Group Messaging System

Carleton University

CSIDC Final Report

Team	*Country*	Canada
	University	Carleton University
	District	Ottawa, Ontario
	Number	1
	Mentor	Gabriel A. Wainer
		Gabriel.Wainer@sce.carleton.ca
		+1 613 520-2600 x 1957
Project	*Title*	Group Messaging System
Student 1	*Name*	Matthew Chmiel
	Email	mattchmiel@hotmail.com
	Course	Systems and Computer Engineering
	Age	22
	Gender	Male
Student 2	*Name*	Shawn French
	Email	shawnfrench@canada.com
	Course	Systems and Computer Engineering
	Age	22
	Gender	Male
Student 3	*Name*	Stephen Haber
	Email	shaber@cyberus.ca
	Course	Systems and Computer Engineering
	Age	22
	Gender	Male

I Abstract

This report documents the Group Messaging System (GMS), developed for the CSIDC 2003. The Group Messaging System is designed to replace messages left with pen and paper, by providing an easy to use system that allows leaving audio or audio/video messages to other users in the system, which can then be retrieved and reviewed from the recipient's GMS inbox. The proposal is accomplished by adding value to an existing PC or laptop. By adding an external interface in the form of a customized button pad, web camera, microphone, speakers and a fingerprint reader (optional – for security) and controlling software, the GMS allows better communication between users of a shared environment.

GMS can be beneficial to many parts of society including providing a more personal communication medium to today's busy families and in university dormitories. It also provides a more persistent and secure way of leaving messages in a professional environment, such as a hospital, and a reception area in an office.

The System Overview section of this report outlines the problem and proposed solution, performance requirements, design methodology and innovations in this system. The Implementation and Engineering Considerations section goes into the details of requirements elicitation and analysis, system design, user interface design, implementation considerations, and the testing process. Finally the Summary section gives a summary of what has been accomplished, and gives recommendations for future work.

1 System Overview

1.1 Problem Statement

In many shared environments, such as homes, hospitals or offices, the way messages are left for absent individuals is that they are written down on scrap paper, notepads or sticky notes and left somewhere for the absent person(s) to notice and read at a later time. Unfortunately, this can lead to communication problems because the messages that are left can be illegible, contain shorthand notations understood only by the message taker, or the message could even be overlooked altogether. Written messages poorly convey tone, and longer messages are cumbersome to compose. They are insecure when left in the open, which could be problematic if the messages contain sensitive information. Paper messages are not persistent unless filled properly by the message receiver. Solutions for the problems described above must be addressed by the message sender, receiver, or both, which adds weight to what should be a simple process.

1.2 Proposed Solution

We propose a solution to the current problems with the pen and paper message process of communication that is beneficial to society and is accomplished by adding value to an existing PC or laptop. By adding an external interface in the form of a customized button pad, web camera, microphone, speakers and a fingerprint reader (optional – for security) and controlling software, we propose a Group Messaging System (GMS) to allow better communication between users of a shared environment (see Figure 1 - GMS prototype).

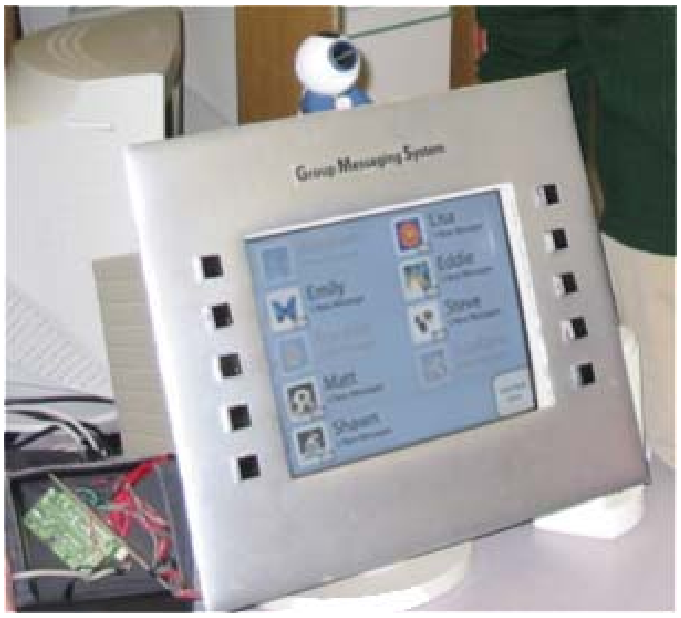


Figure 1 – Photograph of the Group Messaging System Prototype taken on March 14, 2003

The Group Messaging System is designed to replace messages left with pen and paper, by providing an easy to use system that allows leaving audio or audio/video messages to other users in the system, which can then be retrieved and reviewed from the recipient's GMS inbox. In many ways GMS provides a solution for messaging that addresses all the issues previously discussed.

GMS can be beneficial to many parts of society including providing a more personal communication medium to today's busy families and in university dormitories. It also provides a more persistent and secure way of leaving messages in a professional environment, such as a hospital, and a reception area in an office. It could be marketed as a solution to the current problems with pen and paper messages as well as a time saving device and a more personal way of communicating. GMS would be sold as a set of peripherals and software in a box in consumer electronics and computer retail stores.

1.3 Performance Requirements

GMS users can measure the performance of the software by observing the response time of the system in the time it takes to switch between screens and load and save audio and video messages. Therefore, an important goal of the system design would be to ensure all user input is acknowledged immediately with minimal observable load times.

1.4 Design Methodology

It was decided that a systematic process would help the team go from requirements to the finished product in an organized and efficient manner, as well as help create the appropriate deliverables along the way. The process described in Carleton's Software Engineering course [1] is based on the IEEE 1074 standard and describes how the design process can be best managed and documented. The design process called for the creation of a Requirements Analysis Document (RAD) and a System Design document. [2] Then, the design was implemented, and once the prototype was completed, the system was tested. This included a systematical approach to testing each feature of the GMS. Also, usability tests were conducted at the Carleton University fourth-year project poster fair using the actual GMS prototype.

1.5 Innovation

Many new products strive to improve communication on a global scale, as is the case in instant messaging. GMS focuses on improving communication among much smaller groups of individuals. GMS is composed of widely available components, so the innovation comes in the way that all these components are brought together to provide a new system for communication. As we are trying to provide an alternative to the classic pen and paper way of communicating messages, we have created a system that could convey more information, while reducing the time needed to compose the message. Video, and to a lesser extent audio messages provide a clearer and more personal way of communicating those messages. Thus, the innovation in our system is the ability of the device to deliver this functionality with a high level of usability that any child capable of using a VCR could master.

2 Implementation and Engineering Considerations

This section will detail the results at each step in the design as prescribed by the selected process. Sections 2.1 and 2.2 document the elicitation and analysis of the requirements, and section 2.4 contains information about the user interface design; the results of all three steps were formalized in the Requirements Analysis Document. Section 2.3 details all the subsystems of GMS and their responsibilities, as documented in the System Design Document. The next step was the actual system implementation. Details of the issues involved in implementing both the software and hardware of the Group Messaging System are discussed in section 2.5. Finally, section 2.6 describes how the group tested the system's software as well as its user interface. Each design document was submitted as part of the group's fourth year project at Carleton University. [10]

2.1 Requirements Elicitation

The first step of the lifecycle is very important as it defines the system from the customer's perspective. The problem statement is first described in English so that it can be read and understood by the customer, and is then modeled more formally into UML. The formal version can be used to communicate the problem to developers without any of the uncertainties that a spoken language can sometimes introduce.

2.1.1 System Requirements

Since we did not have a customer from whom requirements could be drawn, eliciting requirements could not be done in the standard fashion through face-to-face meetings. Instead, as the idea for GMS came from trying to improve communication within our own homes, it was only natural to model the system using ourselves, our families, friends, and roommates as the main customers.

Many brainstorming sessions helped narrow down the scope of the system from an all encompassing organizer complete with calendar, address book, and appointment manager, to a more specialized video messaging system. Kim Vicente, a well known Canadian cognitive engineer and professor at the University of Toronto recently made comments that mimicked exactly the group's thoughts at the time. Professor Vicente told an interviewer that "people don't want 500 features, they want about four." He went on to point out that "if you focus on what people really need, you'll find that products will be easier to use and be much more successful." [3] In other words, if users are given too many options and functions, they may feel it is far too complicated and not bother learning to use it (or buy it in the first place). This is very important in a device targeted to the average North American household since users of all ages and backgrounds should be able to use GMS easily without any formal training.

Functional Requirements

Functional requirements describe the requirements of the system from the user's perspective. From adding new users, to logging in and out, to composing messages, many of the requirements for the system were found within common messaging applications. Using email as a model, it was decided that the ability to save messages in a more permanent fashion would be helpful to the user. Users of the system might want to separate "current" messages (in the inbox) from messages that they would like to keep for long periods of time. If users knew the message was special when they first viewed it, they could save it then and not worry about accidentally deleting it from their inbox at a later date.

To gauge the response of potential customers, find any missed requirements, and determine their preferences on issues such as composing a subject, a survey was conducted. The survey was filled out by 13 subjects with an average of 3.3 people and 2.2 computers per household. Although each respondent used email, the majority of in-house messages were communicated using handwritten notes or voicemail. Almost all surveyed admitted to forgetting to pass on a message and all surveyed thought that a video messaging system would ensure messages were communicated in a more reliable fashion.

The survey results helped us decide on some functional requirements. For example, 100% of the respondents said they'd like to have the ability to compose audio only messages as well as video (with audio) messages. After being shown a sketch of the intended 10 button interface, only 6 of 13 surveyed said that they would take the time to compose subjects for their messages. Also, 85% (11 of 13) said they would like the ability to save messages in a separate area from the inbox.

The possibility that some people would not like the idea of being recorded on video arose after talking with potential users of GMS. The solution was to give users the ability to record audio-only messages as well as video messages. The GMS functional requirements are summarized below in Table 1.

Functional Requirements
Setup New User Account: add new users to the system
Default Screen: a screen where user may login that shows the # of new messages each user has
User Authentication: identify users and log them into their message inbox
Message Composing: allow users to compose and send video messages to multiple recipients
Message Storage: allow users to store received messages to view at a later time
Message Viewing: allow users to view new and stored messages that have been received
User Logout: allow users to logout at any point of their GMS session

Table 1 - Summary of Functional Requirements

Non-Functional Requirements

The survey helped us determine some non-functional requirements, such as hardware considerations (what were users prepared to pay for), and even the physical environment that people would subject the system to (the kitchen environment might entail a waterproof system). We've paid particular attention to the user interface and human factors, because we wanted to ensure even children could use the system. Also, because GMS stores personal messages onto a hard disk, security was considered. In a commercial version of GMS, some sort of encryption would be used to ensure the user's privacy. However, such a feature was not a main concern in the prototype here presented, and it could be implemented later on using external libraries.

2.1.2 Scenarios & Use Cases

All of the scenarios that might arise in the operation of the system were identified. Each scenario is a concrete informal description of a single feature of the system from the viewpoint of a single actor [4]. Scenarios were identified by stepping through all of the functional requirements from the perspective of the user and documenting the system's reaction. The complete list of was then formalized into UML use cases which covered every piece of functionality of the system and its possible deviations. Since use cases are much more formal then scenarios, particular attention was paid to describing them using application domain terms. For example, Actor names are used instead of an informal description of roles. Also, an effort was made to use the same nouns (such as Message) across all use case descriptions. This would help later on when trying to identify application objects that will be implemented using classes.

2.2 Requirements Analysis

The following is a discussion on how and why the objects in GMS were classified into entity, boundary, and control stereotypes.

2.2.1 Object Model

To describe the application in UML, nouns and verbs from the use cases were mapped into objects and messages for the object model. Using these objects and messages, a class diagram was created to model the system's main classes. Those classes were then stereotyped (using UML stereotypes) into roles of Entity, Boundary, or Control.

Having these types of objects led to models that are more resilient to change [9]. Once all objects were stereotyped, it would be easy to ensure they followed certain rules to ensure minimal impact when major changes must be made. For example, if the production version of the application were to be implemented on a different platform than the development version, the boundary objects would be greatly affected as they deal with low-level OS interfacing. However, as long as the control and entity objects don't use the boundary interface, they could withstand such a change with little to no modification.

Control Objects

A control object takes information from the boundaries (that was taken from an external actor) and reacts to it appropriately by creating new boundaries and/or persisting information into entity objects. The Group Messaging System consists of one main state-full object called a GMS_Frame. The "_Frame" is part of the name because the class inherits from a CFrameWnd, which is a Microsoft Foundation Class (MFC) for creating a frame (window) within an application. [5] This control class was modeled using an extended state pattern as described by a design pattern matrix document. [6]

The main idea for the state pattern is that instead of having the state-full object keep track of what state it's in (i.e., using switch statements everywhere) it simply has a state object that it calls to react to any input. This pattern was used because the GMS application will always be in one state or another, and depending on what state it is in, it reacts to the user input differently. Therefore, the class diagram of the control objects consists of this main "context" object (GMS_Frame), and the many implementations of the generalized state object. The Default_State is an example of an object that inherits from the generalized GMS_State object.

The GMS model called for three "complex" states that were state full themselves. The GMS_Complex_State is really the "context" object of a state pattern that has sub-states (inheriting from GMS_Sub_State) to handle events. One can think of these classes as being a state pattern implementation within the GMS_Frame state pattern. This helped model "wizard-like" states such as ComposeMessage_State which had 3 sub-states: SelectRecipients_State, RecordMessage_State, and ViewComposedMessage_State. See Figure 2 (below) for a class diagram of our control objects.

This model mapped a single boundary object (a GMS_View) to a single control object (a GMS_State). In other words, every state has a view to display data to the user, and every view has access to its state to retrieve the needed data for display. When GMS_Frame tells its current state to paint itself, the state then tells its view to do the painting. Composite states similarly pass the paint events to their current state (as with any other type of state-dependent message), and then the sub-state tells its view to paint itself.

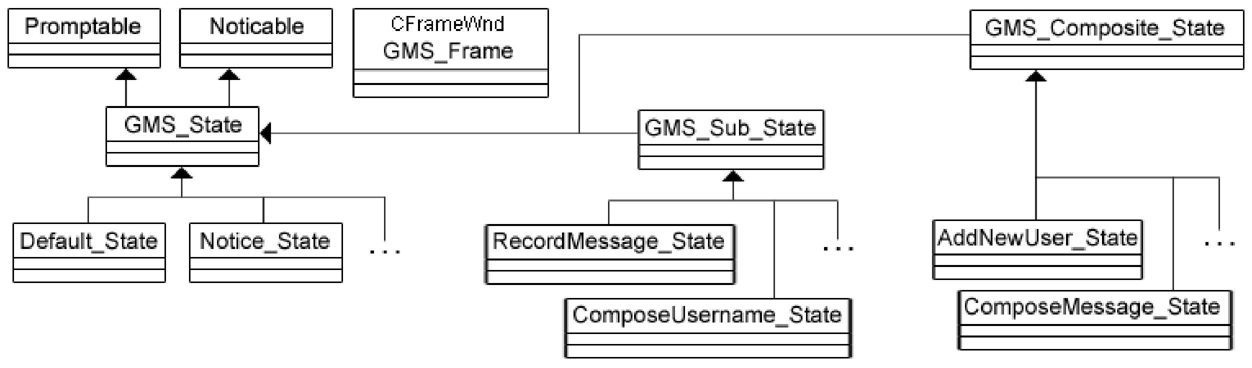


Figure 2 - Control Class Diagram

Boundary Objects

Boundary objects are those objects that communicate directly with the Actors of the system. Often they use entity data to display information to the user, and are created and controlled by controller objects. For GMS, only the objects that displayed information to the user, or captured information from the user were stereotyped as system boundaries. Therefore every class responsible for showing a screen (or a "view") to the user was modeled as Boundary objects inheriting from the GMS_View class. See Figure 3 (below) for a class diagram of our boundary objects.

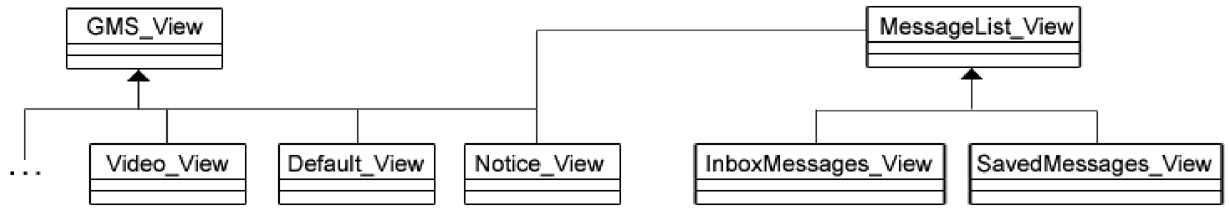


Figure 3 - Boundary Class Diagram

Entity Objects

An entity object represents the persistent data of the system. GMS requires persistent records of the user and the messages that they send and receive. The User, Message, and ReceivedMessage objects were created to handle such data storage. Apart from being able to create and store new information, each entity in GMS would also need to retrieve and modify records from the database. The figure below shows the basic class diagram of the GMS entity objects.

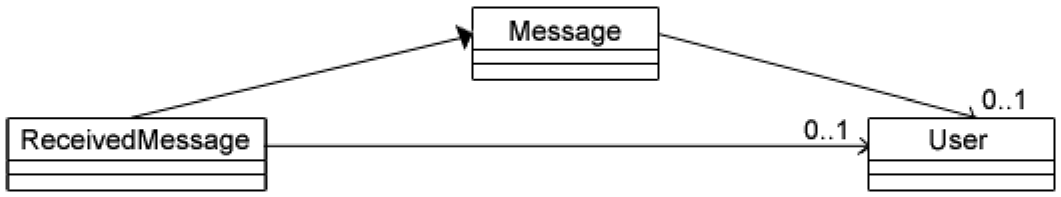


Figure 4 - Entity Class Diagram

2.2.2 Dynamic Model

Another part of analysis is the identification and modeling of interactions between objects. Also, at this point in the development process it's necessary to formalize and explain the non-trivial behavior of state-full objects, namely GMS_Frame. Both of these steps help in identifying the needed attributes for objects, as well as associations and dependencies between objects.

See Figure 8 for the statechart of a GMS_Frame object. It is a UML model of all the different simple states, composite states, and sub states the GMS_Frame can be in at any one time. As well as statcharts, the group created many sequence diagrams to describe the most complicated object interactions (such as changing states). **Error! Reference source not found.** for the sequence diagram associated with instantiating boundary objects

2.3 System Design

The GMS system can be divided into 4 main subsystems:

- Interface (User Boundary)
- Model (Entity)
- Function (Control)
- Technical Platform (UI/Database/Camera Libraries)

Each subsystem provides services to enable GMS to function as a whole. The composition of each subsystem, as well as the services that it provides, are detailed in the following sections. The following figure shows all 4 subsystems and their dependencies on one another (dotted lines):

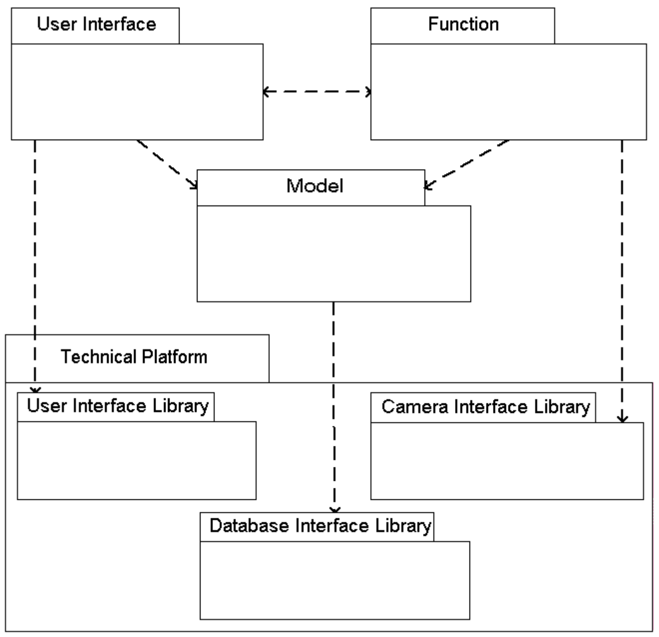


Figure 5 - GMS Subsystems and their Interdependencies

Figure 5, shows how all GMS subsystems (User Interface, Function, and Model) are dependent on the Technical Platform. Also, the figure depicts how the User Interface and Function subsystems are dependent on the public interface of the Model subsystem. Each subsystem's responsibilities, contents and dependencies are explained in the following 4 sections.

2.3.1 Function Subsystem

The function subsystem ties together all other subsystems. Its classes are responsible for the control of the entire system by reacting to user input in the appropriate fashion.

The subsystem consists of control classes that decide what boundary class should be shown depending on user input. The function subsystem is also responsible for taking user input and using it to populate objects in the model subsystem (such as a Message) to be saved into the database. For the “View” classes (in the user interface subsystem) to get and display “State” entity information, the function “State” classes will provide public getters. The getters will ensure that data will never be duplicated and that all entity information is current when being displayed.

The function subsystem will need to use the video camera library to record video and audio from the peripheral.

2.3.2 User Interface Subsystem

The user interface subsystem is responsible for creating and manipulating the graphical user interface. The subsystem provides a public interface (used by the Function subsystem) that creates and displays all GMS visual components and screens. The classes contained in this subsystem are all Boundary classes which provide a display boundary to the User.

As GMS is being implemented using Visual C++ on a windows platform, the user interface subsystem will use many components and services provided by the Technical Platform subsystem. Specifically, the user interface classes will use the Microsoft Foundation Classes (MFC) and the device drivers it provides to create and manipulate bitmaps, shapes, and windows application components (such as the media player to display video). [5]

2.3.3 Model Subsystem

The model subsystem is responsible for providing a means to create, manipulate, and store all of the persistent data used in GMS. It includes all classes stereotyped as entity classes that represent the system data. The data will consist of user and message information that is created, retrieved from, and saved to the database by classes in the function subsystem. The user interface classes will also depend on the model subsystem’s interface to display model data.

Services provided by the Database Management System (DBMS) in the Technical Platform subsystem will allow the model classes to communicate with an ODBC compliant database.

2.3.4 Technical Platform

The libraries for the user interface, database, and camera are considered to be part of the Technical Platform subsystem as the team has no control over their implementation.

The user interface library consists of all of the Microsoft Windows platform components, drivers, and utilities that the implementation of the GMS user interface will use. The Windows Media Player component and basic shape and bitmap drawing libraries will be used to display graphics to the user.

The Microsoft Visual C++ ODBC (Open Database Connectivity) implementation and its associated libraries make up the database interface library. The model subsystem implementation will use this library to add, edit, search, and delete the persistent data stored in an Microsoft Access database.

Finally, the camera library is the system developer kit (SDK) provided by the Camera manufacturer, Logitech. The SDK allows developers to add the ability to record video and audio files using a Logitech camera into their 32-bit Windows applications.

This subsystem contains only external libraries that are all platform specific. For this reason, the libraries in this subsystem are the most likely to change (or be completely replaced) if the implementation platform were to change. That implies that any code in the model, or user interface subsystems using these libraries would also need to be modified.

2.4 User Interface Design

The user interface was designed concurrently with the system design process. The functional requirements, use cases, class diagrams, and dynamic models were used to determine what screens would need to be modeled and later implemented.

Each screen was first drawn by hand, and then a digital version was created using the graphical software Adobe Illustrator 10. Although the software created the graphics based on vectors, the graphics could be easily exported as bitmaps to be used later in the GMS display algorithms. Without the exported bitmaps, all graphical screen components would have to be drawn using only basic shapes. To ensure the GUI had a high degree of usability, and that it provided a pleasant user experience, we constantly requested feedback from friends and family on the visual components and overall look (such as colors and shapes) of the design.

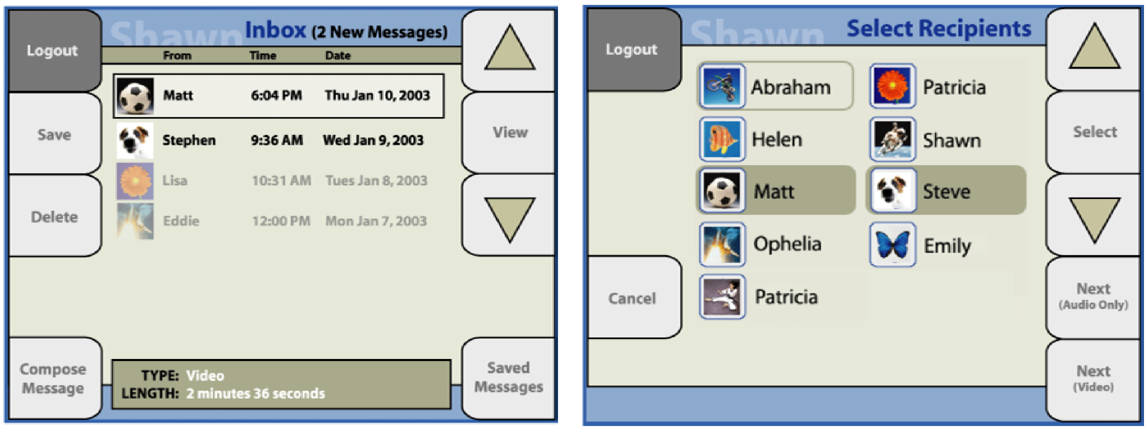
2.4.1 Screen Mock-ups

Before beginning to draw each screen, the group decided on a standard look and feel to follow when drawing screens. Once colors and a common button format were picked, and the placement of common components (such as the screen title, and the current user's name) was standardized, the group digitized the rough sketches using Illustrator.

From the time that the screens were first sketched out in rough, the notion of an easy-to-use interface was at the forefront of the group's graphical endeavors. One of the most important non-functional requirements was that the system should "be designed as an easy-to-use device that can be used by anyone with basic computer skills." This requirement is arguably the most important when considering selling a system to the general public. A product with an unusable or ugly interface will fail in the marketplace as consumers would rather purchase a more usable and "visually pleasing" product. The standardization of components and color scheme are examples of some of the measures taken to ensure a usable product.

Also, a set of usability goals was created to help guide the design process. The system should be easy for all types of users to learn without any formal training or documentation. Also, users should be able to complete their tasks (i.e., compose a message) in as little time and with as few steps as possible. The speed factor is important since our product aims to replace existing systems such as pen and paper. Finally, a very important feature of the interface is that it would not allow users to perform tasks that caused permanent changes to the system without first being prompted to do so. For example, if a user presses a button asking GMS to delete a message, the system should confirm the action before deleting the message.

The two figures are screenshots of the inbox messages and select recipients screen:



From	Time	Date
Matt	6:04 PM	Thu Jan 10, 2003
Stephen	9:36 AM	Wed Jan 9, 2003
Lisa	10:31 AM	Tues Jan 8, 2003
Eddie	12:00 PM	Mon Jan 7, 2003

Figure 6 - GMS Inbox Messages Screen (left) and Select Recipients Screen (right)

In total, GMS has 10 different screens (one for each simple state and sub state of GMS_Frame). Along with normal screens, we needed to model prompts and notices. Notices are very different from prompts as they are flashed on the screen for a few seconds to show users information about an action that the application has taken on their behalf.

2.5 Implementation Considerations

After the thorough design of the system, several implementation decisions needed to be made. The first issues that were dealt with were the selections of which external components (hardware & software) were going to be used. On the hardware side, the two main considerations were the camera, and the input buttons, as the monitor and speakers required no special consideration. A fingerprint reader was planned for authentication but the prohibitively high cost of the SDK associated with implementing it with the GMS software was beyond the budget of the competition. On the software side, the two main considerations were the implementation language and how to display video playback.

2.5.1 Hardware Implementation Considerations

Video Camera

The GMS is essentially a video recording system. In order to facilitate video and sound capture, a Logitech web camera is used. This particular type of camera was chosen since one of the group members already had the hardware and in addition, Logitech provides a generalized SDK that is designed to work with any of their cameras. [8] By using this SDK, we were able to connect to the web camera and interface with its functionality.

Connection to the camera was handled at the control class level. A camera interface object is created, as part of the objects provided by the SDK, and all commands are funneled through this interface before they are interpreted by the camera. All functionality of the camera is handled by the camera object provided as part of the SDK. This includes displaying a live video feed and recording video/audio using any compression codec installed on the system. For the purposes of this prototype the mpeg 1 video compression codec was used to capture video at a resolution of 320 x 200. Under these conditions the GMS will record video at approximately 1/3 of a megabyte per second. This codec and resolution offer some level of compression while at the same time minimizing the load on the processor to play it back. It is important to minimize system requirements since a lower common hardware denominator is best for marketing purposes.

GMS Ten-Button Interface

In order to test the ten-button interface that was designed for GMS, the group decided to create a prototype for use at the Carleton University fourth-year project poster fair. An old keyboard was taken apart and the controller circuit was removed. As can be seen in Figure 7, the controller has 9 ports on the horizontal axis (labeled as “x” ports), and 16 ports on the vertical axis (labeled as “y” ports).

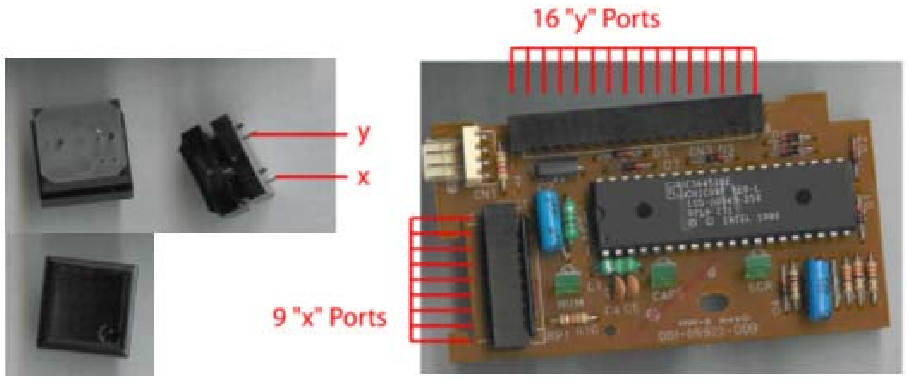


Figure 7 – The Keyboard Controller and 3 Buttons Used for the GMS Ten-Button Prototype

Every key in the keyboard is attached to one “x” port and one “y” port. When the circuit is completed (through a switch) the key’s code is sent through the controller’s PS2 connection to the PC. It was decided to use the 10 number keys of the keyboard for the ten buttons of the GMS prototype. Therefore, the x and y ports for each number key (0-9) was to found and recorded. Ten buttons (3 of which can be seen in Figure 7) were purchased and the “x” and “y” ends were soldered to a wire that connected to the proper “x” and “y” port of the controller circuit. A frame was built using a foam board and silver poster to house the buttons. Once complete, the GMS ten-button prototype was attached to a monitor and the end result was shown in Figure 1.

2.5.2 Cost of GMS Prototype Components

The cost of each piece of the GMS prototype (see Figure 1) is detailed below in Table 2.

Item	Cost
Logitech Webcam	$99.00
Silver Poster	$3.00
Foam Board	$10.00
Experimental Electronics Board	$6.00
Used Keyboard	$5.00
10 Push Buttons (10 x $1.83)	$18.30
Total	$141.30 CDN ~ $100.00 USD

Table 2 - Item Cost Description for the GMS Prototype Components

2.5.3 Software Implementation Considerations

The major consideration in the software implementation phase was what language to use. The top three languages that were considered were: Java, Visual C++ and Visual Basic. A skeleton of the GMS application was attempted in Visual Basic, but had to be abandoned because of Visual Basic's poor support for object oriented concepts. Java was a strong candidate, but had shortcomings when it came to integrating with our peripherals. Visual C++ is object oriented, and has the ability to interface with all the peripherals that we were planning to use, and would most likely have support for other peripherals in the future, so it was decided to implement the GMS prototype using Visual C++.

The next consideration was how to handle video playback. Windows Media Player was chosen because using Visual C++ meant that Windows Media Player could be used as a DirectX component in our software to handle all video and audio play back.

Function (Control Classes)

The function subsystem includes the actual GMS Windows Application class (GMS_Application) as well as the main frame called GMS_Frame. GMS_Frame is the main workhorse class that responds to user input by delegating events to its current state. GMS_State, GMS_Composite_State, GMS_Sub_State and all of their implementations make up the rest of the "Function" classes.

The function subsystem and all of its classes were built in 4 stages:

- Implementation of the GMS_Frame state machine using only the "simple" states
- Implementation of GMS_Frame transitions
- Implementation of GMS_Composite_State transitions
- Implementation of graphical interface and entity manipulation

All of GMS_Frame's states (of type GMS_State) are shown in the state diagram below.

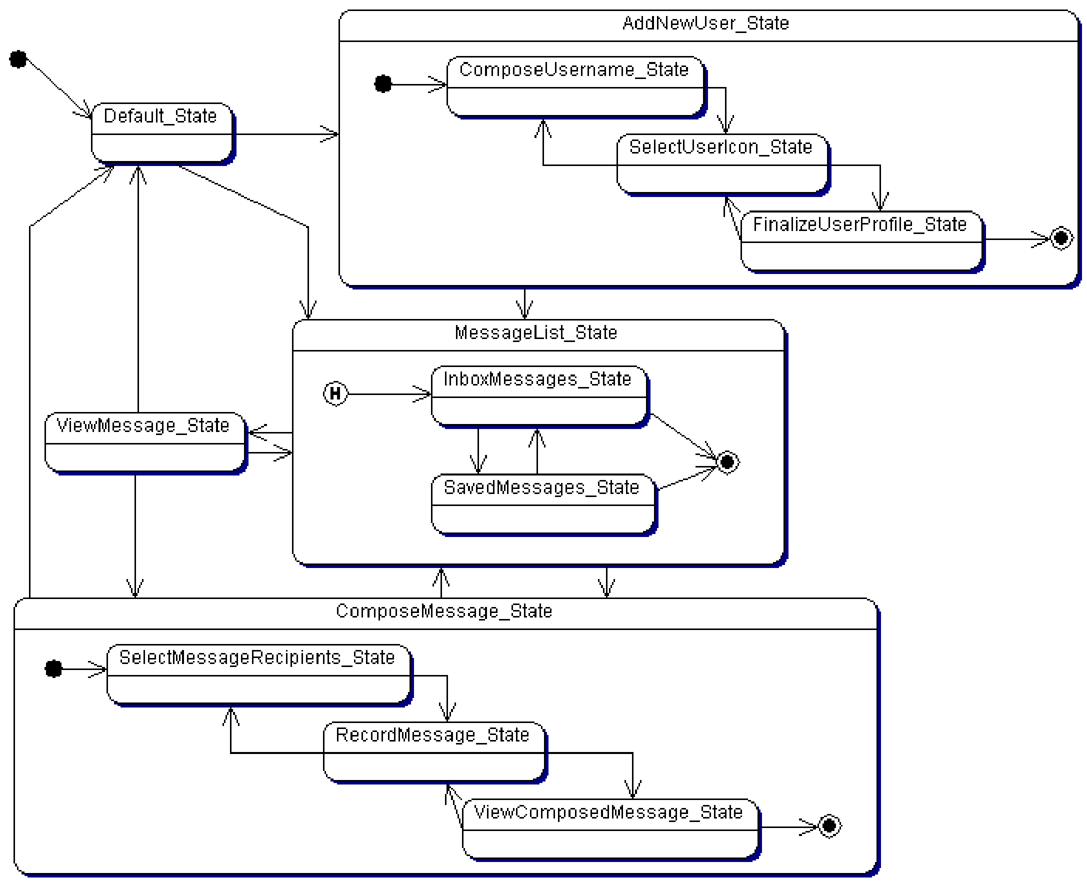


Figure 8 – Statechart Diagram for a GMS_Frame Object

GMS_Frame inherits from CFrameWnd and is the main frame of the application. It was therefore implemented as the main event handler that captured keyboard and windows "paint" events. Keyboard events are sent when a key is pressed (KeyDown event) and released (KeyUp event). A paint event is sent when the application first displays itself, and whenever the pixels need to be redrawn. Our design would have GMS_Frame catching all of these events but delegating their interpretation to the current state (of type GMS_State). As such, a GMS_State needed to implement handlers for all of the events that GMS_Frame receives. However, GMS_State objects only took care of reacting to user input, and delegated the painting related messages to its GMS_View object. This design worked well as we ended up with many very specialized classes all reacting to events that they could handle.

User Interface Display (Boundary Classes)

A large part of many of the screens on the user interface require iterating through elements on the screen, such as the letters or icons on the add user screens or the messages in the inbox. Because this ‘iterate and display’ functionality is used on many screens, two class hierarchies were developed to handle this:

DisplayObject: The DisplayObject class hierarchy is responsible for the formatting and display of data on the screen.

DisplayListWalker: The DisplayListWalker class hierarchy is responsible for iterating through corresponding DisplayObject objects and displaying them in the correct location on the screen.

Each State class in the system has a corresponding View class that is responsible for displaying the current screen. The view class takes direct responsibility for displaying the background bitmap, the title of the screen and the current user, and calls upon the DisplayListWalker of the current state (if one exists) to paint all other elements.

Video Playback (Boundary Class)

As stated in the Software Implementation Considerations section above, video playback is achieved using the Windows Media Player DirectX component. “Microsoft DirectX is an advanced suite of multimedia application programming interfaces (APIs) built into Microsoft Windows operating systems. DirectX provides a standard development platform for Windows-based PCs by enabling software developers to access specialized hardware features without having to write hardware-specific code.” [7]

The Media Player DirectX component was included in the application, and provided an API for interfacing with Media Player. Thus, a State object that requires video to be displayed instantiates a Media Player object and changes the Media Player’s state (current video file, playing, paused, etc.) though calls to its member functions. The Media Player DirectX component was easily inserted into the application and was handled by the view classes that displayed the video as an object. Controlling the video playback simply required making the appropriate function calls to the media player object.

Model (Entity Classes)

The GMS design calls for three main entity classes: Message, User and ReceivedMessage. At their most basic level, these entity objects contain attributes that mirror the fields found in their respective tables in the database. The advantage of separating these data objects and database interfacing away from the higher up control classes is that internal changes can be made to entity (message or user) structures, and to how they are stored (in a database or otherwise), without changing the way these objects are used. For instance, we could choose to remove the database and store persistent data in our own proprietary format, or we could communicate with a database on a remote server. This decreases the coupling between each layer of the system and allows for easier changes down the road.

While entity classes themselves have no control logic, they do possess basic methods that simplify their use and access to their data structures. Entity objects provide getters and setters to their attributes as well as static methods that allow additional tasks to be completed.

2.6 Testing

2.6.1 Code Testing

Extensive testing was completed on the system during and after the implementation stage. Throughout the implementation phase each group member was responsible for unit testing their respective functions and classes. When the system was complete, it was thoroughly tested by executing all use cases and then rolled out to be tested by users at the Carleton University fourth-year project poster fair. Several bugs arose in this stage, but were quickly fixed. The system ended up proving very robust in the real world user testing stage in part because the application tightly controls the input of users and therefore there are not many opportunities to crash the application.

2.6.2 Usability Testing

On March 14, 2003, Carleton University's Systems and Computer Engineering department held its annual Poster Fair. The group took the opportunity to hold usability tests to ensure people (other than its developers) thought the GMS user interface was pleasant and usable. Our usability test asked participants to complete tasks that tested all of the functionality GMS offers. As we knew some people would not have time for a long test, we had a long and a short usability test and asked passers-by if they would participate in this very important step of the development process.

Although only five tests were conducted, the results gathered offered the group more feedback than imagined. Each participant agreed that the interface was very pleasing to the eye and praised our color selection and the overall look and feel. Also, many people commented on how they would like to see a similar device in their homes one day! However, the purpose of the tests was not only to elicit praise from possible users (although we were hoping for some), but to identify any problems that users had while using GMS. The tests pointed out that some of our on-screen instructions were too vague, and some were confusing. Also, the placement of some buttons was questioned as some users pressed others thinking "it should have been there!"

The group is very satisfied with the results, and plans on making changes to the interface based on user errors and feedback.

3 Summary

The Group Messaging System is now a fully functioning, feature complete prototype (except for the fingerprint reader) as per the intended design. We began this project by brainstorming appropriate ideas for the CSDIC. After arriving at the idea of the GMS, we developed the requirements analysis document (RAD), followed by the system design document. From these solid foundations, we broke up the work of implementation into the boundary, control and entity classes and assigned each to a separate group member. Additional implementation issues were also handled at this time, including interfacing with Windows Media Player, the Logitech web camera and creating the ten button user interface. By designing the system in a modular fashion, we were able to divide the work up evenly and work in parallel to complete the system in a timely fashion.

Several enhancements can be made to the current system to improve the functionality and value of the system in the future such as including a touch screen and implementing a security model and adding an administration mode. GMS has already been designed with a UI that could be easily ported to function on a touch screen. This would simplify the interaction with the user, as the user would no longer need to focus on how the input buttons on the side of the current system align with the graphical representation of the buttons onscreen. This would be an obvious improvement, as several users tried to touch the screen of the prototype before realizing it was not a touch screen.

Although a partial security model was planned in the design phase, in the form of user authentication via a fingerprint reader, it has yet to be implemented due to the prohibitively high cost of the SDK for the fingerprint reader devices from several manufacturers. This, like the touch screen did not fit the budget set out by the competition. A second piece of the security puzzle would be to include encryption of all messages in the system. The media files would be encrypted and would incorporate the fingerprint identification information in some way. This level of security may not be necessary in home or similar applications, but it could become a requirement for adoption by businesses and governmental agencies.

The third enhancement, which would be required for a commercial release of the system, is an administration mode. The administration mode is needed to accomplish system level tasks that, depending on the application, should not be made available to every user in the system. These tasks could include: deleting a user; setting system preferences and setting the system date & time. The administration mode was left out of the design and implementation of the prototype, because it was not crucial at this stage of development.

4 References

[1] “CSIDC 2003,” *IEEE Computer Society,* http://www.computer.org/csidc/ (current March 2003).

[2] IEEE-STD-610 ANSI/IEEE Std 610.12-1990, “IEEE Standard Glossary of Software Engineering Terminology,” February 1991.

[3] Kristy Pryma, “The Man Behind the Machine,” *Transcontinental Media Inc*, http://www.itbusiness.ca/index.asp?theaction=61&lid=1&sid=51747 (current March 2003).

[4] B. Bruegge, and A.H. Dutoit, *Object-Oriented Software Engineering – Conquering Complex and Changing Systems*, Prentice Hall, Upper Saddle River, N.J., 2000.

[5] Davis Chapman, Visual C++ 6 in 21 Days, *Macmillan Computer Publishing*, Indiana, August 1998.

[6] A. Shalloway, “Design Patterns From Analysis to Design,” *NetObjectives*, http://www.netobjectives.com/dpexplained/download/dpmatrix.pdf (current March 2003).

[7] “Technology Overview - Microsoft DirectX,” *Microsoft Corporation*, http://www.microsoft.com/windows/directx/productinfo/overview/default.asp (current March 2003).

[8] “Logitech Developers Network,” *Logitech Inc.,* http://developer.logitech.com (current March 2003).

[9] L. C. Briand, "Software Engineering Course Notes," *SYSC 4800*, http://www.sce.carleton.ca/faculty/briand/teaching/94480/teaching94480.html (current March 2003).

[10] M. Chmiel, S. French, S. Haber, Group Messaging System, fourth-year project, Dept. Systems and Computer Eng., Carleton University, Ottawa, ON, 2003.