DATABASE DESIGN DEVELOPEMNT: i-LINK – AN INTEGRATED MESSAGING FRAMEWORK

By

Pamela Michel Chahine

B.ENG in Civil and Environmental Engineering
McGill University, 2002

Submitted to the Department of Civil and Environmental Engineering
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Civil and Environmental Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2004

Signature of Author.....
Department of Civil and Environmental Engineering
May 7, 2004

Certified by .....
Dr. George A. Kocur
Senior Lecturer, Department of Civil and Environmental Engineering
Thesis Supervisor

Accepted by .....
Heidi Nepf
Chairman, Departmental Committee on Graduate Studies

DATABASE DESIGN DEVELOPEMNT: i-LINK – AN INTEGRATED MESSAGING FRAMEWORK

By
Pamela Michel Chahine

Submitted to the Department of Civil and Environmental Engineering
on May 7, 2004 in Partial Fulfillment of the
Requirements for the Degree of Master of Engineering in
Civil and Environmental Engineering

Abstract

For many people, online communication has meant a plethora of communication media – emails, instant messages, blogs, etc. Often this has led to clustered desktop displays and ineffective integration between communication media. Software development teams are adversely affected by the lack of integration between applications. When project managers, software developers and clients come together to work on one or more projects, certain communication and managerial considerations and requirements become apparent.

A server-based Integrated Messaging Framework (IMF) provides all communication media messages with a common messaging format. i-LINK, an intelligent client application uses IMF to send, receive, log, and store messages from different communication channels.

There are contrasting database models from which to choose when designing the IMF data model (i.e.: flat file, relational, object oriented, objected relational). However, the IMF was designed using a relational data model due to the stability it offered, the protection provided through referential integrity and constraints, as well as other unique benefits.

When considering the services that IMF is designed to provide, certain existing messaging standards are conceptually useful to analyze before designing the IMF data model. The RFC 2778 provides a universal Instant Messaging (IM) data model which can be mapped to provide such services as presence and instant messaging. Microsoft Outlook, a Messaging Application Programming Interface (MAPI), presents a data storage mechanism.

The IMF database model integrates concepts from existing messaging standards and refines the integration of different communication media while acting as a central repository.

Thesis Supervisor: Dr. George A. Kocur
Title: Senior Lecturer, Department of Civil and Environmental Engineering

- ACKNOWLEDGEMENTS -

I would first like to extend my deepest gratitude to Tarek Dajani without whom this project would not have been possible. Tarek you have an ambition and a dedication that I will always admire. Working with you was not only a profound learning experience, but also a delight.

I would like to thank my advisor, Dr. George Kocur, for his advice, valuable time, encouragement, and inspiring discussions. Thank you for all your assistance and guidance throughout this software development process.

To Patricia Crumley and Colleen O'Shea, thank you for making my experience at MIT unforgettable. Without your company, this year would have been twice as much difficult and not half as much fun. Thanks for being there for me and making me laugh.

To my parents, Michel and May Chahine, who have always been supportive and made me believe I could do things I never thought I could. I am always amazed at their confidence in me.

Finally, I would like to thank my brother, Pascal. For the past year his patience has been truly needed and his support during this project will always be remembered.

- TABLE OF CONTENTS -

- LIST OF FIGURES -

- CHAPTER ONE -
INTRODUCTION

The purpose of this document is to describe the database design development of IMF- an Integrated Messaging Framework- used by i-LINK- a communication organization and project management windows application designed by Masters of Engineering IT students at Massachusetts Institute of Technology.

1.1 PREAMBLE

Currently people have adapted to the use of instant messengers as a means of communication. Instant messengers provide people a method to view the status of individuals with whom they communicate; informing them if an individual is present or not at the moment of interest. Instant messengers combine this concept of presence with communication options including, but not limited to, messaging and file transfer. However, instant messengers only allow for communication if an individual is present at the time of request.

Therefore, individuals may resort to other means of communication of the messaging form. Some examples include emails, which also incorporate file sharing methods through attachments, and more recently blogs. This has ultimately lead many people to make use of one or more instant messenger applications, along with numerous emails, blog aggregators and file sharing applications simultaneously. Consequently, personal computers have to deal with a cluster of applications that rarely integrate while performing specific tasks. Organization of desktop display has lead to the creation of different software, such as Outlook and Eudora, which attempt to integrate communication in one unified display. However, within these applications the concept of presence is not adopted.

Members of a software development team - project managers, developers, and clients - are faced with these communication obstacles as they attempt to coordinate their activities. These three members of software development projects are constantly communicating with each other. Communication typically occurs through various emails, instant messages and blog postings, often with little or no tracking and logging of decisions and discussions performed through these means. Furthermore, a project, in addition to such communication problems, may run into other difficulties related to managing and coordinating the progress of a project. The members involved in software development projects are continuously producing, accessing and updating documents. These documents are often shared with each other, by either attaching them through emails and instant messages transfers, or placing them in repositories that can be accessed by all or accessed through websites.

We, a group of M.Eng IT students from Massachusetts Institute of Technology, saw a need to provide software that will overcome the communication difficulties many people face in their workplace, integrating access to all communication media through one desktop application. Principally, this piece of software provides functionality for members working on software development projects, offering a way to organize and share project information between several people but extends further to provide both presence and messaging services encompassing several communication media.

A sever-based Integrated Messaging Framework (IMF) and an intelligent client application, named i-LINK, were designed for the members involved in software development project to aid their project management and communication needs. The IMF server provides all communication media messages with a common messaging format as well as a common structure for contacts and addresses management. The i-LINK client uses this framework to send, receive, log, and store messages from different communication channels. Furthermore, the IMF database acts as a central repository and the i-LINK client uses the IMF database to centralize project information.

1.2 Background and Purpose

Designing a database is a vital component of application development; careful planning and design ensures high quality of the application. A database is a tool used to store and manage data. The aim of database design is fourfold. First, it attempts to ensure that appropriate data exists in the database. Second, database design endeavors to simplify the maintenance of the database structure, modification of data, and retrieval of information. Third, it tries to ensure that the structure of the database allows the data to be processed into meaningful, useful information. Finally, database design facilitates the development of software applications that utilize the database.

The purpose of this thesis is to describe the database design of the IMF framework, which is also used by the i-LINK client. These considerations include a study of the various database modeling approaches; followed by an analysis of two data model standards used by published and licensed applications; followed by software definition and requirements based on a study of the communication practices and needs of generic members of a software development project. The thesis then describes the data model selected for this software as well as how it was formed or designed and then explains how the two standards were used in designing the data model.

1.3 Extended Outline

Chapter 2 outlines and examines four database models: flat-file, relational, object-oriented, and object relational models. The advantages and disadvantages of each model will be assessed and compared. An analysis of the four models will provide an understanding as to which model is appropriate in the development of the IMF data model.

After having determined an appropriate database model for IMF, an examination of standard data models will be undertaken. Specifically, the analysis will focus on applications that provide services similar to those that will be provided through i-LINK. Chapter 3 will discuss two published standard data models currently used in licensed applications: Instant Messaging (IM) and Messaging Application Programming Interface (MAPI) data models. Microsoft Outlook, a personal information manager application, uses a MAPI data model. Although the MAPI data model is not a formally established standard data model, it has an extensive storage management data model. Similarly, i-LINK, through the use of the IMF framework, will provide a central storage repository.

MSN Messenger, AOL messenger, YAHOO Messenger – to list a few - are defined as presence and instant messaging services. The RFC 2778[1] provides a standard and universal IM data model that can be used across any messenger defined as a presence and instant messaging service. A study of IM design is necessary with reference to i-LINK bearing in mind the presence and instant messaging services that i-LINK will provide.

The project definition and requirements analysis will be assessed in Chapter 4. During the requirements analysis phase of any software project, research is conducted to gather all the information that will be used to design the system. The business rules and entities defined in the requirements then determine the design of the database. This chapter outlines communication requirements the project manager, developers, and clients working on a software development project by listing and defining the types of communication media they use, how and for what purpose they use them, and why each member values one medium over another. Furthermore, this chapter describes requirements needed for the management of project information common to all members.

The project data model is documented in Chapter 5. The data modeling phase is the process of visually representing the data and creating a data model with entities,

attributes, and relationships which leads up to the database design. This chapter provides a thorough outline of all entities and their corresponding relationships, and highlights the essential attributes in the relational IMF data model, fully normalized to reduce or eliminate any redundant data.

The analysis presented in this thesis will be summarized in Chapter 6. The database model for IMF will be compared to the data models outlined in Chapter 3. Such a comparison will provide an understanding as to the extent to which each of these standards has been utilized in the development of IMF, for further use with the i-LINK client. Furthermore, this chapter will discuss entities which will be implemented at a later date. These entities will have been designed within the IMF data model and will provide added functionality to the i-LINK client.

- CHAPTER TWO -

LITERATURE REVIEW: DATABASE MODELS

Once the requirements for a projected database are established, the core of database design may commence. This chapter provides a summary of benefits and drawbacks for the database models available. The i-LINK application and the Integrated Messaging Framework described in this thesis use database technology heavily.

2.1 DATABASE MODEL SELECTION

In order to make a decision on which database model to implement, it is important to understand the general concepts behind each database model. The following database models will be discussed in this section:

- Flat-File Database Model
- Relational Database Model
- Object Oriented Database Model
- Object Relational Database Model

2.1.1 FLAT-FILE DATABASE MODEL

A flat file database model is adequate for extremely simple and small databases. "It is made up of one more readable files stored in text format."[2] Each file has a number of fields, of constant or variable lengths, to store data. Once a flat file has been created and the data has been stored, a method to create, retrieve, update, or delete records must be incorporated. Hence, a set of many programs needs to be developed in order to access the data stored in the flat files. Using a flat-file database model requires both an understanding of "the structure of each file as well as knowledge of where the data is physically stored."[3] Even the simplest database will require several flat-files, which may have data related to

other data stored in other files. Hence, the process of managing data relationships in a flat-file database model is very difficult.

Drawbacks of a Flat-File Database Model[4]:

- Flat files do not promote a structure in which data can easily be related.
- It is difficult to manage data effectively and to ensure accuracy.
- It is usually necessary to store redundant data, which causes more work to accurately maintain the data.
- The physical location of the data field within the file must be known.
- A program must be developed to manage the data.

Recently, flat-file databases have improved. XML files, for example, are considered an advance to conventional flat-files. XML files are more manageable and can easily be read or written to from a database. However, limitations still exist. Simultaneous read/write capabilities do not exist for XML files and the relational integrity found in most databases is lacking. XML files are still not considered to be as efficient and effective as some of the following database models.

2.1.2 Relational Database Model

The relational database model is the most popular and stable model being implemented by designers. In a relational database model, a parent table can have several child tables, and a child table can have several parent tables. In comparison to flat files, the relational database model provides easier methods to manage data, retrieve data, and produce changes to data throughout the entire database. By placing rules on data – integrity constraints – data becomes easier to manage. Furthermore, in the relational database model, retrieving data stored does not require having knowledge of the database structure. Due to integrity

constraints and normalization, changes to data need only be done once and the changes will be generated throughout the entire database. The structure of an example relational database model is illustrated in Figure 2-1.

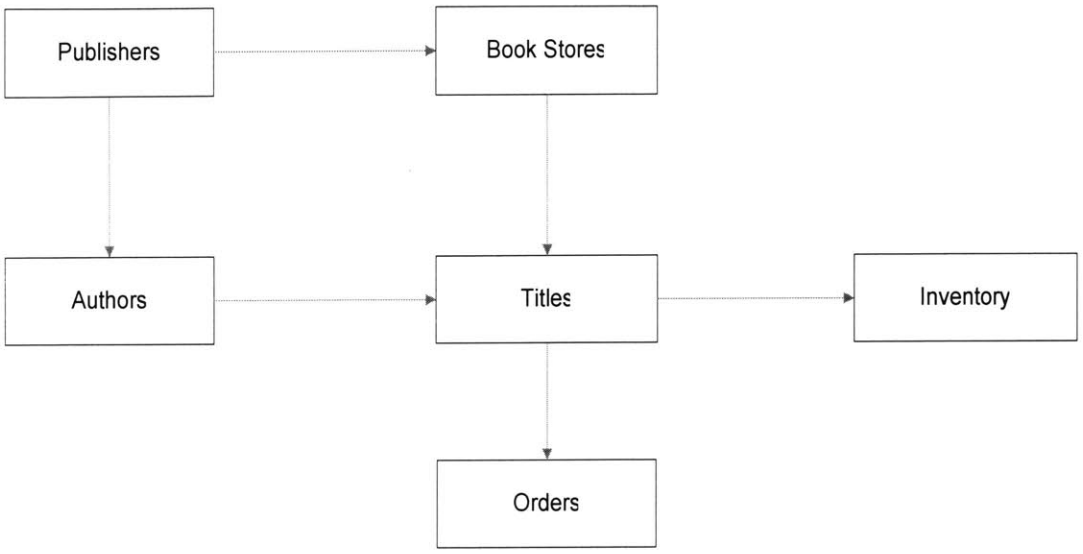

Figure 2 - 1 A Relational Database Model[5]

The relational database model is made up of tables consisting of columns and rows. Each row corresponds to a record and each column contains information for all rows. Different types of relationships can exist between tables in a relational database model: one-to-one, one-to-many, and many-to-many. Referential integrity is the process that ensures that data between related tables is consistent. Referential integrity is controlled by keys – column values that uniquely identify a row in a table or establish a relationship with another table. Primary keys are column values that make a row of data unique, while foreign keys are column values that reference primary keys from a related table.

Benefits of a Relational Database Model[6]:

- Data is accessed very quickly.
- The database structure is easy to change.

- The data is represented logically; therefore users need not understand how the data is stored.
- It is possible to develop complex queries to retrieve data.
- It is easy to implement data integrity.
- Data is generally more accurate.
- It is easy to develop and modify application programs.
- A standard language (SQL) has been developed.

Drawbacks of a Relational Database Model[7]:

- Different groups of information, or tables, must be joined in many cases to retrieve data.
- Users must be familiar with the relationships between tables.
- Users must learn SQL.

2.1.3 Object-Oriented Database Model

An object oriented database is a database in which data can be defined, stored, and accessed using an object oriented language. An object oriented database model uses programming languages such as C++, C#, and Java. Programmers use an object oriented programming language to work with objects to design an application that interacts with a relational database. The elements within a program or database application are represented as objects. Objects are assigned properties, which can be modified, and can also be inherited from other objects. Object oriented applications are easier to develop and maintain with object oriented programming tools. Programming tasks can be automated by an object oriented programming, which reduces the amount of time it takes to develop an application while increasing productivity.[8]

ObjectStore is one of the most successful vendors. ObjectStore has added

database features (such as persistence for objects, relationships between objects, and query expressions) to C++ and Java type systems and language constructs. Using an object-oriented database, ObjectStore delivers complex data management, real-time event processing, and middle-tier caching for Java and C++ applications.[9]

As object oriented programming technology progresses, developers of relational databases "must understand both the relational database language (SQL) as well as the object oriented programming language (Java, for example) that is to be used in order to design the application."[10] It is important for developers to understand relational database concepts in order for the application to access the data. It can be confusing for the developer to switch modes of thinking between relational and object oriented.

The two basic structures in an object oriented database are objects and literals. Objects have two characteristics, operations and properties, through which an object can be associated with other objects. Literals are values associated with objects. Operations are used to retrieve values from other classes, to add values, and to remove values. Properties can either be attributes or relationships. Objects and literals are organized by types, where all elements of a given type have the same set of properties, which can be modified for each individual object. A class is the equivalent of a table in a relational database; an attribute the equivalent of a table column; and an object instance the equivalent to a table row or tuple. Figure 2-2 illustrates how data is related in an object oriented database.

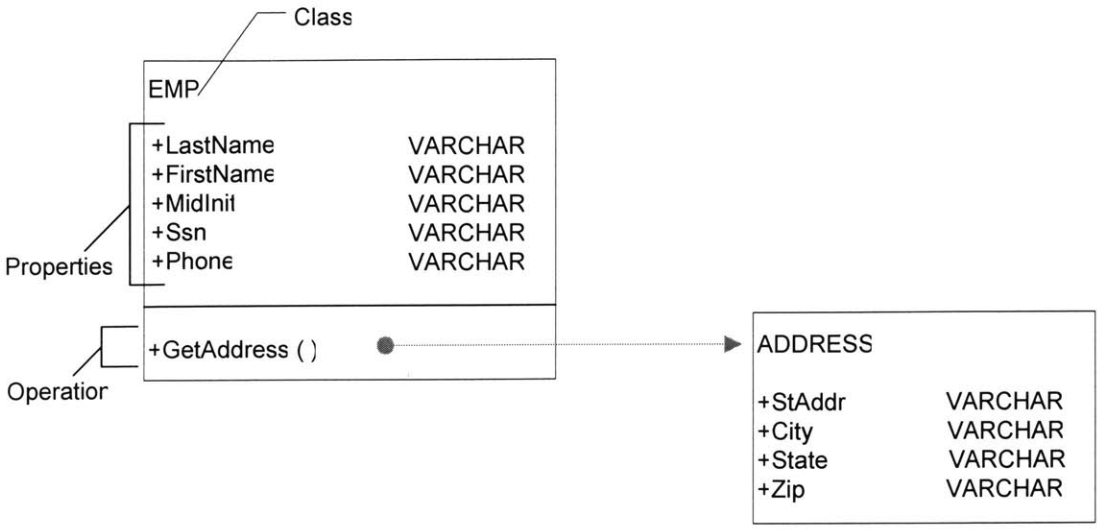

Figure 2 - 2 An Object Oriented Data Model[11]

Benefits of the object oriented model are as follows[12]:

- The programmer need only understand object oriented concepts as opposed to the combination of object oriented concepts and relational database storage.
- Objects can inherit property settings from other objects.
- Much of the application program process is automated.
- It is theoretically easier to manage objects.
- Object oriented data model is more compatible with object oriented programming tools.

Drawbacks of the object oriented model are as follows[13]:

- Users must learn object oriented concepts because the object oriented database does not work with the traditional programming methods.
- Standards have not been completely established for the evolving database model.
- Stability is a concern because object oriented databases are fairly recent.
- Performance is often poor.
- Lack of normalization hinders integrity.

- Not considered as effective as relational databases.

2.1.4 Object Relational Database Model

An object relational database combines concepts of both the relational database model and the object oriented programming approach. Object relational database model has only started to really grow recently and vendors are already incorporating object relational concepts into the new SQL standard, referred to as SQL3 or SQL99.

> Figure 2-3 illustrates an example object relational implementation in the Oracle9 relational database management system (RDBMS). Two user defined types have been created: PERSON and ADDRESS. Each type has columns that define specific data for a column in the base table, providing a 3D effect for the data. For example, the EMP_INFO column in the EMP table has a type of PERSON. PERSON is broken down into the specific categories LAST_NAME, FIRST_NAME, MID_INIT, and SSN.[14]

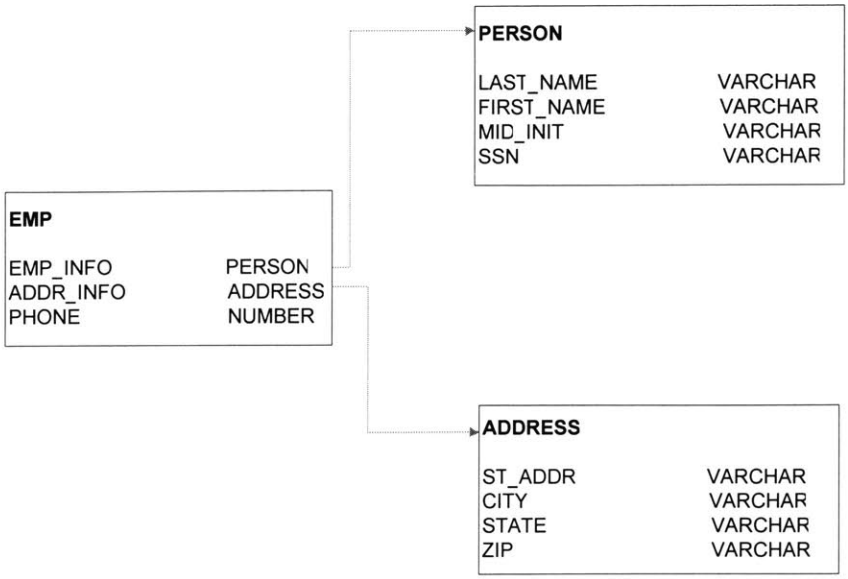

Figure 2 - 3 An Object Relational Database[15]

Benefits of the object relational model[16]:

- The relational database has more of a 3D architecture.
- User defined types can be created.

Drawbacks of the object relational model[17]:

- The user must understand both object oriented and relational concepts.
- Some vendors that have implemented OR concepts do not support object inheritance.

2.2 Database Model Chosen

Although the different database models each have their own benefits and drawbacks, a relational database model was chosen as the ideal model for this project. As opposed to a flat file database model, in the relational database model information is stored in tables that use parent/child relationships and provides a way in which the amount of redundant data can be reduced. Although the object oriented and object-relational database models do make data storage more compatible, these models need to

be further refined and improved.

After a comparison of the different data models, it was decided that the data model to be used for IMF should be a relational data model as it holds to be the most stable. The relational database standards are well established by organizations such as the International Standards Organization (ISO) and the American National Standards Institute (ANSI). There are many relational database vendors to choose from, including Oracle, Microsoft, IBM, and Sybase. It is easy to convert between different relational database implementation. It is easy to define, maintain, and manipulate data with SQL, the Standard Query Language used to define, query, modify, and control data in a relational database. Finally, the data is well protected through referential integrity and other constraints.

- CHAPTER THREE –

LITERATURE REVIEW: EXISTING MESSAGING STANDARDS

After having analyzed database technologies in the previous section, this chapter examines the other core technology upon which IMF is based: messaging standards. This chapter discusses application programming interfaces (APIs) for messaging, including a description of the MAPI, used by Microsoft Outlook, and an Instant Messenger model. These APIs assist in defining the key data and methods in IMF.

3.1 MAPI MODEL (MICROSOFT OUTLOOK)

3.1.1 FEATURES

Microsoft Outlook is a personal information manager (PIM). Like other PIMs, Outlook allows users to maintain information about contacts, keep track of daily schedules, keep track of tasks to complete, and other personal or work related information. Microsoft Outlook also provides email and fax support, group scheduling capabilities, and task management. Microsoft Outlook is a Messaging Application Programming Interface (MAPI) application, since MAPI message stores are the only data sources currently supported by it. MAPI is a set of API commands and functions used to send email. It has become the unofficially accepted standard messaging interface for Windows applications, providing a carefully defined set of messaging services. Access to MAPI services is the same for all versions of the Windows operating system.[18]

3.1.2 DATA MODEL

Microsoft Outlook uses an object data storage mechanism[19], implementing the MAPI standard. The MAPI, Microsoft Outlook, data model, shown in figure

3-1, is made up of seven main objects: Application, NameSpace, Folders collection, Items collections, Properties collections, Explorer, and Inspector objects.

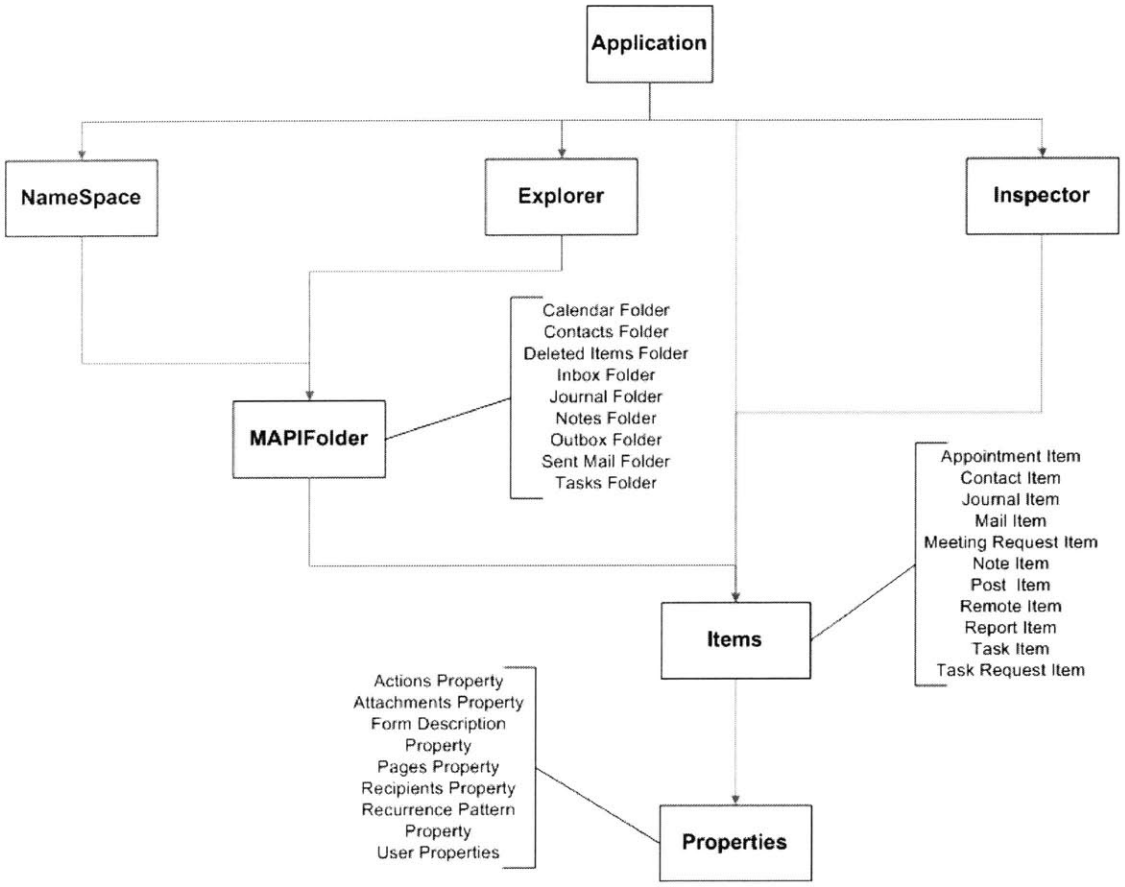

Figure 3 - 1 Microsoft Outlook Object Model

In the Outlook object model, the Application object contains the NameSpace object, which contains MAPIFolder objects that represent all the available folders in a given data source (for example a MAPI message store). The MAPIFolder objects contain objects that represent all the Outlook items in the data source, and each item contains some useful property objects for controlling that item. In addition, there is an Explorer object associated with each folder and an Inspector object associated with each item.

Application Object

The Application object is "the root object of the object model; it gives easy access to all the other objects in the model."[20] It gives direct access to the objects that represent the Outlook interface (the Explorer and the Inspector objects).

NameSpace Object

"The NameSpace object can represent any recognized data source, such as a MAPI message store. The object itself provides methods for logging in and out, returning objects directly by ID, returning default folders directly, and gaining access to data sources owned by other users."[21]

Folder Objects

The Folders collection contains all the MAPIFolder objects in the specified message store (or other recognized data source) or in a folder in that message store. The first time a user runs Outlook, some default folders are created. Each folder contains items of the same type. Default folders include the Calender, Contacts, Deleted Items, Inbox, Journal, Notes, Outbox, Sent Mail, and Tasks folder. Outlook also allows users to create further folders.[22]

- The Calender folder contains all Appointment Item objects.
- The Contacts folder contains all Contact Item objects.
- The Deleted Items folder is the storage area in which all item objects are placed when they have been deleted. The application has options that allow the user to retain these items indefinitely, archive them after a user defined period of time or purge them when the application is closed.
- The Inbox folder contains all Mail Item objects.
- The Journal folder contains all Journal Item objects.
- The Notes folder contains all Note Item objects.

- The Outbox folder is the storage area for items that are completed but not sent.
- The Sent Mail folder is the storage area in which copies of user generated Mail Item objects are moved when they are sent.
- The Tasks folder contains all Task Item objects.

Item Objects

The Items collection of a MAPIFolder object contains the objects that represent all the Outlook items in the specified folder. An Outlook Item can be one of several Outlook item object types. Outlook item objects include the Appointment Item, Contact Item, Journal Item, Mail Item, Meeting Request Item, Note Item, Post Item, Remote Item, Report Item, Task Item, and Task Request Item objects. [23]

- The Appointment Item objects represent an appointment in a Calender folder. An Appointment Item object can represent either a one time or recurring meeting or appointment.
- The Contact Item objects represent a contact in a Contacts folder. A contact can represent any person with whom the user has any personal or professional contact.
- The Journal Item objects represent a journal entry in a Journal folder. A journal entry represents a record of all Outlook moderated transactions for any given period of time.
- The Mail Item objects represent a mail message in the Inbox folder or another mail folder. The Mail Item is the default item object and to some extent the basic element of Outlook.
- The Meeting Request Item objects represent a change to the recipient's Calender folder, initiated either by another party or as a result of a group action.

- The Note Item objects represent a note (an annotation attached to a document) in a Notes folder.
- The Post Item objects represent a post in a public folder that other users can browse. This object has all the characteristics of the mail message. This object is similar to the Mail Item object, except that it is posted or saved rather then sent or mailed to a recipient.
- The Remote Item objects represent a remote item in the Inbox folder or another mail folder. This object is similar to the Mail Item object, but it contains only the Subject, Received, Date, Time, Sender, and Size properties and the first 256 characters of the body of the message. It gives the user who is connecting in remote mode enough information to decide whether or not to download the corresponding message
- The Report Item objects represent a mail delivery report in the Inbox folder or another mail folder. This object is similar to the Mail Item object and it contains a report (such as non-delivery report) or error message from the mail transport system.
- The Task Item objects represent a task in a Tasks folder.
- The Task Request Item objects represent a change to the recipient's task list initiated either by another party or as a result of a group assignment.

Property Objects

An Outlook item can access the following property objects: Actions, Attachments, Form Description, Recipients, Recurrence Pattern, and User Properties. Outlook items can be analyzed or modified by reading or setting its properties. In addition, every Outlook item can contain other objects that represent more complex qualities or behaviors of the item. For example, there are objects that represent the recipients of the item, the files attached to the item, and the customized pages and controls of the item.[24]

- The Actions property objects represent specialized actions that you can

perform on an item.

- The Attachments property objects represent linked or embedded objects contained in an item.
- The Form Description property objects represent the general properties of the form of an item.
- The Pages property objects represent the customized pages of an item. Every Inspector object has a Pages collection whose count is zero if the item has never been customized before.
- The Recipients property objects represent users or resources in Outlook; generally recipients are mail message addresses.
- The Recurrence Pattern property objects represent the pattern of incidence or recurring appointments and tasks for the associated Appointment Item and Task Item objects.
- The User Properties objects represent the custom fields added to an item in design time.

Explorer and Inspector Objects

The Explorer object represents the window in which the contents of a folder are displayed. The Inspector object represents the window in which an Outlook item is displayed.[25]

3.2 INSTANT MESSENGER MODEL

3.2.1 FEATURES

Presence is a way for finding, getting back, and subscribing to changes in the status, such as online or offline, of other users. Instant messaging is a way for sending small, simple messages that are delivered immediately to online users. A

presence and instant messaging system allows users to subscribe to each other and be notified of changes in state, and for users to send each other short instant messages.

RFC 2778[26] provides an abstract model for presence and instant messaging systems. It defines the various entities involved, defines terminology, and outlines the services provided by the system.

3.2.2 Data Model

RFC 2778 provides a descriptive and universal data model that is, and can be, used by any instant messengers that are described as presence and instant messaging services. Instant messenger applications use a database model and map the RFC 2778 model onto it.

In this section, an overview of the data model, shown in figure 3-2, is given. The overview includes a description of the services that outlines the core model entities; a description of the protocols that outlines how these core entities interconnect; a description of the *Principal* element and agents that outlines how user in the real work interact with the core entities; a description of the formats of *Presence Information* and *Instant Messages*; and finally an example of how this model is used. This model provides a way for understanding, comparing, and describing Instant Messenger systems that support the services referred to as presence and instant messaging. Developed instant messenger applications rarely have all the entities described in this model. However, each instant messenger database model will contain entities that encompass two or more elements of this model grouped in different ways.

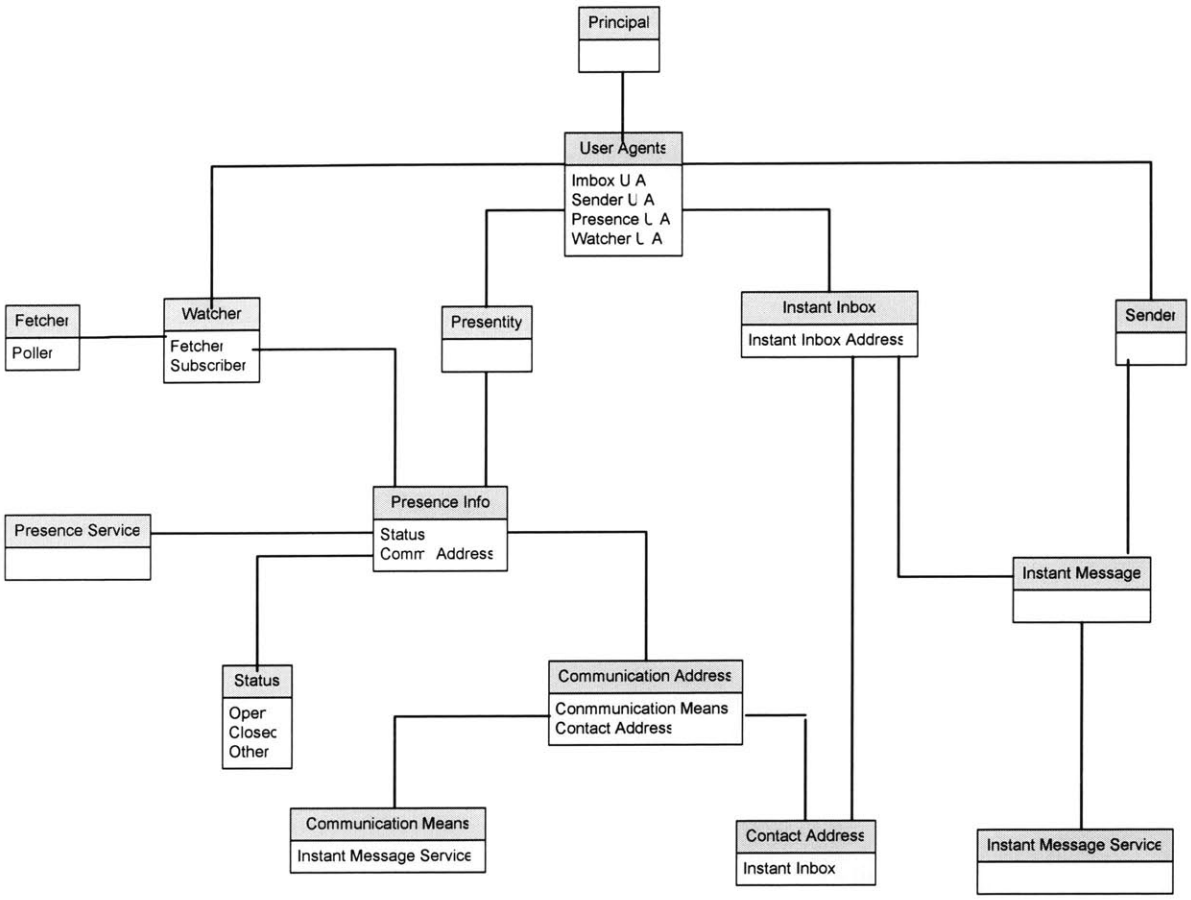

Figure 3 - 2 IM Model

Services

This model identifies two services: a *Presence Service* and an *Instant Message Service*. The *Presence Service* accepts, stores, and distributes information. This information stored is called *Presence Information*. The *Presence Service* has two types of clients: the *Presentities* and the *Watchers*. Developed instant messenger applications often combine these two entities into one.

Presentity (presence entity) provides *Presence Information* to the *Presence Service* to be stored and distributed. *Watcher* receives *Presence Information* about *Presentity* from the *Presence Service*. A *Watcher* can also

receive *Watcher Information* about another *Watcher*. *Watcher Information* is information about *Watchers* that have received *Presence Information* about a particular *Presentity*.

There are two kinds of *Watchers*, called *Fetchers* and *Subscribers*. A *Fetcher* asks the *Presence Service* to forward the *Presence Information* of one or more *Presentities*. A *Fetcher* that requests *Presence Information* on a regular basis is called a *Poller*. A *Subscriber* asks the *Presence Service* to notify it immediately of any changes in the *Presence Information* of one or more *Presentities*. Changes to *Presence Information* are distributed to *Subscribers* via *Notifications*.

The *Instant Message Service* accepts and delivers *Instant Messages* to *Instant Inboxes*. The *Instant Message Service* also has two types of clients: the *Senders* and the *Instant Inboxes*. A *Sender* provides *Instant Messages* to the *Instant Message Service* for delivery. Each *Instant Message* is addressed to a particular *Instant Inbox Address*, and the *Instant Message Service* delivers the message to a corresponding *Instant Inbox*.

Protocols

This model supports two types of protocols: the *Presence Protocol* and the *Instant Message Protocol*. A *Presence Protocol* is the messages that can be exchanged between *Presentity* and the *Presence Service* or between the *Watcher* and the *Presence Service*. The messages carried by the *Presence Protocol* are the *Presence Information*. An *Instant Message Protocol* is the messages that can be exchanged between the *Sender* and the *Instant Message Service* or between the *Instant Inbox* and the *Instant Message Service*. The messages carried by the *Instant Message Protocol* are the *Instant Messages*.

Formats for Presence Information and Instant Messages

In this model, the *Presence Information* consists of a random number of elements, called *Presence Tuples*. Each *Presence Tuple* consists of a *Status* marker, which gives information such as Online/Offline/Busy/Away/Do Not Disturb, an optional *Communication Address*, and an optional *Other Presence Markup*.

Status is defined by the model to have at least two state values *Open* and *Closed*, which determines the acceptance of *Instant Messages*. *Open* means *Instant Messages* will be accepted, and *Closed* means *Instant Messages* will not be accepted. *Open* and *Closed* may also be applicable to other *Communication Means*. *Open* can signify a state meaning available or open for business, while *Closed* means unavailable or closed to business. The model allows *Status* to have other state values that do not imply anything about *Instant Message* acceptance. These other values can be combined with *Open* (i.e.: Online,Away, Be Right Back) and *Closed* (i.e.: Offline,Busy) or can stand independently.

A *Communication Address* is made up of a *Communication Means* and a *Contact Address* attribute. *Communication Means* indicates a method whereby communication can take place. The only type of *Communication Means* defined by this model is the *Instant Message Service*. *Contact Address* is a specific point of contact via some *Communication Means*. The only type of *Contact Address* defined by this model is the *Instant Inbox Address*. However, other possibilities exist: a *Communication Means* might indicate some form of telephony, for example, with the corresponding *Contact Address* containing a telephone number.

Other Presence Markup is any additional information included in the *Presence Information* of a *Presentity*. This model does not define this any further.

An *Instant Inbox* is a container for *Instant Messages*. Its *Instant Inbox Address* is the information that can is included in *Presence Information* to define how an *Instant Message* should be delivered to that *Instant Inbox*. Finally, certain values of the status marker indicate whether *Instant Messages* will be accepted at the *Instant Inbox*.

Principals and Their Agents

This model includes other elements that are useful in illustrating how the protocols and formats work. The *Principal* element represents people, groups, and/or software in the real world outside the instant messenger system that use the system as a means of organization and communication.

A *Principal* interacts with the system via one of several user agents: *Inbox User Agent*; *Sender User Agent*; *Presence User Agent*; *Watcher User Agent*. A user agent is simply relating a *Principal* with a core entity in the system: *Instant Inbox*, *Sender*, *Presentity*, and *Watcher*. The *Inbox User Agent* is a way for a *Principal* to manipulate zero or more *Instant Inboxes* controlled by that *Principal*. The *Sender User Agent* is a way for a *Principal* to manipulate zero or more *Senders*. The *Presence User Agent* is a way for a *Principal* to manipulate zero or more *Presentities*. The *Watcher User Agent* is a way for a *Principal* to manipulate zero or more *Watchers* controlled by that *Principal*. In this model the different user agents are described separately, however developed instant messenger applications will combine at least some of them.

Example

Buddy List applications are simple examples of applying this model. These applications display a user's presence to others, and make it possible to see the presence of others. Therefore, a buddy list can be described as the combination of a *Presence User Agent* and *Watcher User Agent* for a single

Principal, using a single *Presentity* and a single *Subscriber*.

Instant messenger applications extend buddy lists to instant messaging. An instant messenger is a buddy list with additional capabilities for sending and receiving instant messages. Therefore an instant messenger can be described as the combination of a *Presence User Agent*, *Watcher User Agent*, *Inbox User Agent*, and *Sender User Agent* for a single *Principal*, using a single *Presentity*, single *Subscriber*, and single *Instant Inbox*, with the *Presentity*'s *Presence Information* including an *Instant Inbox Address* that leads to the *Instant Inbox*.

- CHAPTER FOUR –
I-LINK – AN INTEGRATED MESSAGING FRAMEWORK

4.1 BACKGROUND ON SOFTWARE DEVELOPMENT PROJECT

Software development is the design and implementation of a new piece of software application to meet the needs of a certain client or a group of clients. Software development encompasses the collaboration of a group of developers with different roles and skills who form a software development team. The software development team is managed by a project manager to whom developers report; both the project manager and the development team take on a project appointed through a client.

Project Manager

The project manager is responsible for both the technical and non-technical direction of a development team. However, a single project manager may work on multiple projects and, hence oversee different development teams. Therefore, a project manager may know little about how the development team functions day to day but is nevertheless responsible for the team's overall performance, progress and product outcome. The manager's role is to control and supervise each team so that it conforms to the goals and expectations of the client for the project at hand.

Software Developer

A software development team is made up of developers with diverse roles including but not limited to: requirement analysts, designers, database administrators, programmers, architects, support engineers, quality assurance engineers, and testers. Developers are responsible for technical work. A developer is able to work on more than one project simultaneously; currently, developers are often expected to finish up technical work on one project while beginning work on another.

Client

A project manager and software development team would not have much purpose if there was no client for whom software needed to be developed. A client who sets the goals, expectations, and requirements of the software needed does not necessarily have to be the final user of the software application.

Thus, the key actors involved in the success of a software development project are the project manager, software developers, and the client or clients.

4.2 Communication Across a Generic Software Development Project

Naturally, problems of communication and coordination arise with a large group of people. If there is only one person on a project, work can be performed in any manner desired because there is no need to communicate or coordinate with anyone else. As the number of people on a project increases, however, the number of communication paths and the amount of coordination needed increases. There is a nonlinear relationship between the number of people and the number of communication pathways, as seen in figure 4-1.

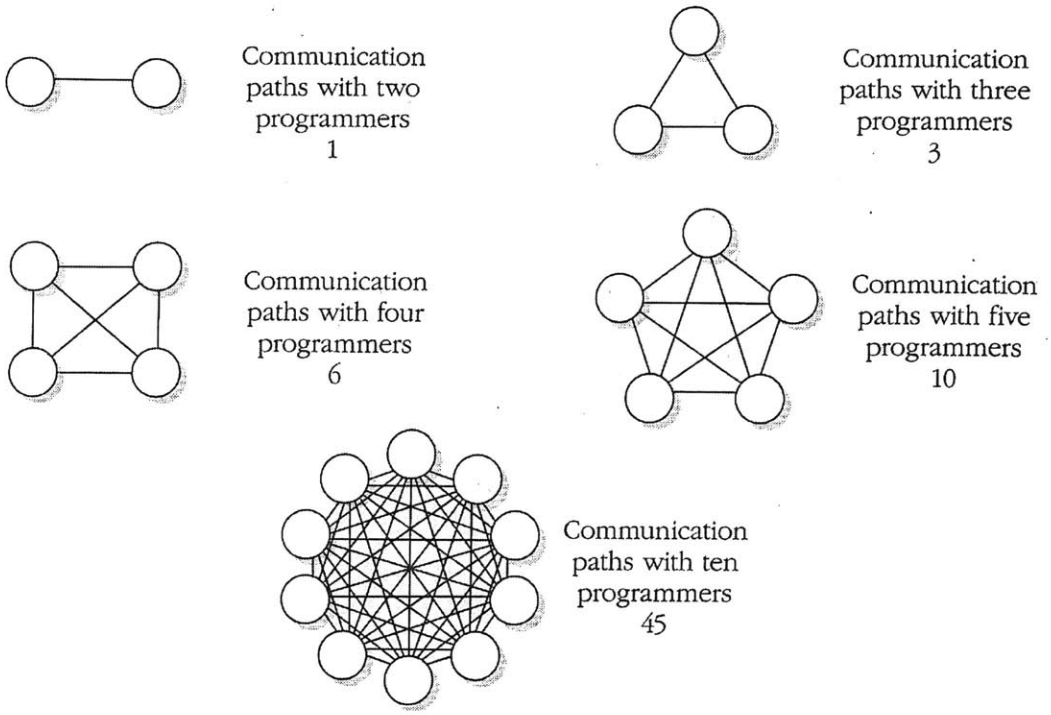

Figure 4 - 1 Communication Paths on Projects of Various Sizes[27]

> A two person project has only one path of communication. A five person project has 10 paths. A ten person project has 45 paths, assuming that every person talks to every other person. The two percent of projects that have 50 or more programmers have at least 1200 potential paths. The more communication paths that exist, the more time is spent communicating and the more opportunities there are for communication mistakes... Large projects call for organizational practices that formalize and streamline communication.[28]

Communication between the three entities of a software development project - project manager, developers and clients - requires a variety of communication means. They need to relay project information, questions, and messages; inform of discussions and decisions; and schedule and plan meetings. These are done either through emails, instant messages, and blogs; or by sharing documents and files through out the project pertaining to all phases of design and development of project. All documents are

primarily shared via attachments either through email or instant messages or Web sites, which are very common. Furthermore each entity needs to access, view and often update common project documents which often lead to documents being placed in a repository where they can be viewed and updated when required.

Project manager's main form of communication with developers, as well as clients, is often through the use of email. Email is considered a structured communication that can easily be referenced. Email is therefore often preferred by a project manager over non structured communication means such as instant messaging. Since project managers may work and manage several projects simultaneously, it is rarely effective to use a separate application for each project.

Developers communicate with the project manager and clients through email; however, they prefer to communicate with each other through blog postings or through instant messaging taking advantage of presence and real time communication.

Clients communicate primarily with project managers and occasionally with developers. Clients are often involved in several projects and their preferred means of communication is often email. Some clients want weekly/periodic reports or weekly/periodic meetings, which are also very common.

Ideally communication between the three entities would address the following concerns:

- Multiple projects require access and maintenance of several repositories.
- Communication between developers over instant messaging and blogs needs to be logged to keep track of decisions and discussions, as well as email communications between project managers, developers, and clients.
- Shared documents must be accessible from within different communication applications.

- Desktop displays may become clustered with multiple communication applications.
- One application should permit managing multiple projects simultaneously.
- Communication means may need to be used for purposes beyond the scope of the project (i.e. personal communication with friends and family)

4.3 Software Recommendation for a Software Development Project

We, a team of M.Eng IT students at MIT, recommend the development of a piece of software to meet the needs of a software development team encompassed by project management and communication organization. This can be achieved by bringing together the components of the various communication media used and by facilitating and centralizing retrieval of project information for all members involved. The recommended software that we have designed provides both an intelligent client application - named i-LINK - and an Integrated Messaging Framework (IMF).

4.3.1 i-LINK

i-LINK, a windows application, allows emails, instant messages, blogs, and files/documents to be sent, received, and shared between users. i-LINK uses the Integrated Messaging Framework web services to organize and share messages, contacts, and media accounts through projects, and uses the IMF database standard to centralize project information. i-LINK will facilitate communication and help coordination and organization between the members of any software development project.

4.3.2 Integrated Messaging Framework (IMF)

The Integrated Messaging Framework is a common messaging format and a common structure for contacts and addresses that can be accessed through a variety of client applications. The IMF can be accessed not only through a web-based or windows-based application (using web services), but also through blog aggregators (using RSS feeds), or through an email client (using notifications). Clients use this framework to send, receive, log, and store messages from the different communication media such as email, blog and instant messaging in one standard message format. Hence, the use of an IMF database standard acts as a central information repository. i-LINK works in conjunction with an IMF implemented in IIS and SQL server.

i-LINK which we have chosen to design using the .NET framework employs an IMF implemented in IIS and SQL server.

4.3.3 i-LINK Requirements

Defining the features and functionalities of the intelligent client, i-LINK, was based on the previously assessed concerns and needs of project managers, developers and clients. An analysis of these needs helped shape the i-LINK requirements, determine the type of data needed, the relationships between the data and the business rules generated. Ultimately, these help outline the IMF data model to be designed (Chapter 5). The remainder of this chapter describes the main requirements i-LINK fulfills.

i-LINK provides a single user interface that acts as unified messaging system as well as a project organization tool. The i-LINK client functions as an email system, instant messaging system, and/or blog reader, at the user's choice. The i-LINK client also provides a simplified view and quick access to all

information about messages, files, accounts, and contacts organized and grouped under projects all from one main page.

4.3.3.1 Messages

i-LINK, using the IMF, allows for instant chats to be logged and saved as instant messages. Email messages are sent, received, and saved as email messages. Blog entry notifications are received and logged as blog messages. File transfers are logged and saved as file transfer messages.

i-LINK provides a unified message view by displaying message headers for all four types of messages on the main page. The i-LINK client interface displays and organizes message headers in a combination of ways (All/Inbox/or Outbox combined with All/Email/IM/Blog/or File) as well as through selection of a specific contact or project from the main page.

i-LINK further provides a standard message composition and sending layout for the four types of messages. i-LINK allows the user to select the medium of communication for each recipient or use the set preferred medium of the recipient, while still having all communication accessible from single application and data store. i-LINK also offers an intelligent message handling feature which applies a user's and recipient's pre-set communication order preference to transmit message in most efficient manner. The i-LINK intelligent message handling feature allows for intelligent sending and receiving of messages, message handling. The user may set communication accounts in order of preference to be used for sending messages and to be used by other users of i-LINK in order that user can also receive message via preferred communication method.

i-LINK provides a message search function. Messages can be searched by specifying one or all of the following four criteria: Project, Contact, Inbox/Outbox, Message type (email, IM, blog, file).

4.3.3.2 Accounts

A user may configure i-LINK to handle email, instant messaging and blog entries, including file attachments/transfers. Multiple email, IM and blog accounts can be handled. A user may configure i-LINK to handle only a subset of media type accounts, and may continue to use other client software for the rest.

4.3.3.3 Contacts

i-LINK allows a user to create a contact list that can be easily and quickly accessed from the main page, allowing for instant view of contact information as well as instant message composition. There are two types of contacts in i-LINK: User Contacts and Guest Contacts.

User Contacts are users that have registered and created i-LINK accounts and have been added as a contact by another i-LINK user in order to be able to view and share files, messages and other project information with. A User Contact can be identified through any of their communication accounts set up with the IMF database. User Contacts that are added to an i-LINK client are added along with respective communication addresses and their order preference pre-selected, for use with the intelligent message handling feature. I-LINK users can further modify User Contact communication addresses' order preferences. An i-LINK client can view User Contact information; view the presence status of a User Contact; commence instant chat and file transfer with User

Contacts; and create instant message composition.

Guest Contacts are not users of i-LINK but have been created and added to a project by an i-LINK user in order to be able to view and share files, messages and other project information with. i-LINK users can specify and order Guest Contact communication addresses for use with the intelligent message handling feature. An i-LINK client can only view Guest Contact information and create instant message composition.

Contacts may be assigned to several projects and, hence, may be appear more than once under different projects. If a contact is not assigned to a specific project, the contact appears under the Main Space project (default project).

4.3.3.4 Projects

i-LINK allows user to create multiple Projects. A Project is a shared space to allow for file sharing, and logged message sharing with authorized contacts. Each Project contains an Accounts, Messages, and Contacts folder. The Accounts folder includes accounts that have been configured under this specific Project. The Messages folder contains all messages that have been sent or received through the Accounts configured for this Project. The Contacts folder includes all user and Guest Contacts that are members of this Project.

Each Project also has a Shared Files folder which allows for document sharing among contacts of Project. Shared Folder items are organized and displayed by date, subject and file name. When i-LINK user wishes to share a file, user does not have to send file as attachment separately to several contacts, but simply places file in the shared folder

under a certain project and the project contacts are instantly notified of the shared file added via i-LINK messages for user contacts or other clients for guest contacts.

Projects have two types of access, administrator rights are only granted to creator of Project and member rights granted to all contacts added to Project by administrator. Both administrator and members can add other contacts as members to a project; however, only the administrator can remove a contact. Therefore guest and user contacts can be automatically added and removed to a user's i-LINK client under a certain project by another user. The i-LINK user can then decide to view or hide displaying a certain contact in a Project. An i-LINK use may decide to unsubscribe from a Project, terminating membership. The administrator along with the members can further grant or deny other members access to view and use specific documents and messages. Projects are further labeled, by the administrator only, as public or private access to all others to be viewed.

- CHAPTER FIVE –
IMF DATA MODEL

The i-LINK client uses the Integrated Messaging Framework (IMF) database standard. After consideration of the different data model options discussed in Chapter 2 and an examination of the concepts employed in existing messaging standards (i.e.: MAPI and IM) discussed in Chapter 3, as well as the RFC 2822 email standard and the RSS 2.0 blog markup standard[29], it was decided that the data model to be used for IMF should be a relational data model as it holds to be the most stable. The relational database standards are well established by organizations such as the International Standards Organization (ISO) and the American National Standards Institute (ANSI). There are many relational database vendors to choose from, including Oracle, Microsoft, IBM, and Sybase. It is possible to convert between different relational database implementation. It is easy to define, maintain, and manipulate data with SQL, the Standard Query Language used to define, query, modify, and control data in a relational database. Finally, the data is well protected through referential integrity and other constraints.

The i-LINK data model was put in 5th normal form, providing greater overall database organization. All redundant data was eliminated. Normalization allowed data integrity to be easily maintained within the database; made the database and application design processes much more flexible; and made security easier to manage.

This chapter first describes the role of the six most important data entities: Users, Contacts, Addresses, AddressAccounts, MessageHeaders, and Project; followed by a description of the relationships between them and the other entities present in the IMF data model. Throughout the Figure 5-1 provides an illustration of IMF data entities and entity relationships.

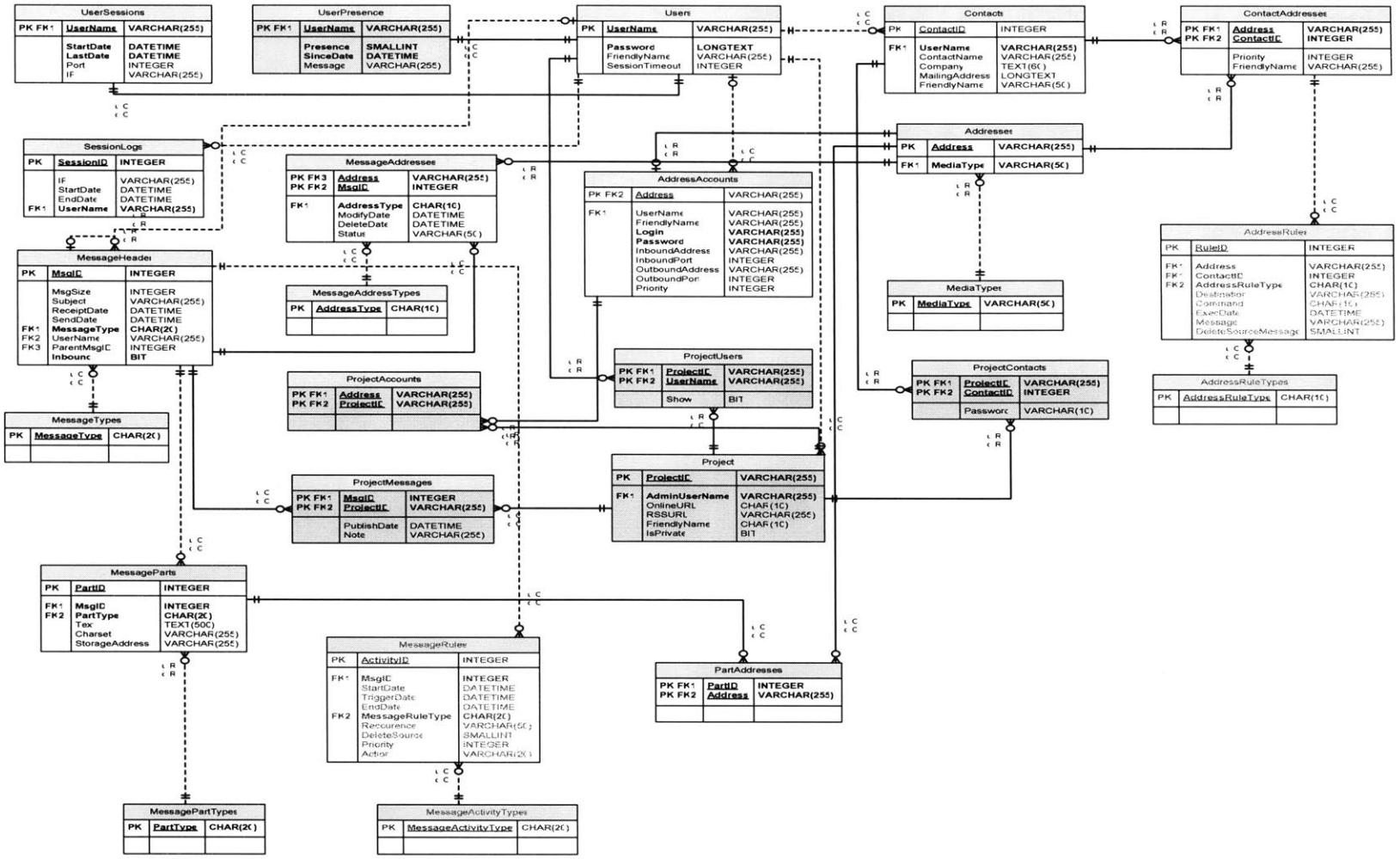

Figure 5 - 1 IMF Data Model

5.1 USERS ENTITY

All i-LINK registrants are included in the Users entity. User accounts are the i-LINK access points to the IMF database, which allow registrants to use the IMF web services to manage messages and project information. However, users can also access the IMF database through projects using other means, described later in this section.

Users records consist of *UserName* and *Password* attributes which allow a user entry to an i-LINK client. The *SessionTimeout* attribute in a Users record, dictates the time span after which a user record is propagated into a UserSessions record, further discussed in section 5.7.

Users		
PK	**UserName**	**VARCHAR(255)**
	Password	**LONGTEXT**
	FriendlyName	VARCHAR(255)
	SessionTimeout	INTEGER

Figure 5 - 2 Users Entity

Among the six most important entities, the Users entity is related to the Contacts, AddressAccounts, MessageHeaders, and Project entities, discussed section 5.2, 5.4, 5.5, and 5.6 respectively. Other entities related to the Users entity include the UserPresence, UserSessions, SessionLogs, ProjectUsers entities, discussed in section 5.7. Once a Users record is created, a corresponding UserPresence record is created further discussed in section 5.7.

5.2 CONTACTS ENTITY

This entity includes user owned and project owned contacts. A Contacts record includes all relevant information, optionally filled out, constituting the contact profile. A Users record can further be labeled as Contacts record providing additional profile

information for the user.

Contacts		
PK	ContactID	INTEGER
FK1	**UserName**	**VARCHAR(255)**
	ContactName	VARCHAR(255)
	Company	TEXT(60)
	MailingAddress	LONGTEXT
	FriendlyName	VARCHAR(50)

Figure 5 - 3 Contacts Entity

Among the six most important entities, the Contacts entity is related to the Users entity described in the previous section. Other entities related to the Contacts entity include the ContactAddresses and ProjectContacts entities, discussed in section 5.8.

5.3 ADDRESSES ENTITY

This entity defines global address entries. Each Addresses record is described by an *Address* attribute, which is the actual address text, and a *MediaType* attribute, which can be one of the following types: email, blog, IM, or system.

Addresses		
PK	**Address**	**VARCHAR(255)**
FK1	**MediaType**	**VARCHAR(50)**

Figure 5 - 4 Addresses Entity

Among the six most important entities, the Addresses entity is related to the AddressAccounts entity, discussed in next section. Other entities related to the Addresses entity include the ContactAddresses, MessageAddresses and PartAddresses, discussed in sections 5.8, 5.9, and 5.9 respectively.

5.4 ADDRESSACCOUNTS ENTITY

This AddressAccounts entity simply defines IMF accounts. An AddressAccounts record further describes a user's Addresses record by providing additional information for accessing account servers. An AddressAccounts record contains *InboudAddress*, *InboundPort*, *OutboundAddress*, *OutboundPort* attributes in order to provide the location for the media account server. The *Login* and *Password* attributes in an AddressAccounts record provide the authentication information to access the server. Finally the *Priority* attribute holds the user's order preference for the AddressAccounts record in order to be used during the intelligent message handling feature.

AddressAccounts		
PK,FK2	Address	VARCHAR(255)
FK1	UserName	VARCHAR(255)
	FriendlyName	VARCHAR(255)
	Login	**VARCHAR(255)**
	Password	**VARCHAR(255)**
	InboundAddress	VARCHAR(255)
	InboundPort	INTEGER
	OutboundAddress	VARCHAR(255)
	OutboundPort	INTEGER
	Priority	INTEGER

Figure 5 - 5 AddressAccounts Entity

Among the six most important entities, the AddressAccounts entity is related to the Addresses and Users entities discussed in the previous sections. Another entity related to the AddressAccounts entity is the ProjectAccounts entities, discussed in section 5.10.

5.5 MESSAGEHEADERS ENTITY

Although there is no individual message entity, however messages are the fundamental components of the IMF database. Messages are any form of data exchange, these include all emails, instant messages, blog postings, files shared and files transferred. The MessageHeaders entity simply defines the IMF messages and represents what could

have been called a message entity. Messages can be exchanged in an IMF project. Each Message Header record is mainly described by a *Subject* attribute; a *ReceiptDate* attribute; a *SendDate* attribute; and a *MessageType* attribute, which can either be a document or a message.

MessageHeader		
PK	**MsgID**	**INTEGER**
	MsgSize	INTEGER
	Subject	VARCHAR(255)
	ReceiptDate	DATETIME
	SendDate	DATETIME
FK1	**MessageType**	**CHAR(20)**
FK2	UserName	VARCHAR(255)
FK3	ParentMsgID	INTEGER
	Inbound	**BIT**

Figure 5 - 6 MessageHeaders Entity

Among the six most important entities, the MessageHeaders entity is related the Users entity, discussed in section 5.1. Other entities related to the MessageHeaders entity include MessageAddresses, ProjectMessages and MessageParts, described in section 5.9, 5.11, and 5.11 respectively.

5.6 PROJECTS ENTITY

This entity allows a user to group contacts, accounts, and messages into a shared space. Project is the element used to share information with contacts, using the i-LINK client, a blog aggregator, the World Wide Web, or an email account.

A Project record is owned by one registered user, *AdminUserName*, who is the creator of the project. Project content is accessible via the web using the *OnlineURL* attribute, or via a blog aggregator using the *RSSURL* attribute. In order to provide privacy to project content on the web, a project a described as public or private using the *isPrivate* attribute. Public means that the project can be accessible without authentication,

where as private requires authentication.

Project		
PK	**ProjectID**	**VARCHAR(255)**
FK1	**AdminUserName**	**VARCHAR(255)**
	OnlineURL	CHAR(10)
	RSSURL	VARCHAR(255)
	FriendlyName	CHAR(10)
	IsPrivate	BIT

Figure 5 - 7 Project Entity

Among the six most important entities, the Project entity is related the Users entity, discussed in section 5.1. As stated earlier, a project allows a user to organize accounts and messages and share information with contacts; therefore other entities related to the Project entity, are the ProjectUsers, ProjectContacts, ProjectAccounts, and ProjectMessages entities, discussed in sections 5.7, 5.8, 5.10, and 5.11.

5.7 Other Entities Related to Users

The Users entity has a direct one-to-many relationship with the following previously described entities: Contacts, AddressAccounts, Project, and MessageHeaders. That is each user can have one or more contacts associated with him/her; each user can have one or more accounts registered with the system; each user can create and own one or more projects; and each user can have one or more messages associated with him/her. The Users entity also has some relationships with the following entities: UserPresence, UserSessions, SessionLogs, and ProjectUsers. All related entities are shaded in the figure below.

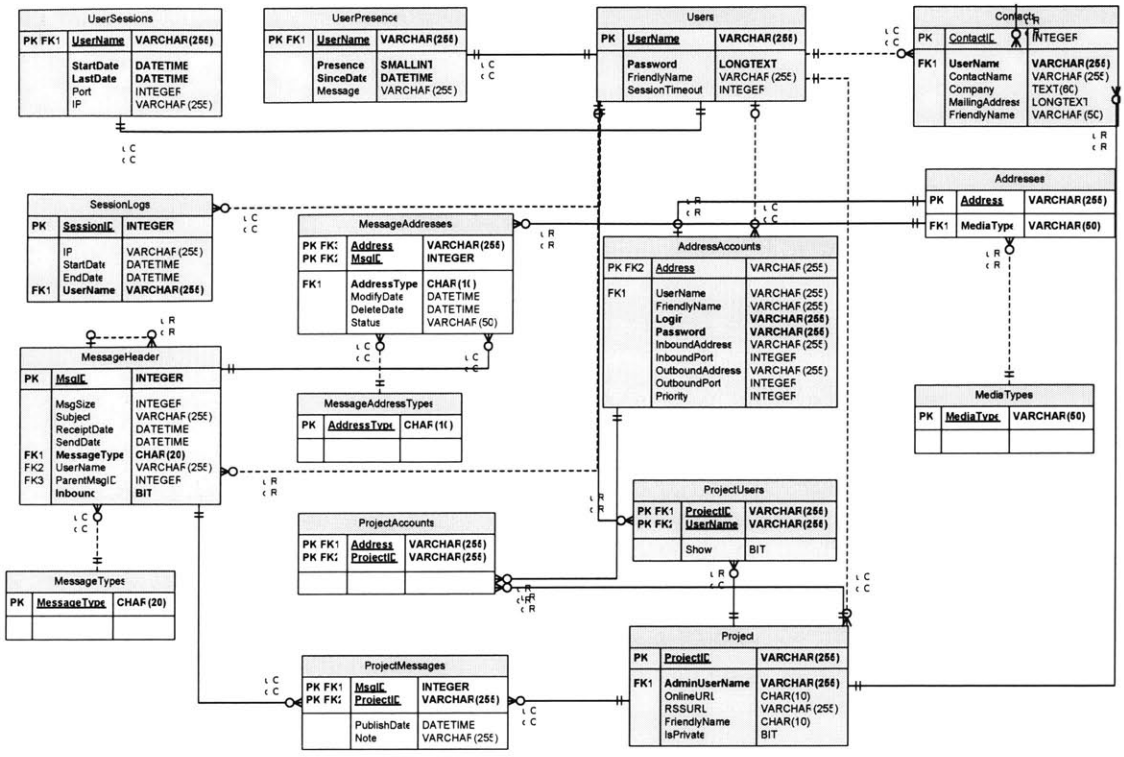

Figure 5 - 8 Entities Related to Users

UserPresence

This entity describes a user's status, *Presence* attribute, and the date and time since this status has been in effect, *SinceDate* attribute. The Users entity has a direct one-to-one relationship with the UserPresence entity. Once a user record is created a user presence record is created in correspondence. A user can only have one user presence at any time; however the *Presence* and *SinceDate* attributes are altered as the user changes his/her status while using the client.

UserPresence		
PK,FK1	**UserName**	**VARCHAR(255)**
	Presence	**SMALLINT**
	SinceDate	**DATETIME**
	Message	VARCHAR(255)

Figure 5 - 9 UserPresence Entity

UserSessions

This entity describes a session for a user using the system by keeping track of the *Port* and *IP* address attributes for each user. The *LastDate* attribute represents the last time the user accessed or did something using the client. The Users entity has a direct one-to-one relationship with the UserSessions entity. A UserSessions record is created when a user is actively using the client. A user can only have a one UserSessions record at any time; however, the *LastDate* attribute is altered as the user accesses and makes use of the client.

UserSessions		
PK,FK1	**UserName**	**VARCHAR(255)**
	StartDate	**DATETIME**
	LastDate	**DATETIME**
	Port	INTEGER
	IP	VARCHAR(255)

Figure 5 - 10 UserSessions Entity

SessionLogs

This entity keeps track of what users accessed the system and when. It is a list of all sessions that have ended or have been timed out. A session is said to be timed out, if it has been inactive for thirty minutes. The Users entity has a direct one-to-many relationship with the SessionLogs entity. A SessionLogs record is created once a UserSessions record is closed or has been timed out. A user can have used and created one or more sessions, therefore a user will have one ore more SessionLogs records associated with him/her.

SessionLogs		
PK	**SessionID**	**INTEGER**
	IP	VARCHAR(255)
	StartDate	DATETIME
	EndDate	DATETIME
FK1	**UserName**	**VARCHAR(255)**

Figure 5 - 11 SessionLogs Entity

ProjectUsers

This entity is a join entity to implement a many-to-many relationship between the Users and Project entities. A project can have one or more system users subscribed to it, and similarly a user can belong to one to more projects. To avoid data redundancy, the ProjectUsers entity and two one-to-many relationships were created eliminating a direct relationship between the Users and Project entities.

ProjectUsers		
PK,FK1 **PK,FK2**	**ProjectID** **UserName**	**VARCHAR(255)** **VARCHAR(255)**
	Show	BIT

Figure 5 - 12 ProjectUsers Entity

5.8 Other Entities Related to Contacts

As mentioned previously, the Contacts entity has a direct many-to-one relationship with the Users entity. Additionally, the Contacts entity has some direct relationships with the following entities: ContactAddresses and ProjectContacts. All related entities are shaded in the figure below.

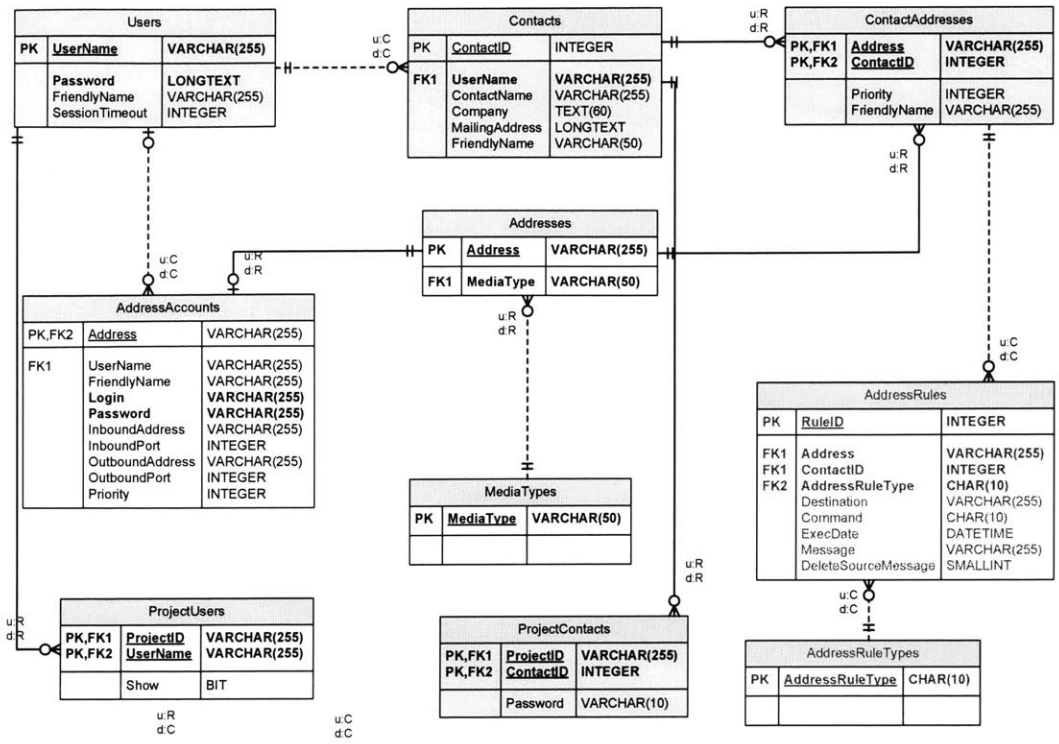

Figure 5 - 13 Entities Related to Contacts

ContactAddresses

This entity groups the Contacts and Addresses entities into one entity, in order to allow easier processing. The Contacts entity has a direct one-to-many relationship with the ContactAddresses entity. A contact can be related to one or more records in the ContactAddresses entity depending on the number of addresses associated with him/her. Similarly, an address can be related to one or more Contacts records as well depending on the number of contacts the address belongs to. The *Priority* attribute holds the contacts order preference for the ContactAddresses record in order to be used during the intelligent message handling feature.

ContactAddresses		
PK,FK1 **PK,FK2**	**Address** **ContactID**	**VARCHAR(255)** **INTEGER**
	Priority FriendlyName	INTEGER VARCHAR(255)

Figure 5 - 14 ContactAddresses Entity

ProjectContacts

This entity is a join entity to implement a many-to-many relationship between the Contacts and Project entities. A project can have one or more contacts subscribed to it, and similarly a contact can belong to one to more projects. To avoid data redundancy, the ProjectContacts entity and two one-to-many relationships were created eliminating a direct relationship between the Contacts and Project entities. Each ProjectContacts record has a *Password* attribute; this allows a contact access to the project if that project is labeled as a private project.

ProjectContacts		
PK,FK1 **PK,FK2**	**ProjectID** **ContactID**	**VARCHAR(255)** **INTEGER**
	Password	VARCHAR(10)

Figure 5 - 15 ProjectContacts Entity

5.9 Other Entities Related to Addresses

As mentioned previously, the Addresses entity has a direct one-to-many relationship with the AddressAccounts entity; that is an address can belong to one or more accounts. The contacts entity also has a direct relationship with the MessageAddresses entity. Related entities are shaded in the figure below.

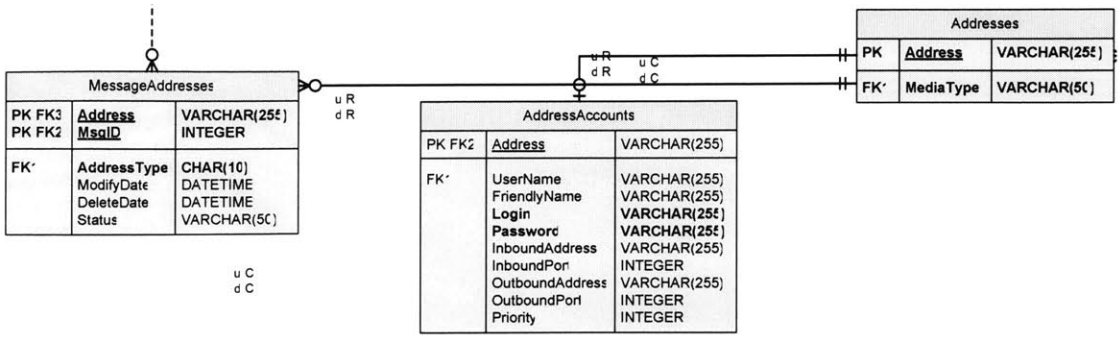

Figure 5 - 16 Entities Related to Addresses

MessageAddresses

This entity is a join entity to implement a many-to-many relationship between the Addresses and MessageHeaders entities. An address can be assigned to one or more messages and similarly a message can have one ore more addresses tagged to it. To avoid data redundancy, the MessageAddresses entity and two one-to-many relationships were created eliminating a direct relationship between the Addresses and MessageHeaders entities. Each MessageAddresses record is defined using an *AddressType* attribute, which can be one of the following types: BCC, CC, To, or From.

MessageAddresses		
PK,FK3 **PK,FK2**	**Address** **MsgID**	**VARCHAR(255)** **INTEGER**
FK1	**AddressType** ModifyDate DeleteDate Status	**CHAR(10)** DATETIME DATETIME VARCHAR(50)

Figure 5 - 17 MessageAddresses Entity

5.10 Other Entities Related to AddressAccounts

As mentioned previously, the AddressAccounts entity has a direct many-to-one relationship with the Users and Addresses entities. Additionally, the AddressAccounts entity has one further direct relationship with the ProjectAccounts entity. All related entities are shaded in the figure below.

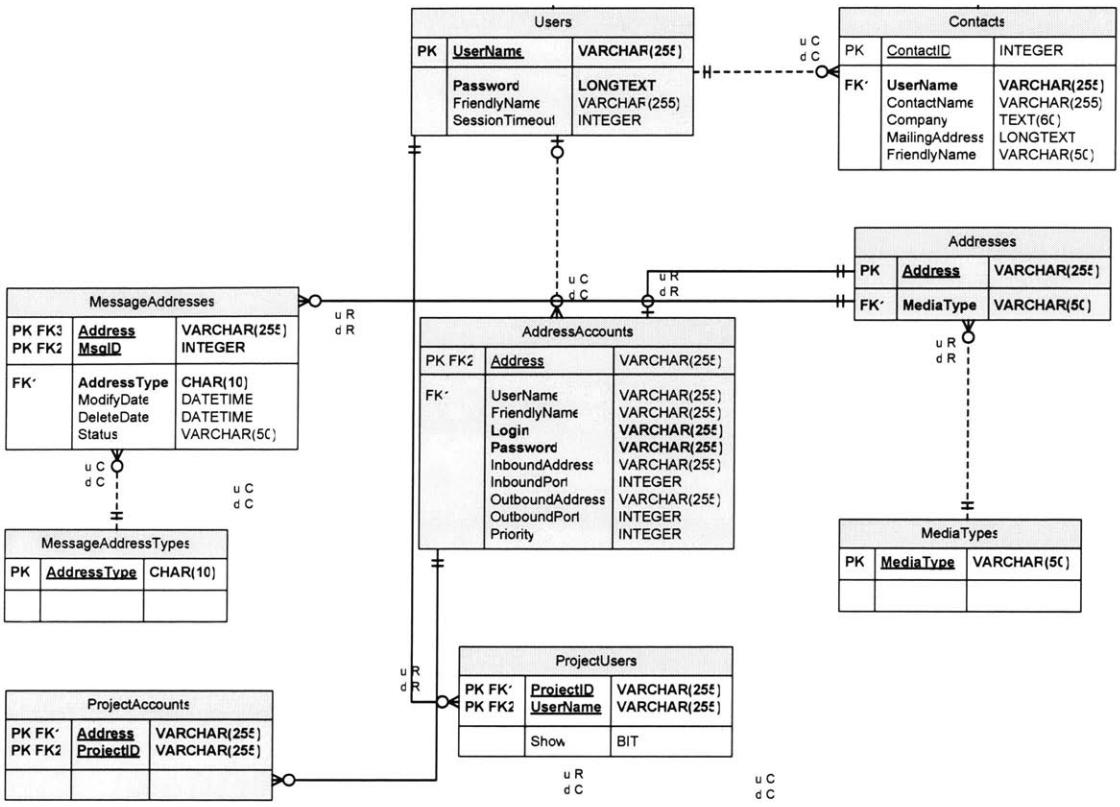

Figure 5 - 18 Entities Related to AddressAccounts Entity

ProjectAccounts

This entity is a join entity to implement a many-to-many relationship between the AddressAccounts and Project entities. An account can be added to one or more projects, and similarly a project can have one or more accounts assigned to it. To avoid data redundancy, the ProjectAccounts entity and two one-to-many relationships were created eliminating a direct relationship between the AddressAccounts and Project entities.

ProjectAccounts		
PK,FK1 PK,FK2	Address ProjectID	VARCHAR(255) VARCHAR(255)

Figure 5 - 19 ProjectAccounts Entity

5.11 Other Entities Related to MessageHeaders

As mentioned previously, the MessageHeaders entity has a direct many-to-one relationship with the Users entity. Additionally, the MessageHeaders entity has other direct relationships with the ProjectMessages and MessageParts entities. All related entities are shaded in the figure below.

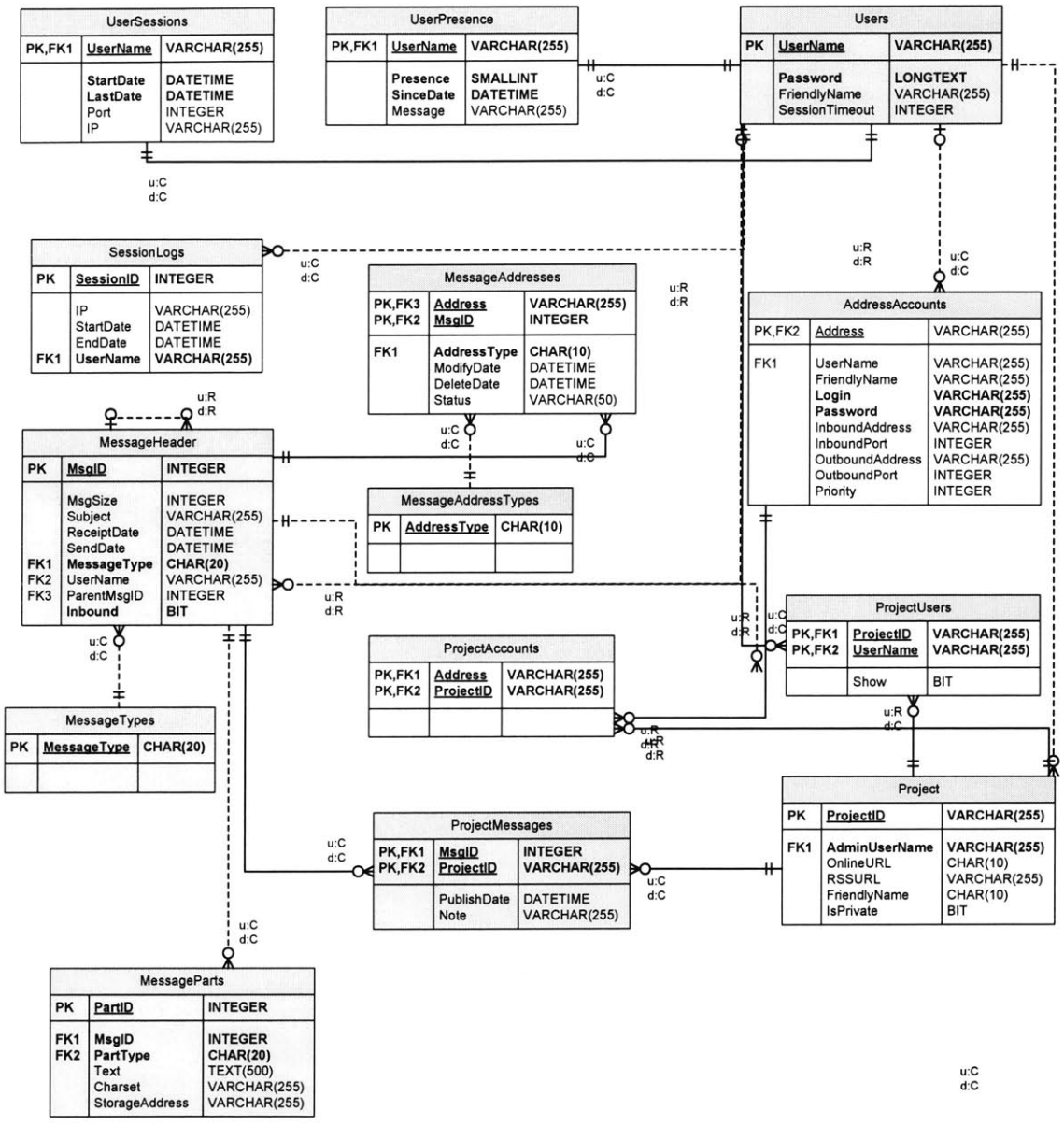

Figure 5 - 20 Entities Related to MessageHeaders

ProjectMessages

This entity is a join entity to implement a many-to-many relationship between the MessageHeaders and Project entities. A message can be added to one or more projects and similarly a project can contain one or more messages. To avoid data redundancy, the ProjectMessages entity and two one-to-many relationships were created eliminating a direct relationship between the MessageHeaders and Project entities.

ProjectMessages		
PK,FK1 **PK,FK2**	**MsgID** **ProjectID**	**INTEGER** **VARCHAR(255)**
	PublishDate Note	DATETIME VARCHAR(255)

Figure 5 - 21 ProjectMessages Entity

MessageParts

This entity describes the content of a message, such as a document file, an image, or simple text. The MessageHeaders entity has a direct one-to-many relationship with the MessageParts entity. A MessageHeaders record can have one or more MessageParts records, defined by the *MessagePartType* attribute, which can either be a file, image, or text.

MessageParts		
PK	**PartID**	**INTEGER**
FK1 **FK2**	**MsgID** **PartType** Text Charset StorageAddress	**INTEGER** **CHAR(20)** TEXT(500) VARCHAR(255) VARCHAR(255)

Figure 5 - 22 MessageParts Entity

5.12 Other Entities Related to Projects

As mentioned previously, the Project entity has a direct many-to-one relationship

with the Users entity; and a direct relationship with the ProjectContacts, ProjectUsers, ProjectAccounts, and ProjectMessages entities. All related entities are shaded in the figure below.

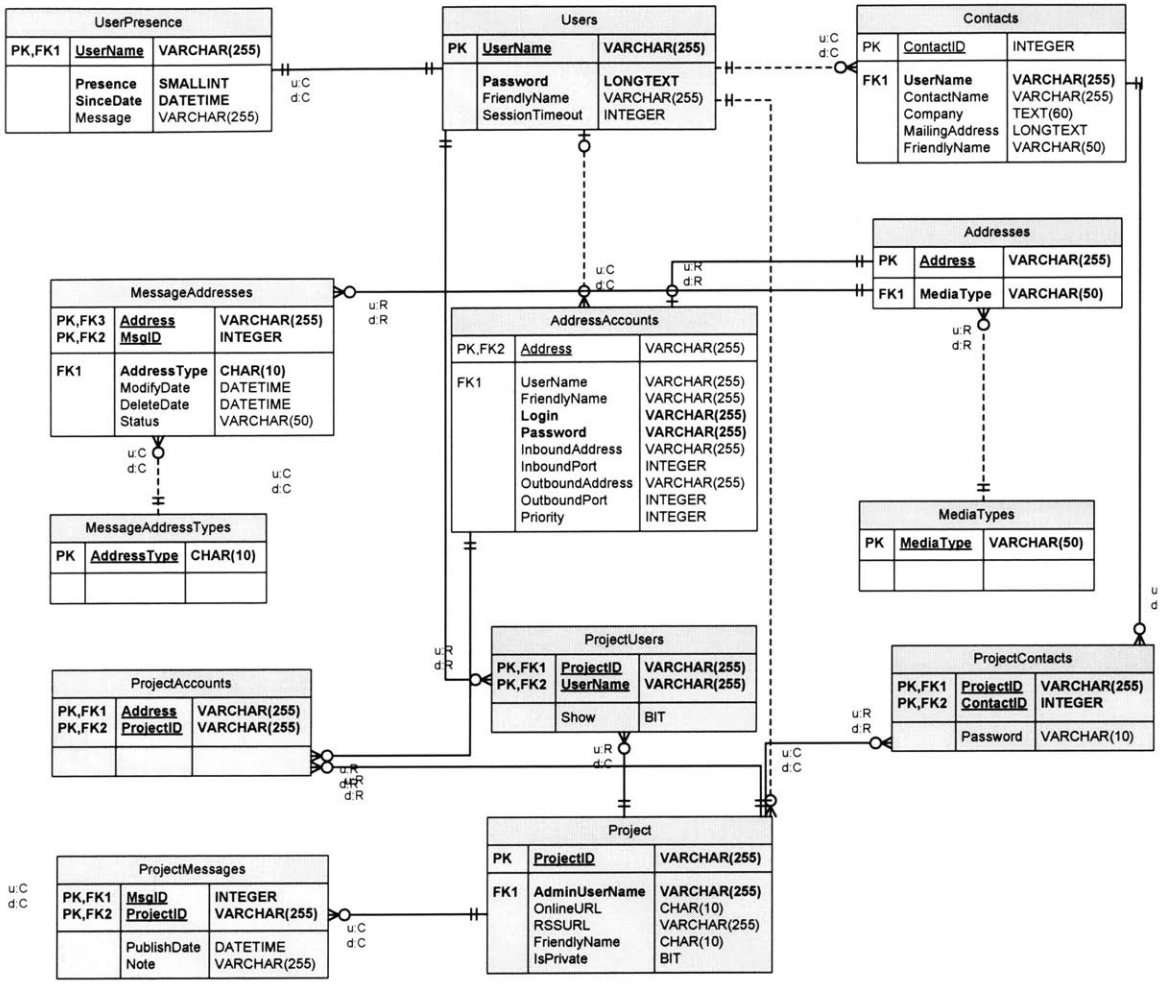

Figure 5 - 23 Entities Related to Projects

- CHAPTER SIX -

SUMMARY AND CONCLUSION

6.1 REVIEW

In recognition of the communication and management difficulties particular to a generic development team a new application – the i-LINK client – was recommended and was to be supported by an Integrated Messaging Framework (IMF). The database development of IMF began by identifying the different types of database models upon which IMF may have been built. These four models were: flat-file, relational, object-oriented, and object-relational. A comparison between these four types led to the conclusion that the relational database model provided the greatest stability. Furthermore, the abundance of vendors and the fact that the relational database standard is well established increased its appropriateness. Finally, the protection and manageability of data through the relational model secured the decision to utilize it in the development of the IMF database model.

Another look at data models, this time at published models employed in licensed applications, aided to conceptualize the formation of the IMF data model. The MAPI model, currently implemented in Microsoft Outlook, was discussed. An understanding of the objects it employs and their relationships ensued. The MAPI model provided a conceptual framework upon which the repository storage ability of IMF was to be developed. Similarly, the IM model – employed by a variety of today's instant messengers – outlined a series of entities. These entities, and their relationships, were analyzed for later use in providing presence and instant messaging services in the i-LINK client.

At such a point, design of the IMF data model was virtually able to be begun. However, an analysis of a generic software development team was needed to finalize the

specific requirements of the recommended software – the i-LINK client. The requirements discovered from this analysis were loosely subdivided into four categories: messages, accounts, contacts and projects.

The IMF data model implemented by i-LINK was, hence, provided. Within the discussion of the data model was a detailed analysis of all the entities created. These entities were conceptually distinguished as main entities and entities that relate to these main entities.

6. 2 COMPARISONS

As previously stated, the IMF data model (Chapter 5) took form based on the analysis and study of two data models: IM and MAPI (Chapter 3). However, both models were inadequate in providing a data model framework sufficient to meet the requirements established for i-LINK (Chapter 4). Therefore, several entities in the IMF data model were created that go beyond both the IM and MAPI data model frameworks. The following sections explain how the concepts of the MAPI data model framework and the elements of the IM data model framework were used within the IMF data model. Furthermore, additional entities that will be implemented at a later date, and, hence, have been removed from the analysis of the IMF data model, are outlined and defined.

6.2.1 IMF DATA MODEL: MAPI CONSIDERATIONS

As previously described, i-LINK provides an email messaging service and hence uses a MAPI message store similar to Microsoft Outlook. However, the Microsoft Outlook MAPI data model (Chapter 3), shown in figure 6-1, includes not only an email messaging service but also a variety of personal information management services that i-LINK does not implement at this time, including: calendar, journal and task services. Therefore the framework of the MAPI data

model is not fully used within the IMF data model. Therefore, discussed in this section are the main concepts of the MAPI data model that were used in the IMF data model. While concepts discussed may reference a specific folder, item or property object, within the context of the following discussion these concepts are meant to incorporate all folders, items and properties related to that reference.

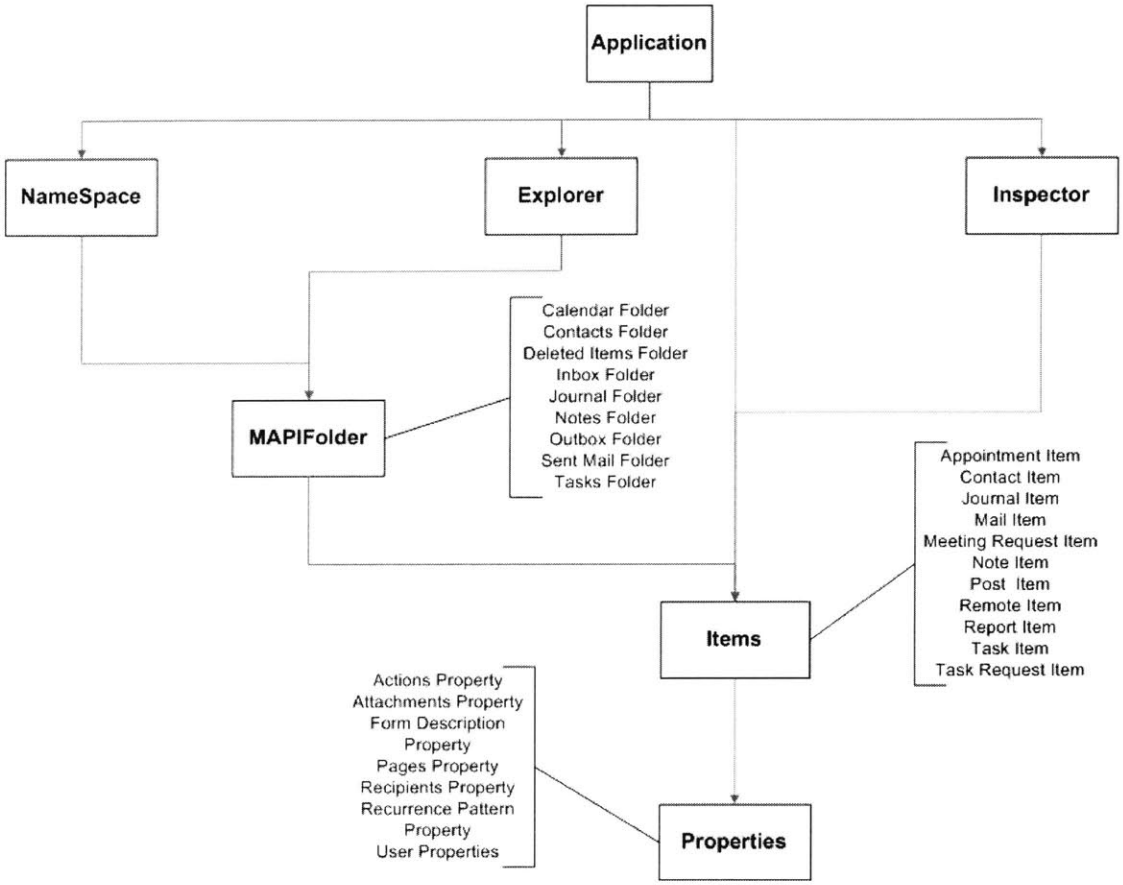

Figure 6 - 1 MAPI (Microsoft Outlook) Model

The concept Remote Item[30] in the MAPI data model is used in the IMF data model through the MessageHeaders entity, containing the *Subject*, *ReceiptDate*, *Size*, *UserName* (which is the sender), giving the user enough information to decide whether or not to download the message. The Mail Item[31] concept in the MAPI data model is also used in the IMF data model through the MessageHeaders entity representing all communication messages sent or received

and is similarly considered the basic element of IMF. The concept of Attachments[32] in the MAPI data model is used in the IMF data model through the MessageParts entity to further describe the content embedded in the MessageHeader record.

The concept of a Post Item[33] in the MAPI data model is used in the IMF data model through the Project Messages entity. Project Messages records are similarly saved and posted into a project, similar to a public folder, in order to be shared and used by other contacts.

The concept of Contact Item[34] in the MAPI data model is used the IMF data model through the Contacts entity representing any person with whom the user has any personal or project related contact relations with.

The concept of Recipients[35] in the MAPI data model is used in the IMF data model through the Users and Address Accounts entities. Both users and accounts represent resources that make use or are used by the i-LINK client.

6.2.2 IMF Data Model: IM Model Considerations

The IMF data model fully maps all the IM data model elements, services and agents. The IMF data model defines entities that encompass two or more elements of the IM data model. This section describes how the IM data model elements, shown in figure 6-2, were used in the IMF data model.

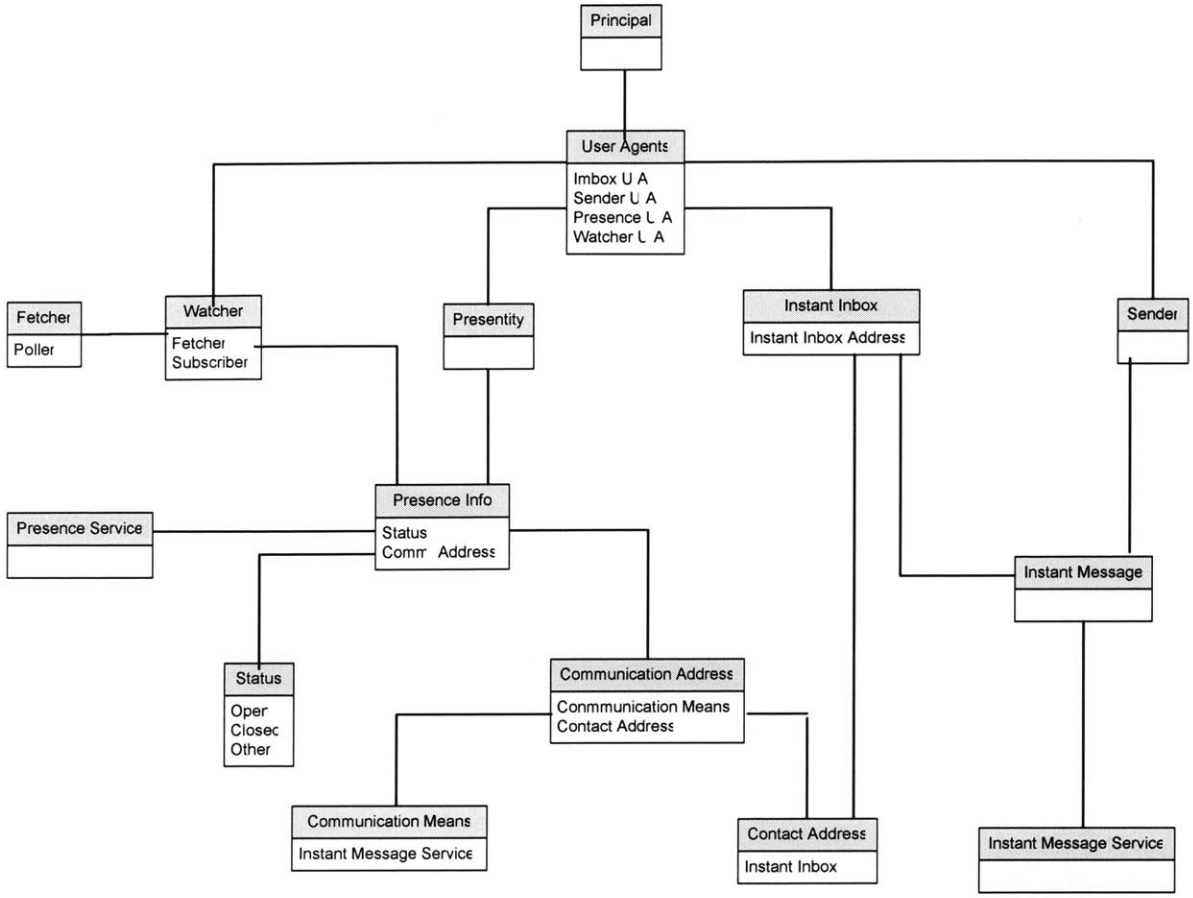

Figure 6 - 2 IM Model

The Users entity is classified as a combination of the *Presentities*[36], *Watchers*[37] and *Senders*[38] elements from the IM model. Users of IMF are classified as presentities because a user can change his/her status, providing the system with Presence Information. Users are classified as *watchers*, of type *Fetcher*[39] because users are instantly notified of any changes in status of any other user who is classified as a presentity. Finally, users are also classified as *senders* because a user provides messages (emails, instant messages, blogs and files) to be used for the services (email service, an instant messaging service, a blog posting service, and file transferring service). Also the Users entity is classified as a type of *Principal*[40] element of the IM model, since users use the system as a means of organization and communication.

The User Presence entity in the IMF data model is classified as the *Status*[41] element within the *Presence Information*[42] element in the IM model. Similar to the IM model status marker, the IMF User Presence entity allows for an Open, Closed, or Other value. In terms of i-LINK, an Open value represents a status of online, away, be right back, or busy; and Open value means that a user can accept and initiate instant chat messages and file transfers. A Closed value represents a status of offline and means that it cannot handle any instant communication, however can still send all other messages through the i-LINK client to all other accounts. Finally an Other value can be created by a user and is represented by the *Message* attribute in the User Presence entity.

There are two other elements embedded within the Presence Information element in the IM data model, the *Communication Means*[43] and the *Contact Addresses*[44], that the IMF data model also maps. In the IM data model, the *Communication Means* elements represents only instant messaging services, while in the IMF data model all communication services provided (email service, an instant messaging service, a blog posting service, and file transferring service) are classified as *Communication Means* elements. In the IM data model the *Contact Address* element represents only the *Instant Inbox Addresses*, while in the IMF data model the Addresses entity includes all addresses and is not limited to only instant messaging addresses.

The Addresses entity in the IMF data model can be mapped as the *Instant Inbox Addresses* element from the IM model because it represents how the user, in this case acting as a *principal*, can receive messages into an address account.

The AddressAccounts entity, the entity for accounts in the IMF data model, is classified as an *Instant Inboxes*[45] element from the IM model , because

it is considered a receptacle for all messages intended to be read by the users, who are considered as principals.

The MessageHeaders entity, the entity for messages in the IMF data model, is classified as an *Instant Messages* element from the IM model, because it is considered an identifiable unit of data exchange to be sent to and received from an Address Account.

6.3 IMF Data Model: Future Implementations

Address Rule Entity

This entity provides specific features for processing messages to and from a contact's address. This entity will be used for providing blocking and forwarding features. Each Contact Address record can have one or more address rules records, defined by the *AddressRuleType* attribute, which can either be block or forward. The blocking feature blocks a contact's address from sending or receiving messages. The forwarding feature forwards all messages to and from a contact's address to another specified address.

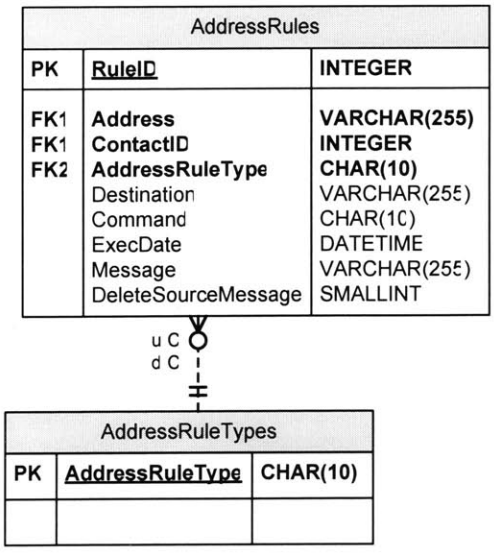

Figure 6 - 3 AddressRules Entity and Related Entity

Message Rule Entity

This entity provides specific features for sending messages. This entity will be used to provide delayed sending and event sending features. Each Message Headers record can have one more message rules records, defined by the *MessageRuleType* attribute, which can either be 'delay' or 'event'. The delayed sending feature delays the dispatch of a message by the server until a certain date. The event sending feature allows for sending messages between two dates several times. Each Message Record is described by the *TriggerDate* attribute which defines that date to send delayed message or by the *StartDate*, *EndDate*, *Recurrence* attributes which define when to send event messages and how often.

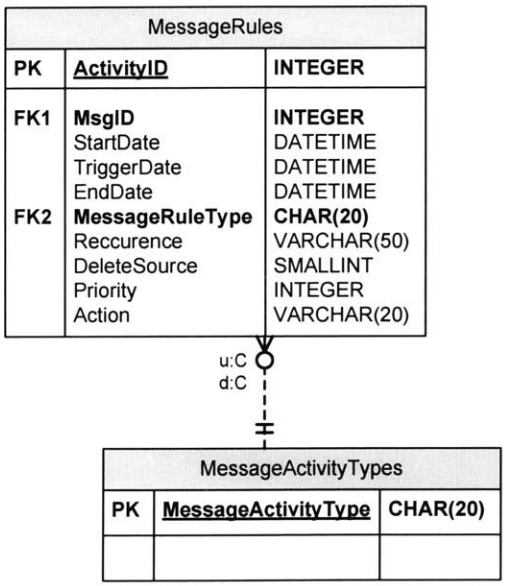

Figure 6 - 4 MessageRules Entity and Related Entity

6.4 CONCLUSION

Bearing in mind the requirements – communicative and managerial – of a software development team, a software recommendation has been made. This piece of software, the i-LINK client, has an Integrated Messaging Framework (IMF) that was

designed using a relational database model and conceptualized in reference to two existing standard models. Furthermore, i-LINK, which we have chosen to design using the .NET framework, employs an IMF implemented in IIS and SQL server.

In conclusion, while i-LINK draws upon the software currently available, its innovative design allows for an integration of several applications utilized simultaneously by members of a software development team. It therefore simplifies and improves current software options in an effort to streamline and improve communication while increasing the efficiency of project management. It further provides functionalities that are currently not available on the market and is designed to provide a greater amount of functionalities in the future.

- ENDNOTES -

[1] Day, M.; Rosenberg, J.; Sugano, H. (2000). "RFC 2778 – A Model for Presence and Instant Messaging." *Network Working Group*. Retrieved March 3, 2004 from World Wide Web: http://www.faqs.org/rfcs/rfc2778.html

[2] Stephens, Ryan K.; Plew, Ronald R. *Database Design.* Indiana: Sams Publishing, 2001. 41.

[3] Ibid.

[4] Ibid.

[5] Ibid, 45.

[6] Ibid.

[7] Ibid.

[8] Ibid, 46.

[9] *ObjectStore Overview. Annual Report.* (n.d.). Retrieved April 7, 2004 from http://www.progress.com/psc/annual_report_2003/objectstore_overview/index.ssp

[10] Stephens, Ryan K.; Plew, Ronald R. *Database Design.* Indiana: Sams Publishing, 200146.

[11] Ibid, 47.

[12] Ibid.

[13] Ibid.

[14] Ibid, 49.

[15] Ibid.

[16] Ibid.

[17] Ibid.

[18] *Microsoft Outlook Objects*. (1997). Retrieved March 3, 2004 from http://www.microsoft.com/officedev/articles/Opg/005/005.htm

[19] Microsoft Outlook does not employ a database; rather, a C++ binary data stream is used in an independent file space. The use of C++ programming is what characterizes Outlook as a object oriented model. Its data storage is not a database but a large binary file containing the various Objects discussed in this section.

[20] *Microsoft Outlook Objects*. (1997). Retrieved March 3, 2004 from http://www.microsoft.com/officedev/articles/Opg/005/005.htm

[21] Ibid.

[22]Ibid.

[23] Ibid.

[24] Ibid.

[25] Ibid.

[26] Day, M.; Rosenberg, J.; Sugano, H. (2000). "RFC 2778 – A Model for Presence and Instant Messaging." *Network Working Group*. Retrieved March 3, 2004 from World Wide Web: http://www.faqs.org/rfcs/rfc2778.html

[27] McConnell, Steve. *Rapid Development*. Washington: Microsoft Press, 1996. 312.

[28] Ibid.

[29] Dajani, Tarek. Integrated Message Framework: Strategy, Design and Implementation. Unpublished M.Eng Thesis, Massachusetts Institute of Technology, Cambridge. May, 2004.

[30] The Remote Item objects represent a remote item in the Inbox folder or another mail folder. This object is similar to the Mail Item object, but it contains only the Subject, Received, Date, Time, Sender, and Size properties and the first 256 characters of the body of the message. It gives the user who is connecting in remote mode enough information to decide whether or not to download the corresponding message.

[31] The Mail Item objects represent a mail message in the Inbox folder or another mail folder. The Mail Item is the default item object and to some extent the basic element of Outlook.

[32] The Attachments property objects represent linked or embedded objects contained in an item.

[33] The Post Item objects represent a post in a public folder that other users can browse. This object has all the characteristics of the mail message. This object is similar to the Mail Item object, except that it is posted or saved rather then sent or mailed to a recipient.

[34] The Contact Item objects represent a contact in a Contacts folder. A contact can represent any person with whom the user has any personal or professional contact.

[35] The Recipients property objects represent users or resources in Outlook; generally recipients are mail message addresses.

[36] *Presentity* (presence entity) provides *Presence Information* to the *Presence Service* to be stored and distributed.

[37] *Watcher* receives *Presence Information* about *Presentity* from the *Presence Service.* A *Watcher* can also receive *Watcher Information* about another *Watcher.*

[38] A *Sender* provides *Instant Messages* to the *Instant Message Service* for delivery.

[39] A *Fetcher* asks the *Presence Service* to forward the *Presence Information* of one or more *Presentities*.

[40] A *Principal* interacts with the system via one of several user agents.

[41] *Status* is defined by the model to have at least two state values *Open* and *Closed*, which determines the acceptance of *Instant Messages*.

[42] *Presence Information* consists of a random number of elements, called *Presence Tuples*. Each *Presence Tuple* consists of a *Status* marker, an optional *Communication Address*, and an optional *Other Presence Markup*.

[43] *Communication Means* indicates a method whereby communication can take place.

[44] *Contact Address* is a specific point of contact via some *Communication Means*.

[45] An *Instant Inbox* is a container for *Instant Messages*.

- BIBLIOGRAPHY -

- Dajani, Tarek. Integrated Message Framework: Strategy, Design and Implementation. Unpublished M.Eng Thesis, Massachusetts Institute of Technology, Cambridge. May, 2004.

- Day, M.; Rosenberg, J.; Sugano, H. (2000). "RFC 2778 – A Model for Presence and Instant Messaging." *Network Working Group*. Retrieved March 3, 2004 from World Wide Web: http://www.faqs.org/rfcs/rfc2778.html

- McConnell, Steve. *Rapid Development*. Washington: Microsoft Press, 1996.

- *Microsoft Outlook Objects*. (1997). Retrieved March 3, 2004 from http://www.microsoft.com/officedev/articles/Opg/005/005.htm

- *ObjectStore Overview. Annual Report*. (n.d.). Retrieved April 7, 2004 from http://www.progress.com/psc/annual_report_2003/objectstore_overview/index.ssp

- Stephens, Ryan K.; Plew, Ronald R. *Database Design*. Indiana: Sams Publishing, 2001