase 2 of the project.

Conclusion

The eManager application was built using VC++ utilizing COM methodologies incorporating CDO, ADO, LDAP and Active Directory. The ability to utilize CDO for Exchange is demonstrated beyond regular handling. The application provides convincing examples in achieving the custom controls easily. The eManager application provides the foundation for effective application development using processes and threads, and realizes their effective management and assimilation into successful application design. To ensure that the key software engineering techniques are understood, each and every artifact of OOAD methodologies are introduced and dealt with great detail throughout the application development process. The above discussion outlined the relevant examples and highlighted methods of operation and how each of the artifacts outlined influenced the development of the eManager application.

The software process is to furnish the ability to measure what is happening to the project. The discussion utilized the object-oriented life-cycle model, object-oriented analysis, object-oriented design and the testing and maintenance of object-oriented software. The eManager application development underlines the importance of

documentation, maintenance, reuse, portability, testing and utilization of CASE tools. The Phase 1 of the eManager application is to be integrated with the existing enforcement process applications. The eManager application will reduce the post delivery maintenance costs by 10 percent as post delivery maintenance and application development is treated with no distinction throughout the application development. The eManager application provides the communication solutions that are vital for enforcement agencies to carryout critical tasks efficiently and will bring the next generation technological advancements, while supporting instant messaging, send and receive email, personal calendar, personal and group scheduling, personal contacts, personal tasks and the ability to collaborate and schedule with other users. Most of all the eManager application provides and supports the ability to administer user and group administration tasks by system administrators with friendly user interface.

Appendix A: Use Cases

Use Case UC: eManager System Application

Description: eManager application provides an integrated solution for managing and organizing e-mail messages, schedules, tasks, notes, contacts, and other information for the enforcement workers. The system is an integrated application system. It is designed to provide the essential real-time communication between stakeholders. The application provides critical support as backbone to Rhoden Technologies.

Primary Actor: Enforcement Workers and Attorneys

Stakeholders and Interests:

Enforcement Workers

- The enforcement workers are interested in effective and comprehensive communication between clients, Attorneys and law enforcement agencies
- The enforcement workers need the alert system to alarm them of any critical activities that require their attention and effective judgment in providing valuable service to the clients

Attorneys

- The attorneys are interested to be aware of critical information concerning the client's performance, outstanding amounts owed to the state and federal agencies, effective procedures and practices applied in choosing the legal actions against the client
- The attorneys are interested in identifying any deliberate actions that caused the legal proceedings and their impact on the speedy status changes leading to effective financial aid distribution to the client.
- The attorneys are interested to make decision that are acceptable to eliminate funds being distributed to clients

Frequency: powerfully 24/7

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition: The enforcement workers may choose to respond to the alerts and emails received. The actions alerts require specific tasks be completed/initiated before deadline.

Main Success Scenarios:

1. System initializes the client
2. System authenticates user
3. System transmits the emails.
4. System senses new emails and alerts the user
5. System moves items between email folders
6. System deletes the selected items

7. System displays the selected item for the user to read

Extensions:

1.a. Authentication failure:

1. The system ends process.

2.a. Invalid server connection:

1. The system ends process.

Special Requirements: Client system requirements:

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 1 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-RI00-Read Items

Description: System validates the user input and guide and transfer input requirements to respective Use Case processes based on the operations performed. The display inbox action requires the operations performed in accordance with reading inbox and returning the end results back. The display appointments action requires the operations performed in accordance with reading appointments and returning the end results back. The display contacts action requires the operations performed in accordance with reading contacts and returning the end results back. The display deleted items action requires the operations performed in accordance with reading deleted items and returning the end results back. The display drafts action requires the operations performed in accordance with reading drafts and returning the end results back. The display sent items action requires the operations performed in accordance with reading sent items and returning the end results back.

Primary Actor: System

Stakeholders and Interests:

Enforcement Workers:

- The Enforcement Workers are interested in viewing and reading the message items
- The Enforcement Workers are interested in viewing and reading the drafts items
- The Enforcement Workers are interested in viewing and reading the deleted items

- The Enforcement Workers are interested in viewing and reading contact details
- The Enforcement Workers are interested in viewing and reading appointments

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

The system displays the requested item for the worker to view and read.

Main Success Scenarios:

System performs the following to read inbox items.

1. System displays the inbox message item in accordance with UC-RI01 use case.

System performs the following to read appointment items.

2. System displays the appointment items in accordance with UC-RI02 use case.

System performs the following to read contact items.

3. System displays the contact items in accordance with UC-RI03 use case.

System performs the following to read deleted items.

4. System displays the deleted items in accordance with UC-RI04 use case.

System performs the following to read draft items.

5. System displays the draft items in accordance with UC-RI05 use case.

System performs the following to read sent items.

6. System displays the sent items in accordance with UC-RI06 use case.

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 2 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-RI01-Read Inbox Items

Description:

The system reads the email messages received by the system for the particularized user. The email details are sent to the client through XML and shown to the user.

Primary Actor: System

Stakeholders and Interests:

Enforcement Workers:

- The Enforcement Workers are interested in viewing and reading the inbox message items

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

System authenticates the worker successfully.

Main Success Scenarios:

System performs the following to read inbox items.

1. System connects to the server
2. System reads the inbox records
3. System generates the XML
4. System returns the XML details to the client
5. System interprets the XML
6. System loads the records into the grid in the form

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 3 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-RI02-Read Appointment

Description:

The system reads the appointment details received by the system for the particularized user. The appointment details are sent to the client through XML and shown to the user.

Primary Actor: System

Stakeholders and Interests:

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

The system displays the requested item for the worker to view and read.

Main Success Scenarios:

System performs the following to read Appointment items.

1. System connects to the server
2. System reads the appointment records particularized for the user
3. System generates the XML
4. System returns the XML details to the client
5. System interprets the XML
6. System loads the records into the grid in the form

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 4 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-RI03-Read Contacts

Description:

The system reads the contact details received by the system for the particularized user. The contact details are sent to the client through XML and shown to the user.

Primary Actor: System

Stakeholders and Interests:

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

The system displays the requested item for the worker to view and read.

Main Success Scenarios:

System performs the following to read Contact items.

1. System connects to the server
2. System reads the contact details saved by the worker
3. System generates the XML
4. System returns the XML details to the client
5. System interprets the XML
6. System loads the records into the grid in the form

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 5 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-RI04-Read Deleted Items

Description:

The system reads the deleted items details received by the system for the particularized user. The deleted items details are sent to the client through XML and shown to the user.

Primary Actor: System

Stakeholders and Interests:

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

The system displays the requested item for the worker to view and read.

Main Success Scenarios:

System performs the following to read Deleted Items.

1. System connects to the server
2. System reads the deleted records particularized for the user

3. System generates the XML
4. System returns the XML details to the client
5. System interprets the XML
6. System loads the records into the grid in the form

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 6 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-RI05-Read Drafts

Description:

The system reads the drafted email messages received by the system for the particularized user. The draft email message details are sent to the client through XML and shown to the user.

Primary Actor: System

Stakeholders and Interests:

Enforcement Workers:

- The Enforcement Workers are interested in viewing and reading the inbox message items

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

System authenticates the worker successfully.

Main Success Scenarios:

System performs the following to read drafts items.

1. System connects to the server
2. System reads the drafts records

3. System generates the XML
4. System returns the XML details to client
5. System interprets the XML
6. System loads the records into the grid in the form

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 7 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-RI06-Read Sent Items

Description:

The system reads the sent items details received by the system for the particularized user. The sent items details are sent to the client through XML and shown to the user.

Primary Actor: System

Stakeholders and Interests:

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

The system displays the requested item for the worker to view and read.

Main Success Scenarios:

System performs the following to read Sent Items.

1. System connects to the server
2. System reads the sent items records particularized for the user

3. System generates the XML
4. System returns the XML details to the client
5. System interprets the XML
6. System loads the records into the grid in the form

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 8 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-NI00-Create New Items

Description:

The system accepts the new message details and transmits the message or saves the message based on the user action. A copy of the sent message is saved in the sent items folder when the message is transmitted and the message is saved in the drafts folder when the message is saved as draft. System accepts new contact details and creates the new contact details. System accepts new appointment details and creates the new appointment details.

Primary Actor: System

Stakeholders and Interests:

Enforcement Workers:

- The Enforcement Workers are interested in creating new email messages
- The Enforcement Workers are interested in creating new contacts
- The Enforcement Workers are interested in creating appointments
- The Enforcement Workers are interested in saving drafts

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

The system displays the requested item for the worker to view and read.

Main Success Scenarios:

System performs the following to create new message.

1. System creates the new message in accordance with UC-NI01 use case.

System performs the following to create new contact.

2. System creates the new contact in accordance with UC-NI02 use case.

System performs the following to create new appointments.

3. System creates the new appointments in accordance with UC-NI03 use case.

System performs the following to create new drafts.

4. System saves the drafts in accordance with UC-NI04 use case.

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 9 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-NI01-Create New Message

Description:

System accepts the new message details along with attachment details, collects the attachments and transmits the information to the server. The server transmits the message and a copy of the message is saved in the sent items folder of the user.

Primary Actor: System

Stakeholders and Interests:

Enforcement Workers:

- The Enforcement Workers are interested creating new email messages

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

System authenticates the worker successfully.

Main Success Scenarios:

System performs the following to create new email message.

1. System accepts the to email address
2. System accepts the cc email address
3. System accepts the email subject
4. System accepts the email message body
5. System accepts the attachments
6. System transforms the email message into XML document
7. System sends the XML document to the server
8. Server restores email message from the XML

System skips steps 9 and 10 when the user wants the email message be saved as a draft and performs step 11

9. System sends the email message to recipients
10. System saves the copy of the email message sent in user's sent items folder

System skips step 11 when sending the message to recipients

11. System saves the email message in drafts folder
12. System returns success/failure indicator back to client

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 10 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-NI02-Create New Contact

Description:

System accepts the new contact details and transmits the information to the server. The server saves the contact details in the contacts folder of the user.

Primary Actor: System

Stakeholders and Interests:

Enforcement Workers:

- The Enforcement Workers are interested creating new contacts

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

System authenticates the worker successfully.

Main Success Scenarios:

System performs the following to create new contacts.

1. System accepts the contact name, address, phone numbers, email address, website address, office details, secretary details, spouse details, date of birth and anniversary dates.
2. System transforms the contact details into XML document
3. System sends the XML document to the server

4. Server restores the contact details from the XML
5. System saves the contact details in user's contacts folder
6. System returns success/failure indicator back to client

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 11 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-NI03-Create New Appointments

Description:

System accepts the new appointment details and transmits the information to the server. The server saves the appointment details in the appointments folder of the user.

Primary Actor: System

Stakeholders and Interests:

Enforcement Workers:

- The Enforcement Workers are interested creating new appointments

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

System authenticates the worker successfully.

Main Success Scenarios:

System performs the following to create new appointments.

1. System accepts the appointment details such as participants, subject, location, date and time and description of the appointment.
2. System transforms the appointment details into XML document
3. System sends the XML document to the server

4. Server restores the appointment details from the XML
5. System saves the appointment details in user's calendar folder
6. System returns success/failure indicator back to client

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 12 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-NI04-Create New Drafts

Description:

System accepts the new message draft details along with the attachment details and transmits the information to the server. The server saves the message in the drafts folder of the user.

Primary Actor: System

Stakeholders and Interests:

Enforcement Workers:

- The Enforcement Workers are interested creating new email message draft

Frequency: Daily

Preconditions: The enforcement worker must be authorized to log into the system to access the emails. The eManager client must be installed on the network client. The client must be in the same network as the eManager server.

Post Condition:

System authenticates the worker successfully.

Main Success Scenarios:

System performs the following to create new drafts.

1. System accepts the new message draft details.
2. System transforms the message details into XML document
3. System sends the XML document to the server
4. Server restores the message details from the XML
5. System saves the message details in user's drafts folder
6. System returns success/failure indicator back to client

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 13 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Use Case: UC-MT00-Create New Alert

Description:

The eManager client process registers the client with the server. The server notifies the client of arrival of the new email messages.

Primary Actor: System

Stakeholders and Interests:

Enforcement Workers:

- The Enforcement Workers receive alerts of the arrival of the new email messages and new appointment requests from users and/or other stakeholders

Frequency: Daily

Preconditions: The worker is authenticated by the system to access the emails and is currently logged on the eManager system.

Post Condition:

The system displays the new email message and/or the appointment requests from other stakeholders.

Main Success Scenarios:

1. System registers the client with the server by passing the clients address.

   System continues to perform steps 3 and 4

2. The Server inspects the users email message folder continually for new email messages or new appointment requests.

3. System validates if the client is still active

   System performs step 5 when the client goes inactive

4. The Server alerts the client of the new item found in users folder

5. System terminates the client reference

Extensions:

Special Requirements: None

Technology and Data Variations List: None

Open Issues:

Use Case Diagram:

(Please refer to Figure 14 in Appendix C)

Last Updated by: Solomon Vedaprakash

Date Last Update: **October 08, 2005 12:14:21**

Appendix B: CRC Cards

CRC Cards

Enforcement Worker	
USER	
01. Logon to eManager 02. Read Inbox 03. Read Drafts 04. Read Outbox 05. Read Contacts 06. Read Calendar 07. Read Sent Items 08. Read Attachments 09. Read Appointments 10. Read Deleted Items 11. Create New Message 12. Save Drafts 13. Add Attachments 14. Create New Contact 15. Create New Appointment 16. Delete Inbox 17. Delete Drafts 18. Delete Outbox 19. Delete Contacts 20. Delete Sent Items 21. Delete Attachments 22. Delete Appointments 23. Delete Deleted Items 24. Update Drafts 25. Update Contacts 26. Update Attachments 27. Update Appointments 28. Move Inbox Item 29. Move Drafts Item 30. Move Sent Items Item 31. Move Deleted Items Item	01. Appointment 02. Attachment 03. Contact 04. Message

eManagerSystem	
01. Validate User 02. Validate Application Call 03. Terminate Process 04. Register Client 05. Process Server Call 06. Alert User 07. Release Server Registration 08. Serve Server Events	01. User

Appointment	
01. Read Appointment 02. Create New Appointment 03. Save Appointment 04. Delete Appointment 05. Update Appointment 06. Move Appointment 07. Send Appointment Request 08. Generate XML	01. User 02. Contact 03. XML

Message	
01. Read Message 02. Create New Message 03. Save Message 04. Delete Message 05. Update Message 06. Move Message 07. Send Message 08. Generate XML	01. User 02. XML

Contact	
01. Read Contact 02. Create New Contact 03. Save Contact 04. Delete Contact 05. Update Contact 06. Move Contact 07. Generate XML	01. User 02. Appointment 03. XML

XML	
01. Read Appointment 02. Create New Appointment 03. Save Appointment 04. Delete Appointment 05. Update Appointment 06. Move Appointment 07. Send Appointment Request 08. Generate XML	01. User 02. Appointment 03. Message 04. Contact

Appendix C: List of Figures

Figure 1: eManager UC1 [Use Case]

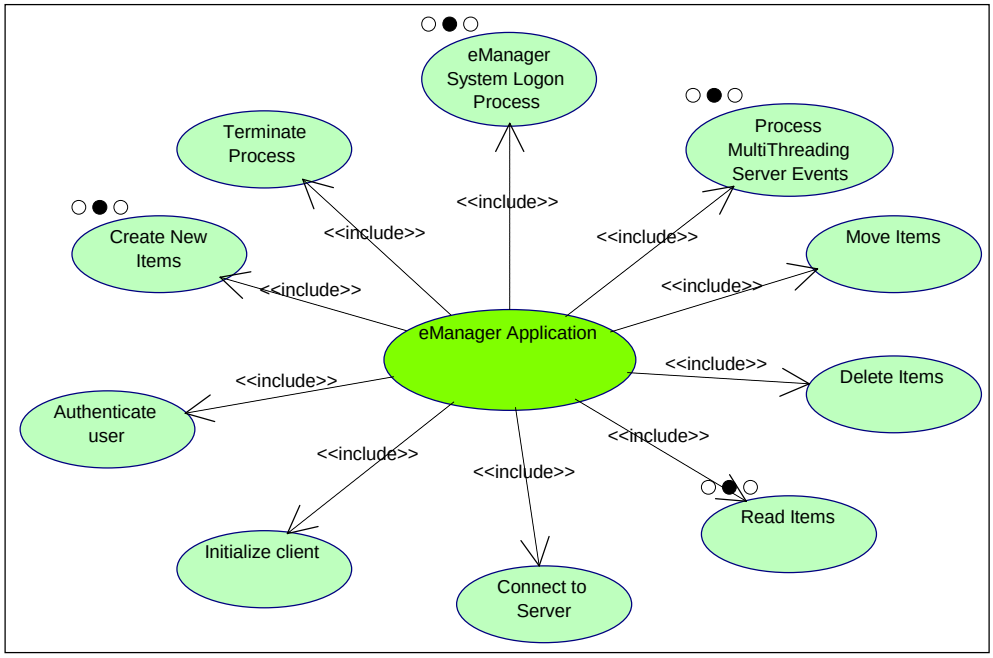


Figure 2: Read Items UC-RI00 [Use Case]

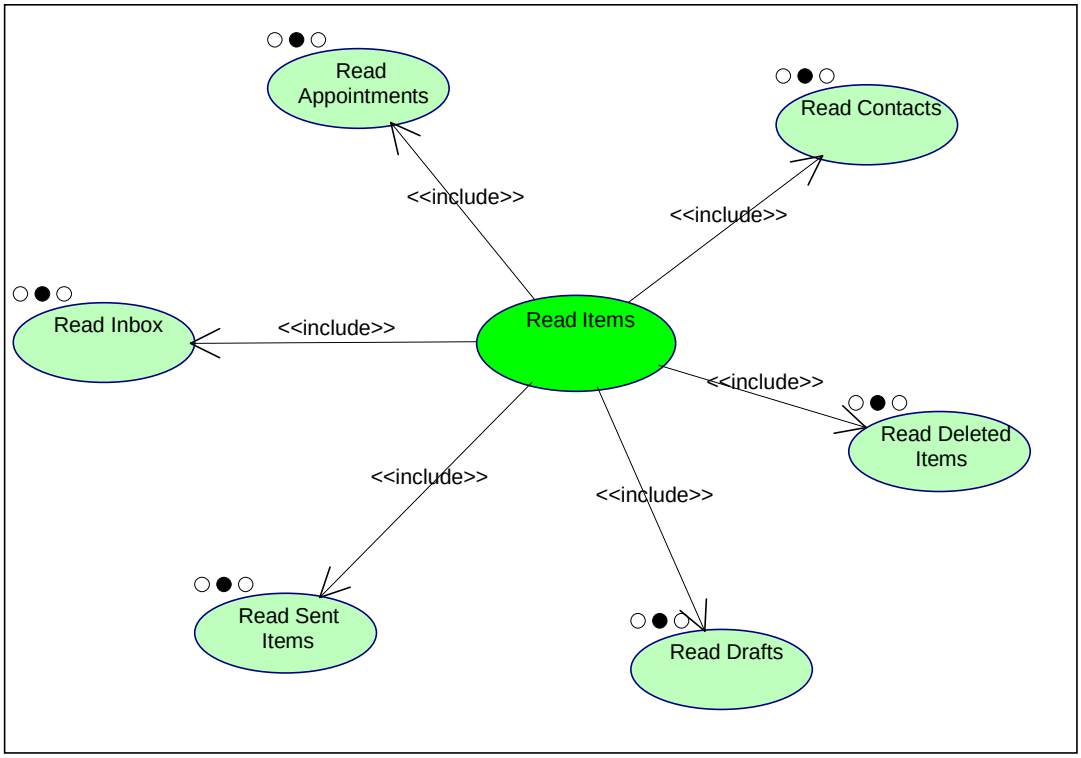


Figure 3: Read Items UC-RI01 [Use Case]

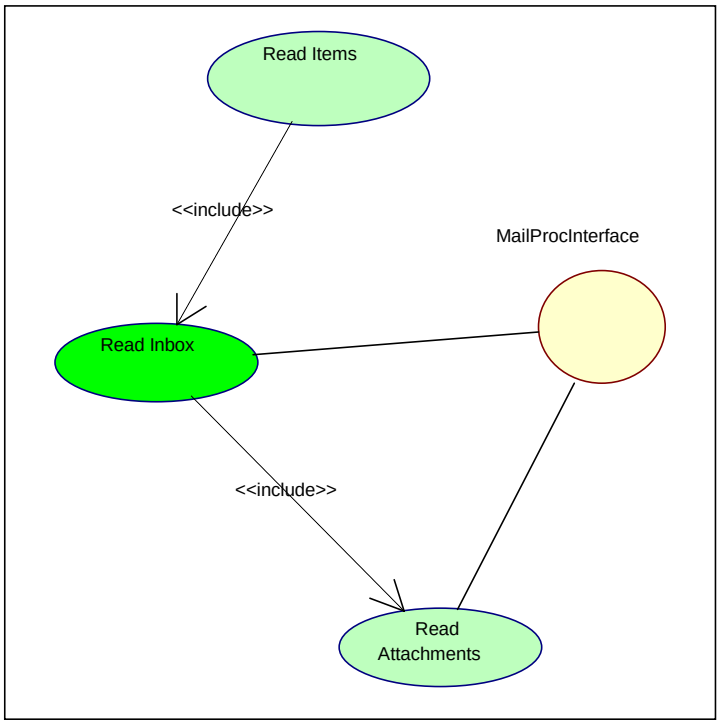


Figure 4: Read Items UC-RI02 [Use Case]

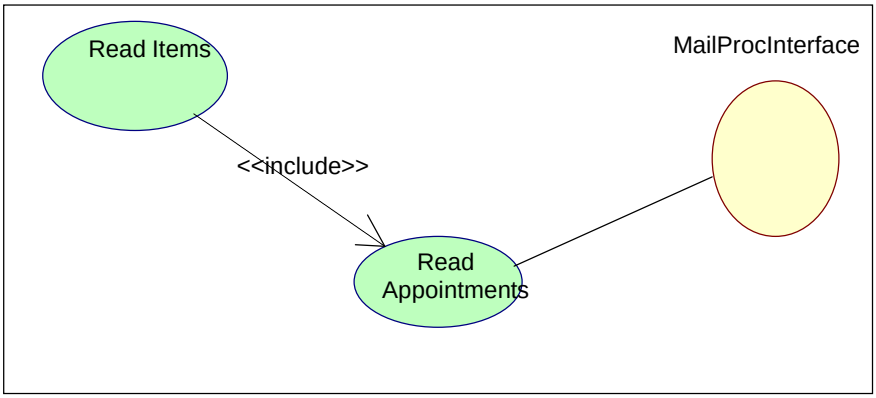


Figure 5: Read Items UC-RI03 [Use Case]

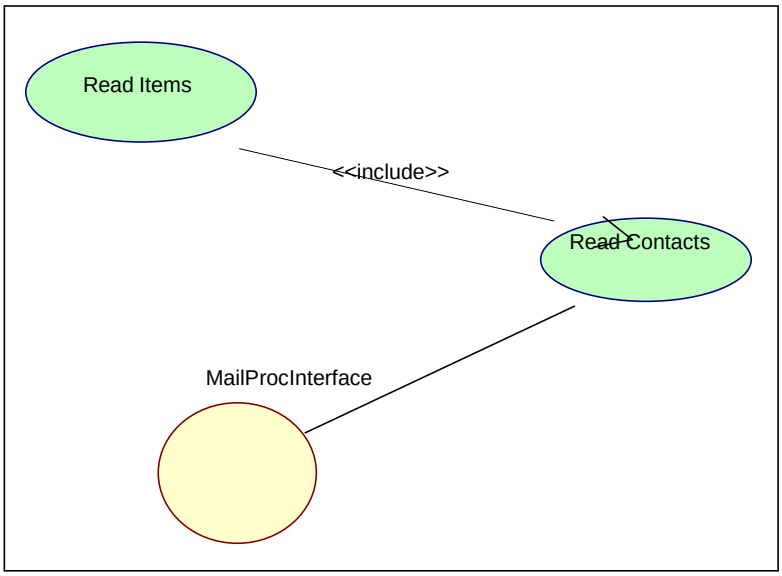


Figure 6: Read Items UC-RI04 [Use Case]

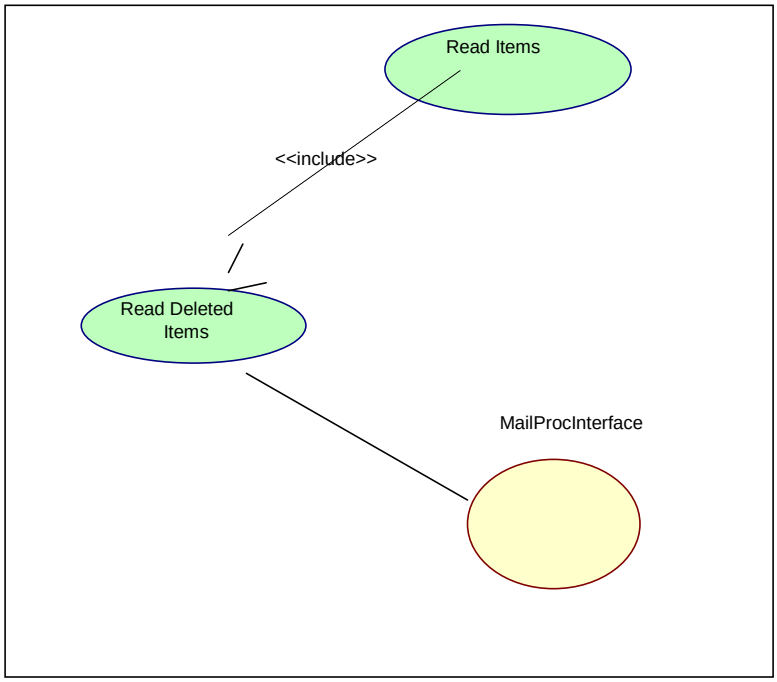


Figure 7: Read Items UC-RI05 [Use Case]

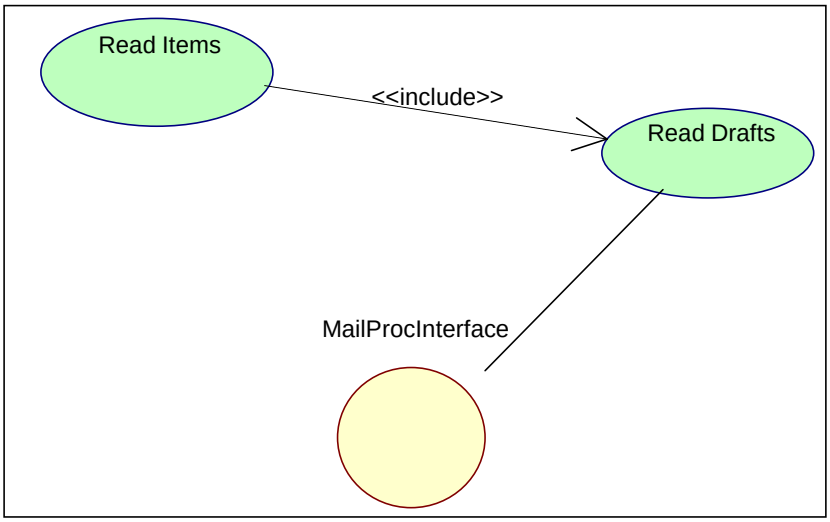


Figure 8: Read Items UC-RI06 [Use Case]

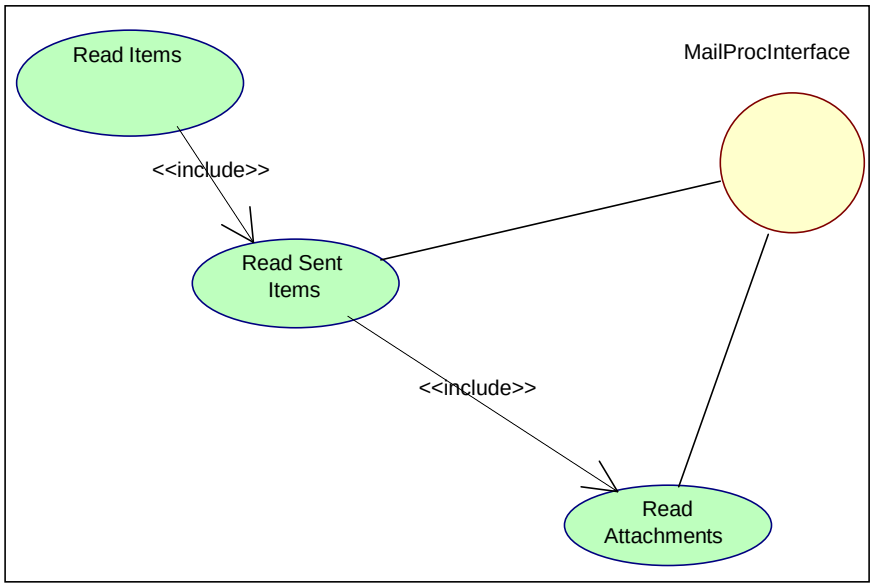


Figure 9: CrNew Items UC-NI00 [Use Case]

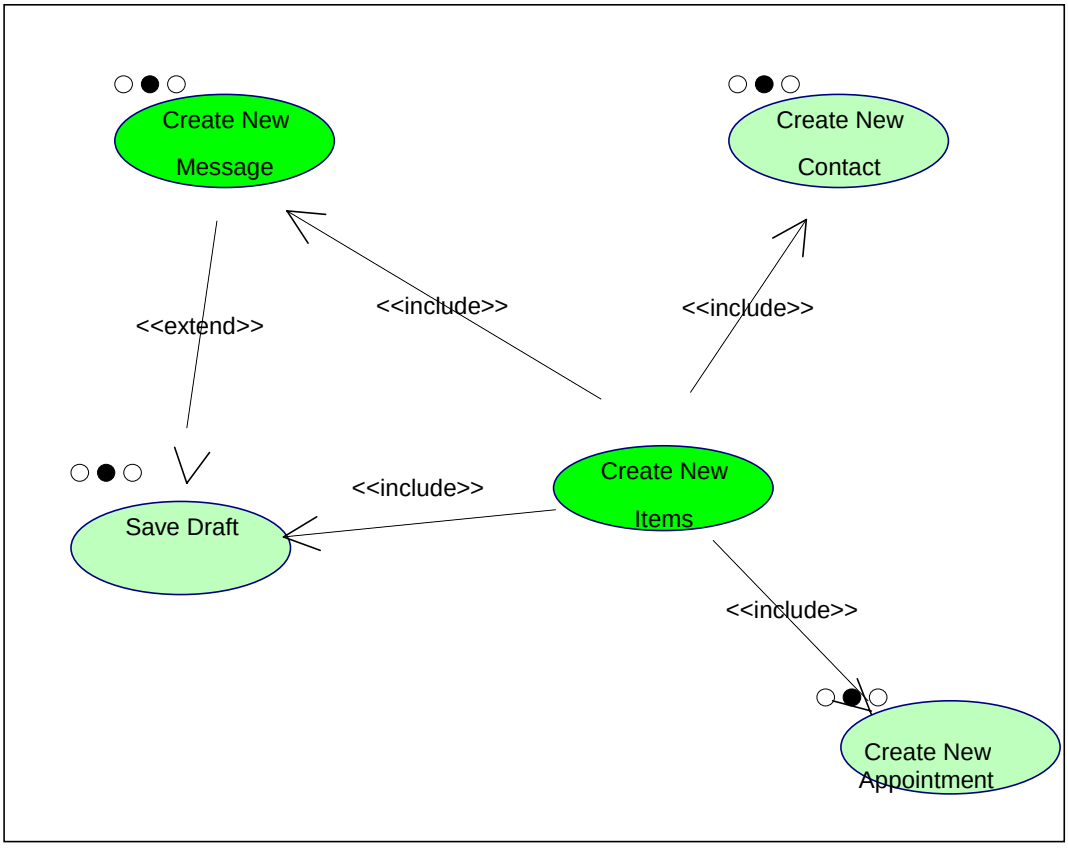


Figure 10: CrNew Items UC-NI01 [Use Case]

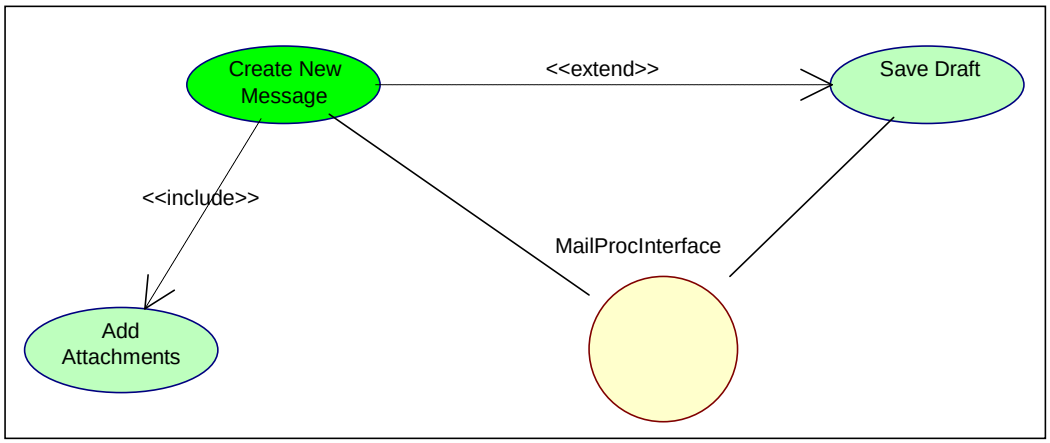


Figure 11: CrNew Items UC-NI02 [Use Case]

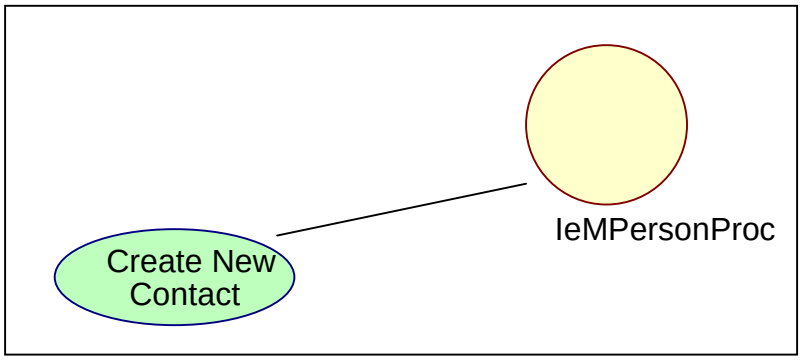


Figure 12: CrNew Items UC-NI03 [Use Case]

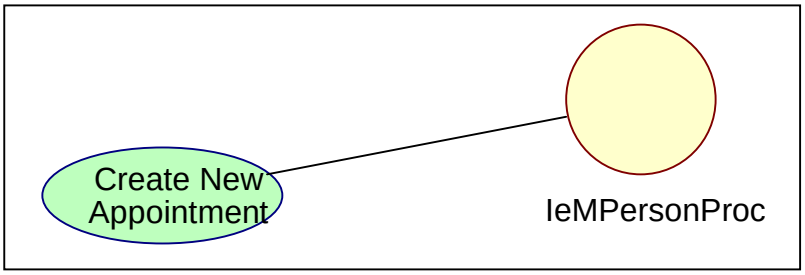


Figure 13: CrNew Items UC-NI04 [Use Case]

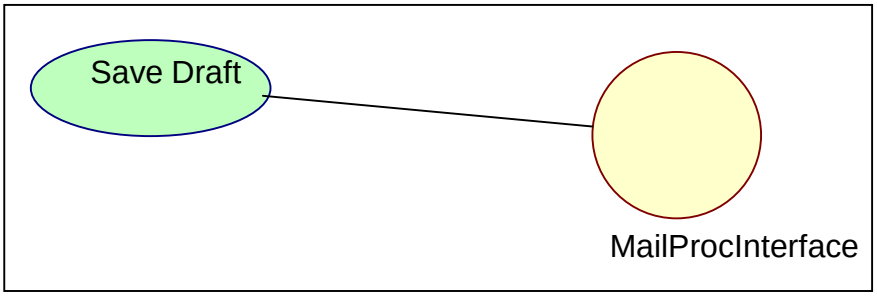


Figure 14: MtClient Process UC-MT00 [Use Case]

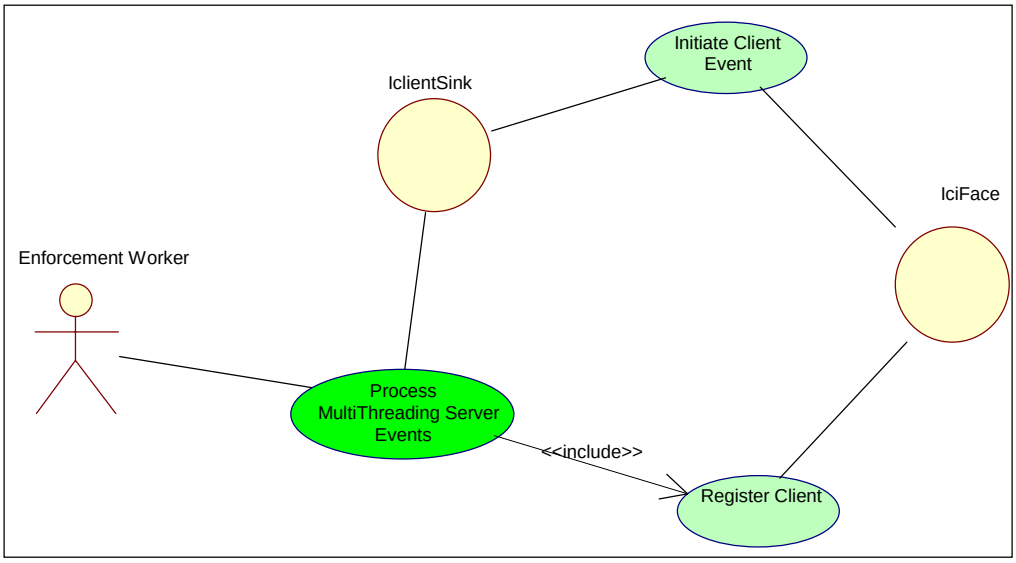


Figure 15: eManager Context Diagram [Use Case]

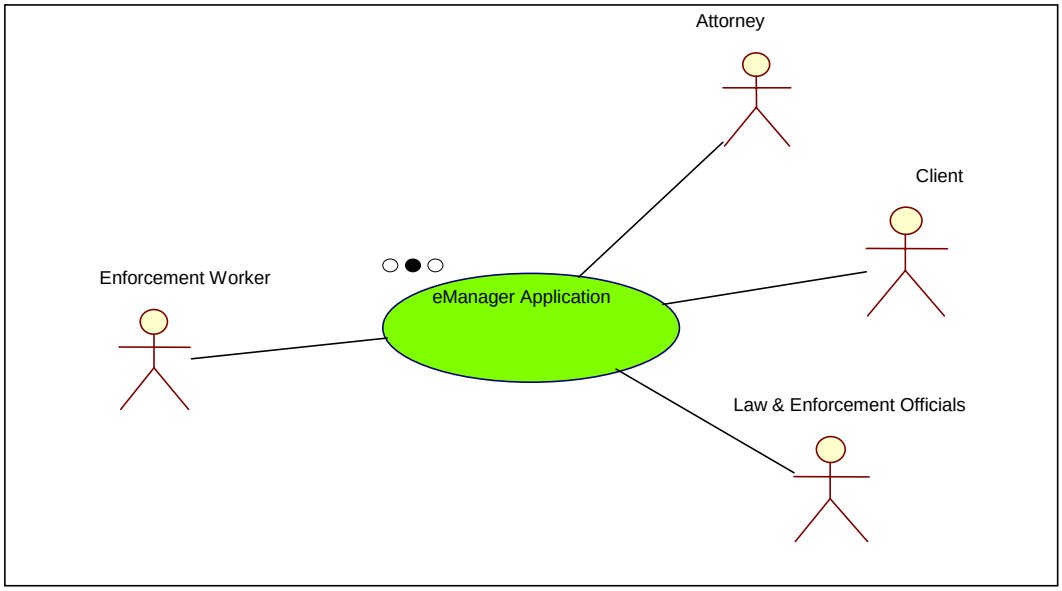


Figure 16: Application accessing Directory Service using LDAP wire protocol [CMN00]

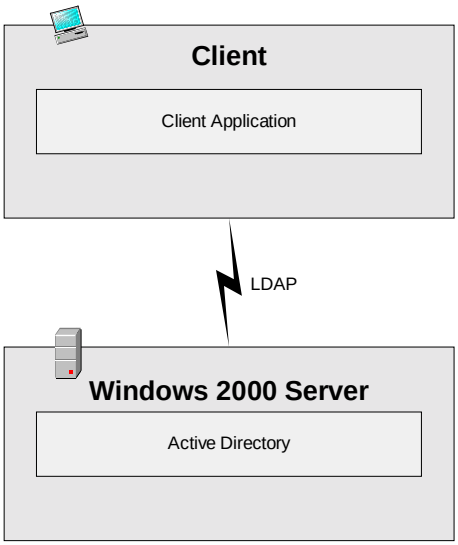


Figure 17: The relationship of ADSI and ADO to Active Directory [CMN00]

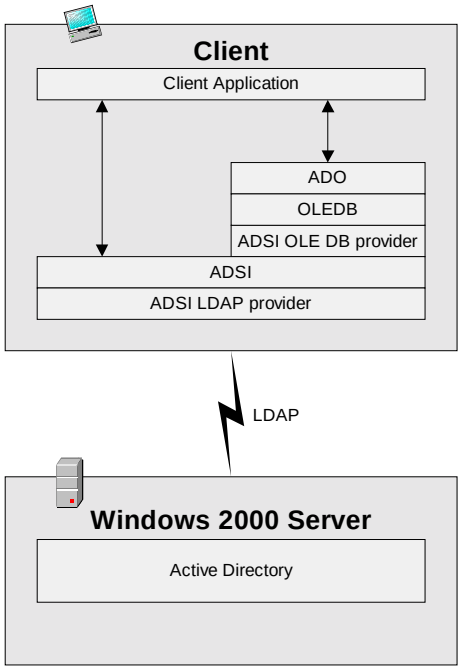


Figure 18: CDO to access Active Directory [CMN00]

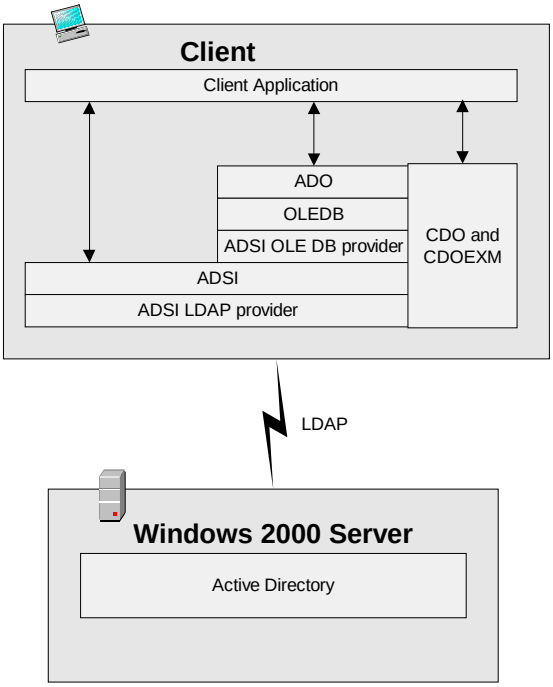


Figure 19: Proxy / Stub Overview [JSW00]

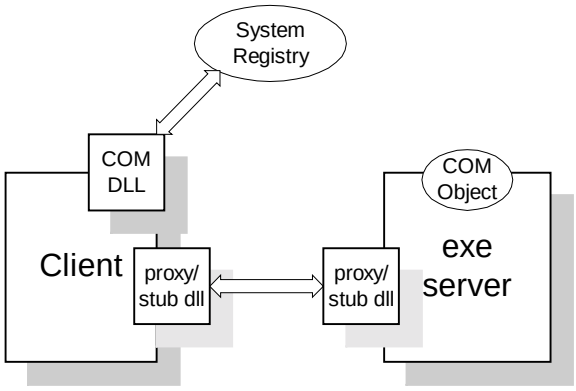


Figure 20: Connection Points and Sinks [JSW00]

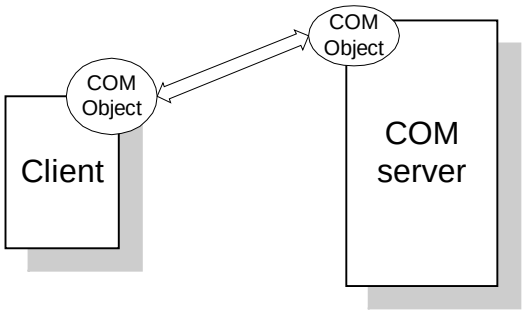


Figure 21: eManager Application Architecture

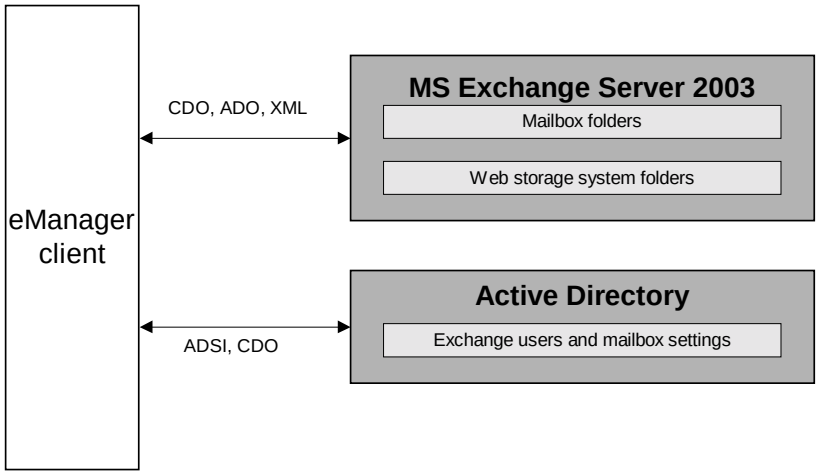


Figure 22: eManager Contacts Tab Control Dialog

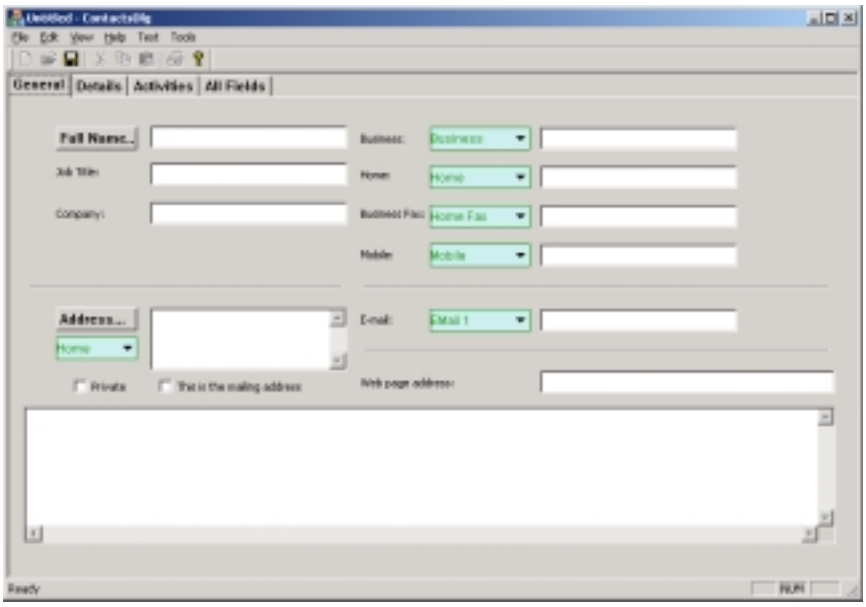


Figure 23: Custom ComboBox Control Class Diagram

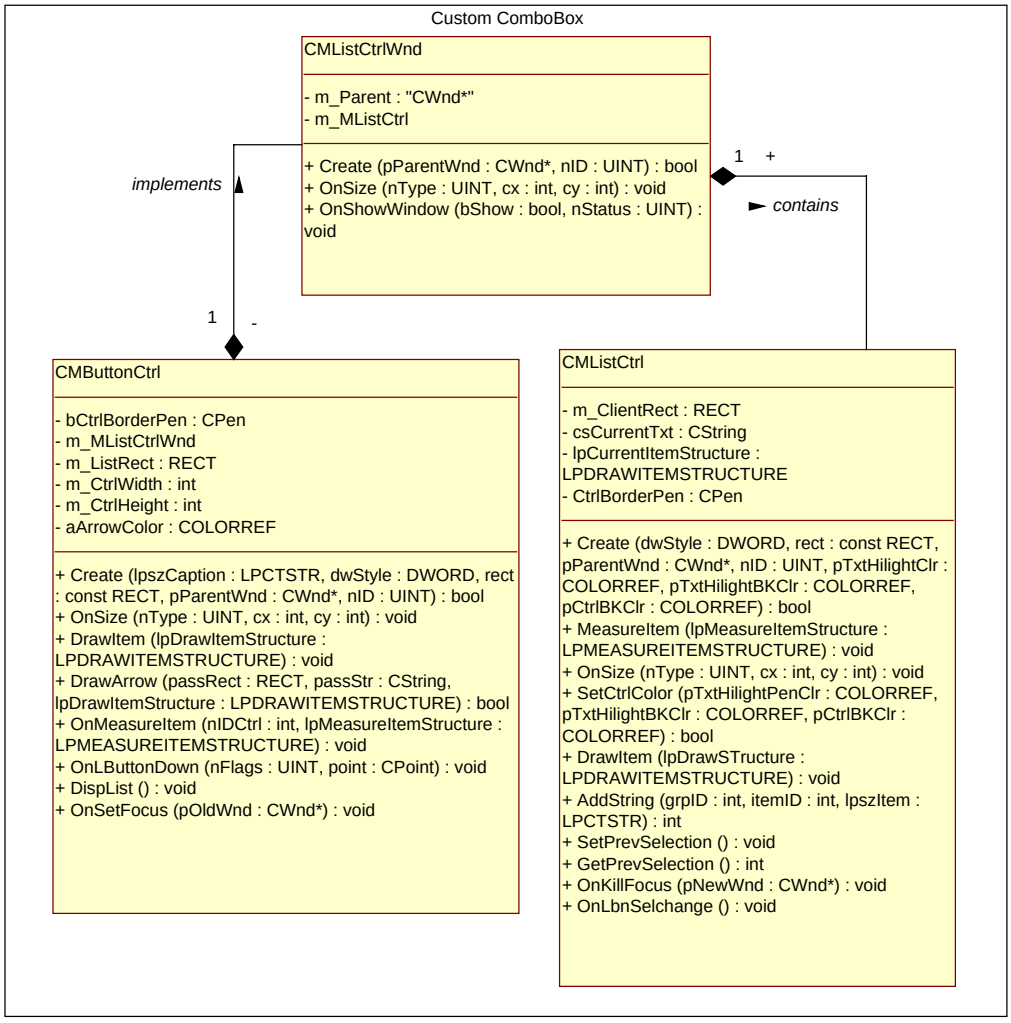


Figure 24: eManager Contacts window with custom ComboBox shows the Business 2 item selected

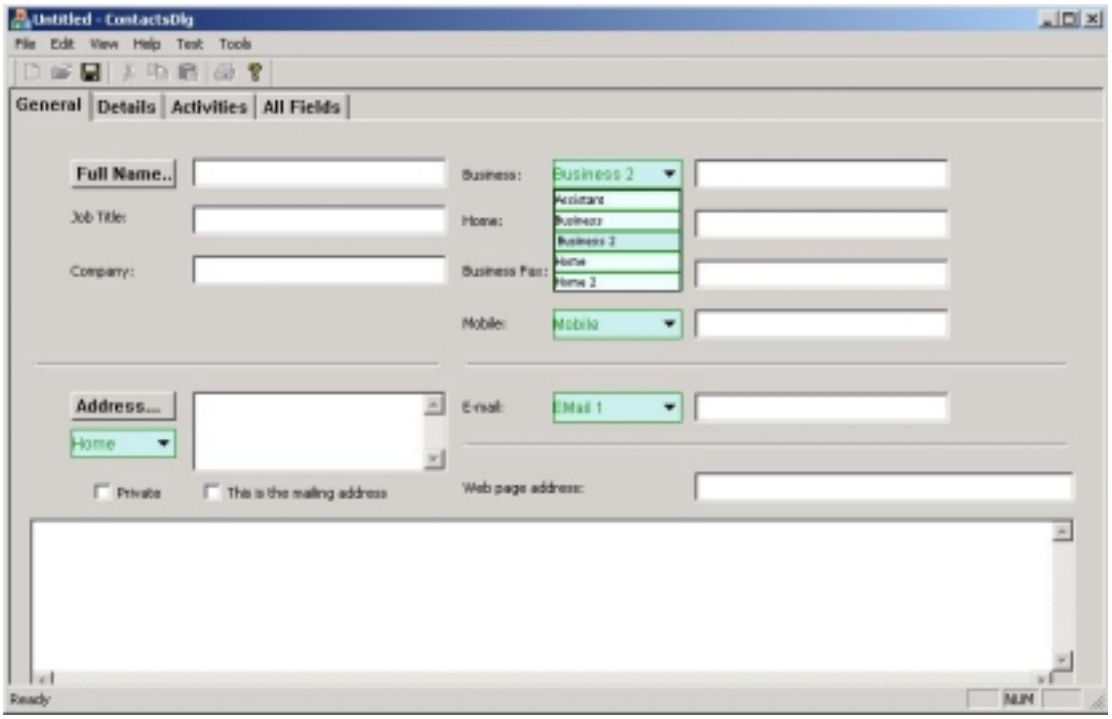


Figure 25: eManager application attachments processing using XML document and Stream object

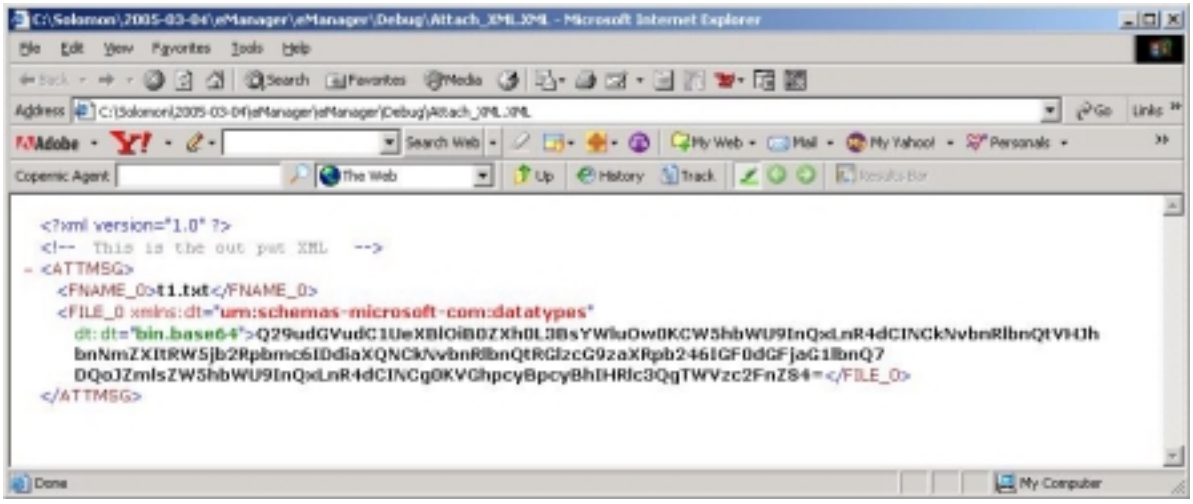


Figure 26: eManager Data Communications through XML

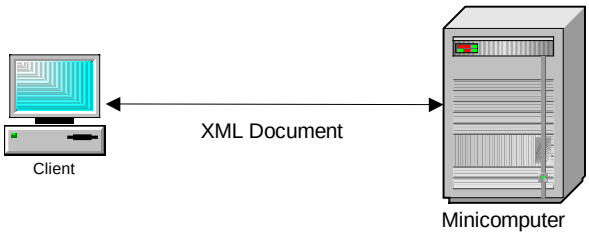


Figure 27: eManager application class diagram

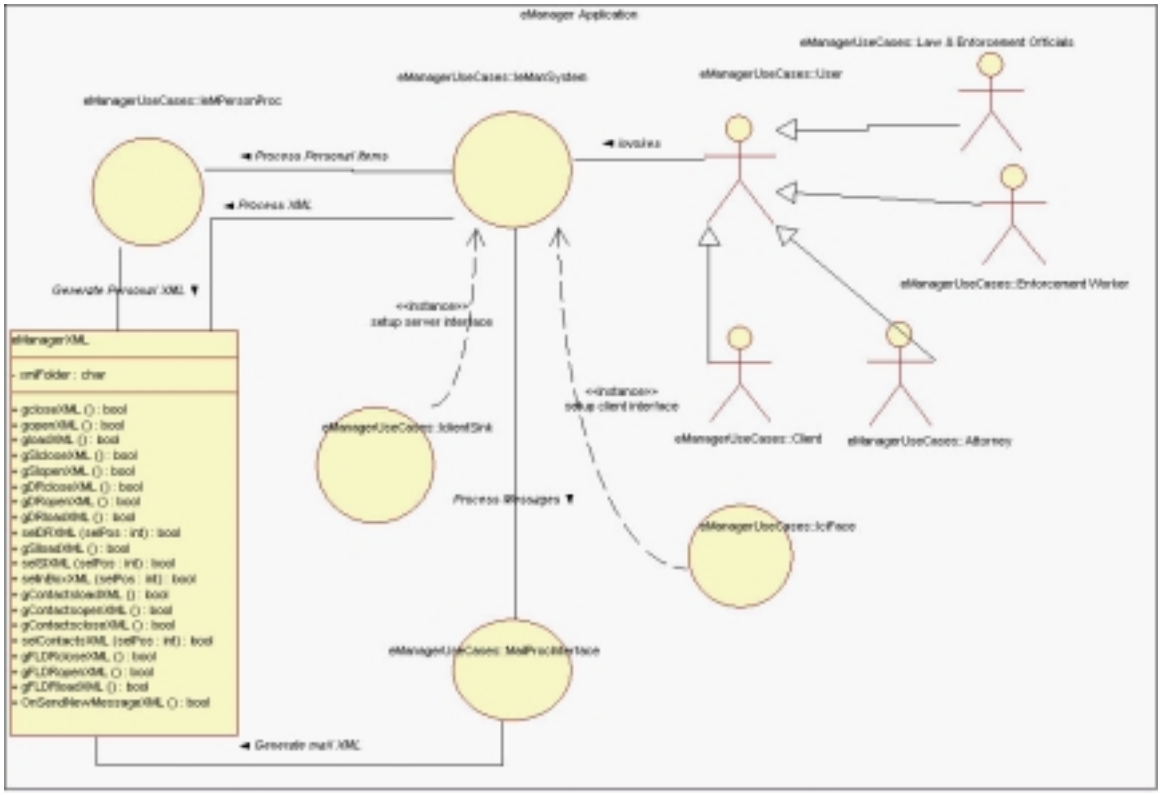


Figure 28: eManager application Logon/Server Event

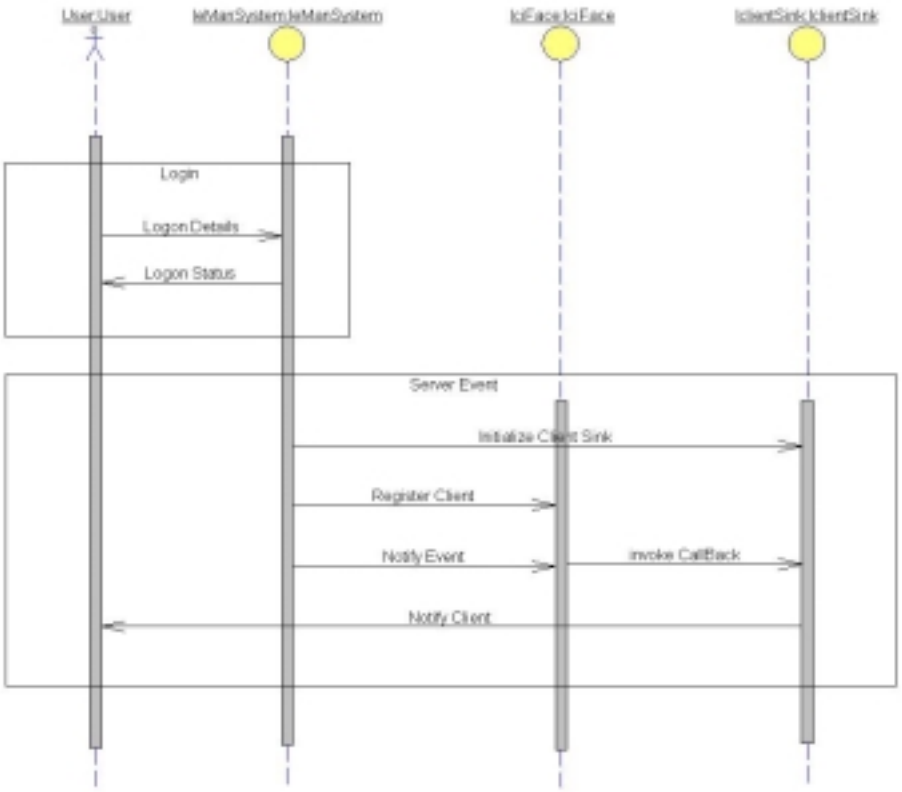


Figure 29: eManager application Send Mail

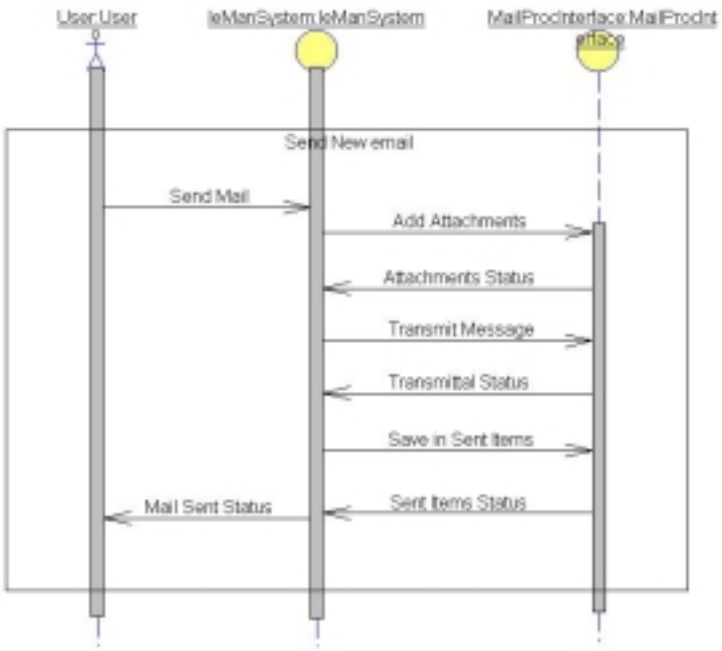


Figure 30: eManager application Read Items

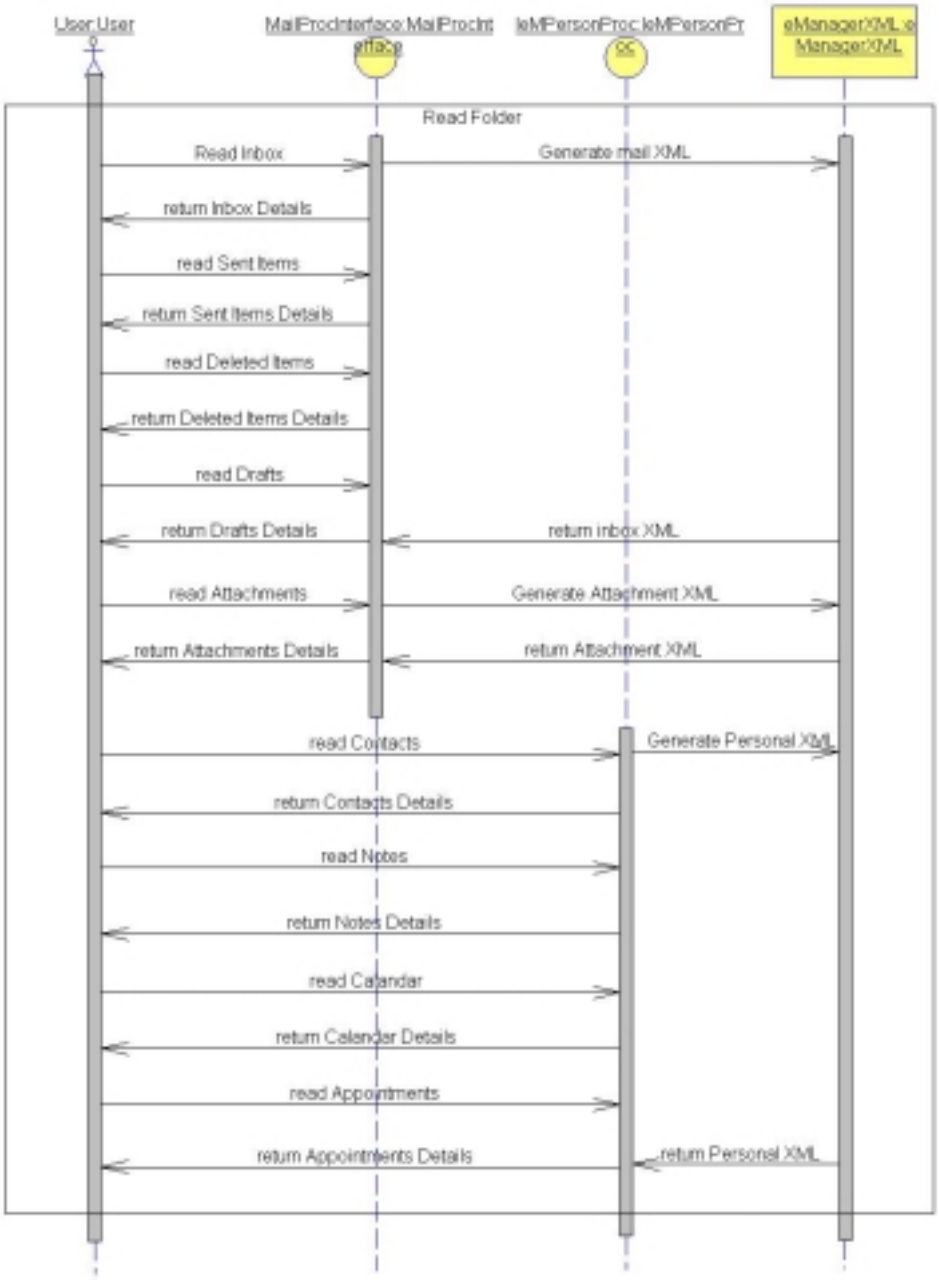


Figure 31: eManager application Save Draft

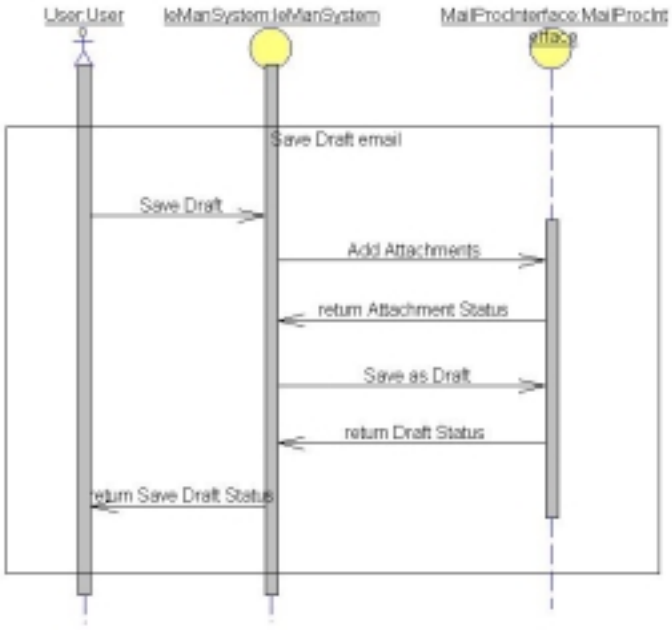


Figure 32: eManager application Move Item

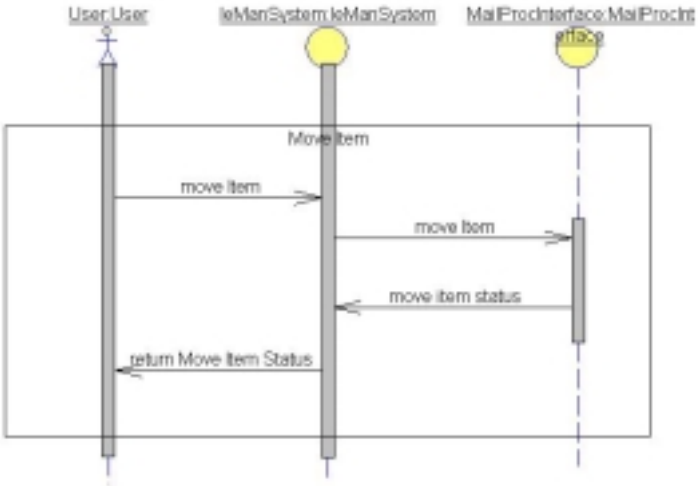


Figure 33: eManager application Delete Item

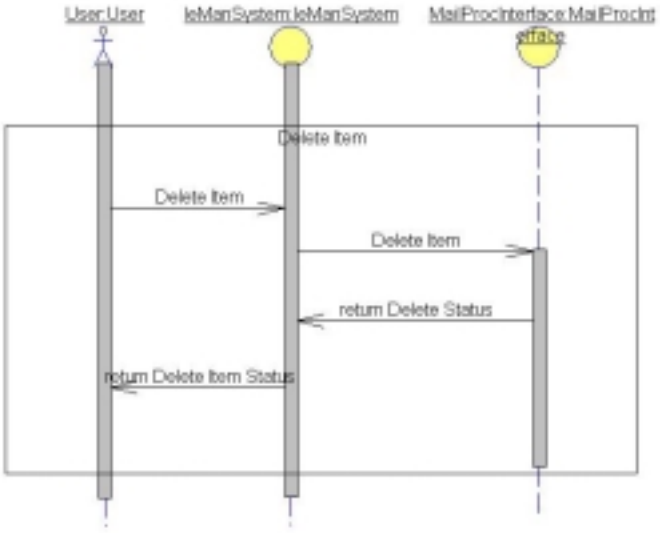


Figure 34: eManager application Create New Message State Diagram

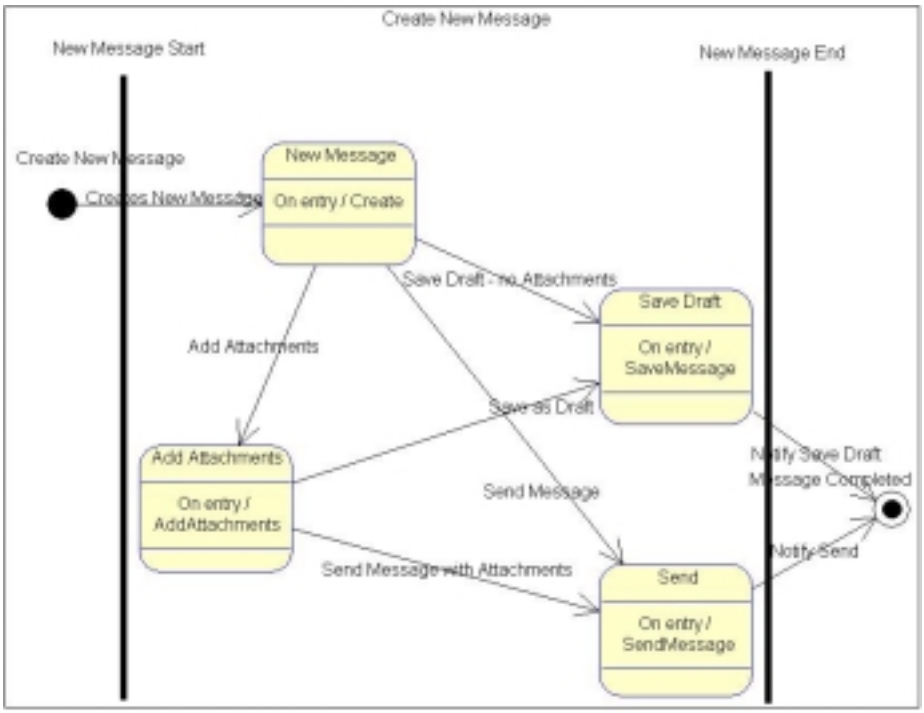


Figure 35: eManager application Component Diagram

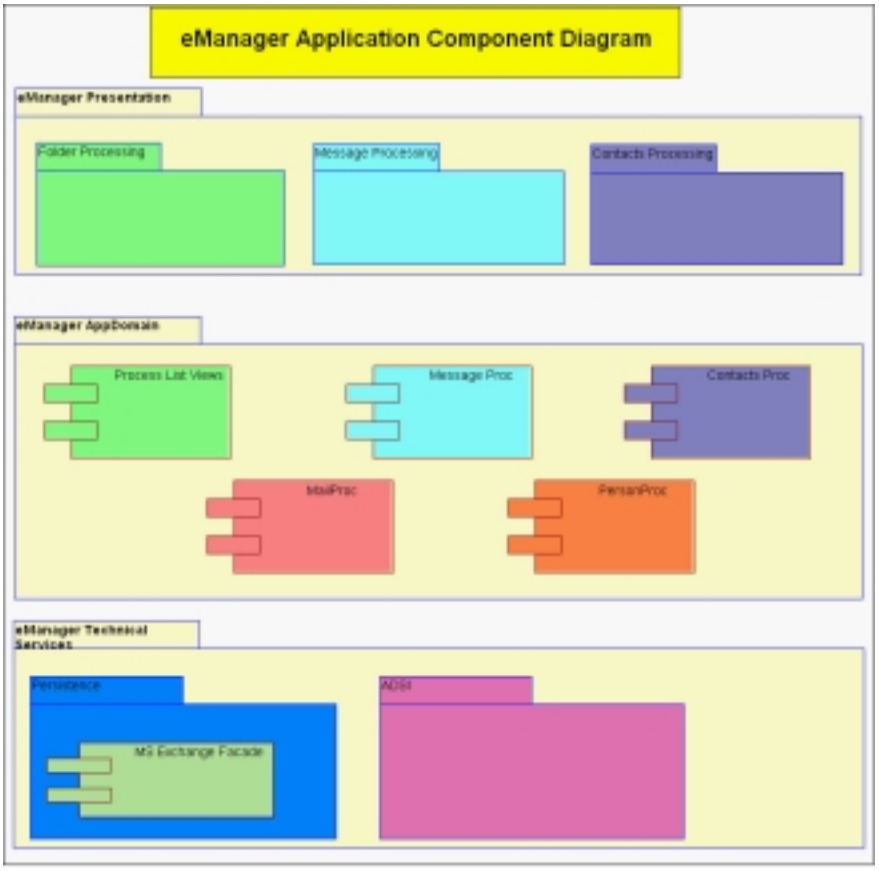
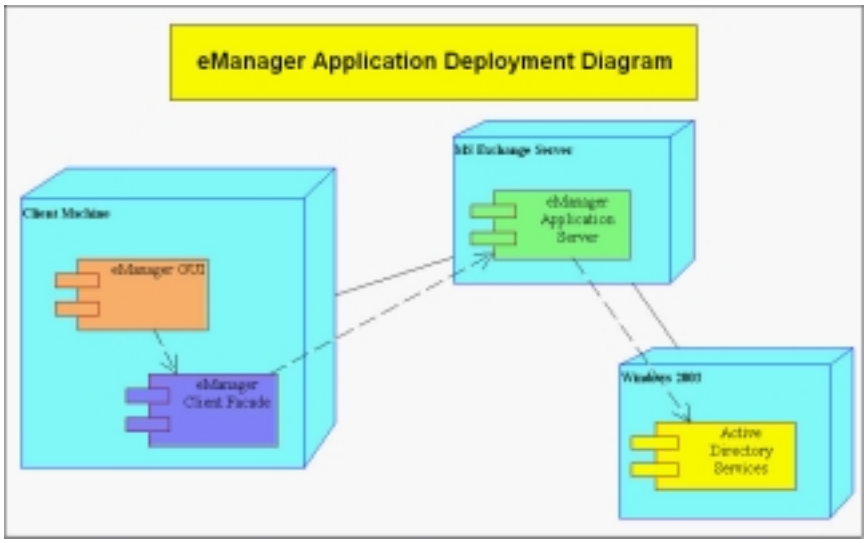


Figure 36: eManager application Deployment Diagram



References

[BDN96] Baudoin, Claude & Hollowell, Glenn. “Realizing the Object-Oriented Lifecycle”, Upper Saddle River, NJ: Prentice Hall, 1996.

[CAC] Cockburn, Alistair, “Using CRC Cards”, <http://alistair.cockburn.us/crystal/articles/ucrcc/usingcrccards.html> (May 18, 2005 20:45)

[CMN00] Cindy Martin, “Programming Collaborative Web Applications with Microsoft Exchange Server 2000”

[DWCA] Andrew W Troelsen, “Developer’s Workshop to COM and ATL 3.0”

[FRU00] “Functional Requirements and Use Cases”, Bredemeyer Consulting (1999–2000), <http://www.bredemeyer.com/use_cases.htm> (May 20, 2005 16:45)

[JMS02] Kris Jamsa, “C/C++/C# Programmer’s Bible – The Ultimate Guide to C/C++/C# Programming” – Second Edition

[JSW00] John E. Swanke, “COM Programming by Example Using MFC, ActiveX, ATL, ADO and COM+”

[KBWC] Kent Back, Ward Cunningham, “A Laboratory For Teaching Object Oriented Thinking”, October 1989, <http://c2.com/doc/oopsla89/paper.html> (May 19, 2005 16:35)

[MSD05] “Collaborative Applications”, Microsoft Corporation, 2005, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wss/wss/_esdk_intro_coll_apps.asp >(June 16, 2005 21:33)

[MSD052] “Using Active Directory Service Interfaces”, Microsoft Corporation, 2005, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/adsi/adsi/using_adsi.asp>(June 18, 2005 21:44)

[MSD053] “RegisterWindowMessage Function”, Microsoft Corporation, 2005, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/messagesandmessagequeues/messagesandmessagequeuesreference/messagesandmessagequeuesfunctions/registerwindowmessage.asp >(June 18, 2005 23:33)

[MSDN05] “IDataSource Methods”, Microsoft Corporation, 2005, <http://msdn.microsoft.com/library/default.asp?url=/library/en-

us/cdosys/html/9d3b50c4-fcab-4455-b6d9-2acd71d51517.asp >(May 6, 2005 20:22)

[OOA96] Brummond, Nils, “Object Oriented Analysis and Design using CRC Cards”, 1996, <http://www.csc.calpoly.edu/~dbutler/tutorials/winter96/crc_b/>(May 16, 2005 10:05)

[PBS97] “Performance based supportability analysis: A success story”, Logistics Spectrum, May/Jun1997, <http://www.findarticles.com/p/articles/mi_qa3766/is_199705/ai_n8776821 >(May 15, 2005 9:05)

[PLY00] Perdita Stevens, Rob Pooley, “Using UML Software engineering with objects and components”

[REL01] “Developing Good Reliability Specifications”, ReliaSoft 2001, <http://www.weibull.com/hotwire/issue3/relbasics3.htm >(May 15, 2005 8:45)

[RMD] Rubin, M David, ”Introduction to CRC Cards”, SoftStar-Inc, 1994-2002, <http://www.softstar-inc.com/Methodology/CRCIntro.htm> (May 18, 2005 21:01)

[SADI] “Sample Application Design and Implementation”, <http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/sample-app/sample-app1.3.1.html> (May 21, 2005 10:44

[SCH05] Stephen R. Schach, “Object-Oriented & Classical Software Engineering”, Sixth Edition

[SDR05] “How to Draw UML Diagrams”, SmartDraw 2005, <http://www.smartdraw.com/tutorials/software-uml/uml5.htm> (May 19, 2005 17:22)

[SER01] “Cost-Effective User Centered Design”, Serco Ltd (2001) <http://www.usability.serco.com/trump/methods/recommended/requirements.htm >(May15, 2005 8:15)

[WBW90] R. Wirfx-Brock, B. Wilkerson, and L. Wiener, “Designing Object-Oriented Software”, Prentice-Hall, Englewood Cliffs, NJ, 1990. (Chapters 1, 12, and 13)

[WMH100] Daniel Kopitchinski, “Windows Message Handling – Part 1”, 2000 <http://www.codeproject.com/dialog/messagehandling.asp>(June 16, 2005, 17:45)

[WMH200] Daniel Kopitchinski, “Windows Message Handling – Part 2”, 2000 <http://www.codeproject.com/dialog/messagehandling2.asp>(June 16, 2005, 17:45)

[XYL01] XiangYang Liu, “Early-binding to a non-existent COM object”, 2001, http://www.codeproject.com/com/comearlybind.asp (May 20, 2005 15: 30)

Glossary

AD Microsoft Active Directory is the foundation for distributed networks built on Windows 2000 Server and Windows Server 2003 operating systems that use domain controllers. Active Directory provides secure, structured, hierarchical data storage for objects in a network such as users, computers, printers, and services. Active Directory provides support for locating and working with these objects.

ADO Microsoft® ActiveX® Data Objects (ADO) enables client applications to access and manipulate data from a variety of sources through an OLE DB provider. Its primary benefits are ease of use, high speed, low memory overhead, and a small disk footprint. ADO supports key features for building client/server and Web-based applications.

ADSI Active Directory Service Interfaces (ADSI) is a COM-based interface that supports multiple directories and multiple languages

ATL Active Template Library is a library of templates to support Visual C++ developers who must create lightweight COM / Component Object Model components. ATL / Active Template Library has pre-written boilerplate code which can be use to make programmers job easier, faster, more efficient and much less tedious.

CDO Collaboration Data Objects library allows you to access the Global Address List and other server objects, in addition to the contents of mailboxes and public folders.

COM Common Object Module is an interface-programming paradigm. Interfaces never provide an implementation and never define state. An interface only describes what can be done. The supporting class defines how it is accomplished.

CRC CRC stands for "Class-Responsibility-Collaborator". It names a brainstorming technique that works with scenario walkthroughs to stress test a design. It also supports a rapid and thorough exploration of design alternatives. It may be used during initial model construction as a brainstorming technique, and again later to evaluate the design. It may be done by one person, or up to 5 people, after which it needs careful facilitation.

DLL DLL stands for Dynamic Link Library. A DLL is a special type of executable file, which can only be called from within another program. DLL's are often produced to modularize programs. DLLs are used extensively where the functions within those DLLs are shared by more than one process. On UNIX platforms DLL's are normally referred to as Libraries.

DNS The Domain Name Server System is a global network of servers that translate host names like www.somesite.com into dotted numerical IP (Internet Protocol) addresses, like 144.268.89.76

ExOLEDB

Microsoft Exchange 2000 Server provides a new, high-performance OLE DB provider that can be used on the local server to access Exchange store items: the Exchange OLE DB (ExOLEDB) provider. Through the ExOLEDB provider, programmers can access the Exchange store using OLE DB, Microsoft ActiveX Data Objects (ADO), and Collaboration Data Objects (CDO).

GUI Graphical User Interface lets users to interact with computer using pictures and symbols in addition to entering typed text. Examples of GUIs include Mac OS, Mac OS X, Microsoft Windows, and the X Window System. The

purpose of a GUI is to make a computer easier to use. Rather than having to memorize many complicated commands and type them precisely. The user may point and click with an input device, usually a mouse, to run programs or manipulate files.

IIS Internet Information Services (IIS) is a powerful Web server that provides a highly reliable, manageable, and scalable Web application infrastructure for all versions of Windows Server 2003. IIS helps organizations increase Web site and application availability while lowering system administration costs. IIS supports the Microsoft Dynamic Systems Initiative (DSI) with automated health monitoring, process isolation, and improved management capabilities.

LDAP Lightweight Directory Access Protocol is a client-server protocol for accessing a directory service. It was initially used as a front-end to X.500, but can also be used with stand-alone and other kinds of directory servers. LDAP lets user to locate organizations, individuals, and other resources such as files and devices in a network, whether on the Internet or on a corporate intranet, and whether or not you know the domain name, IP address, or geographic whereabouts are known. An LDAP directory can be distributed among many servers on a network, then replicated and synchronized regularly. An LDAP server is also known as a Directory System Agent (DSA).

MFC MFC is the Microsoft Foundation Classes. It's a wrapper to the Windows API, which allows user to create windowed (GUI) programs under MS Windows.

MTTF Mean Time to Fail (Software Measure) is a basic measure of reliability for non-repairable systems. It is the mean time expected until the first failure of a piece of equipment. MTTF is a statistical value and is meant to be the mean over a long period of time and large number of units. For constant failure rate systems, MTTF is the inverse of the failure rate. If failure rate is in failures/million hours, MTTF = 1,000,000 / Failure Rate for components with exponential distributions.

MTBF Mean Time Between Failures is a basic measure of reliability for repairable items. It can be described as the number of hours that pass before a component, assembly, or system fails. It is a commonly used variable in reliability and maintainability analyses.

UP Unified Process is a software engineering process, aimed at guiding software development organizations in their endeavors.

URL Uniform Resource Locators help access the web storage system efficiently.

XML EXtensible Markup Language was designed to describe data and to focus on what data is.