Spring 2006

Audio Conferencing Participant List Manager 3.0

Maurice Olsen
Regis University

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

Disclaimer

Audio Conferencing Participant List Manager 3.0

List Manager 3.0

Maurice Olsen

Regis University

School for Professional Studies

Master of Science in Computer Information Technology

Abstract

Global Crossing's conferencing division specializes in audio, video, and web-based collaboration. The Conference Participant List Manger 3.0 application framework (LM) allows call center operators to add, modify, delete, and report on audio conference call participant information. LM communicates with the physical conferencing bridge device through an API provided by the bridge vendor. The LM software framework consists of a thin-client, web service, platform service, and a domain definition library. The architecture implemented in the LM software project allows for ease of maintenance and speed of enhancement delivery through the use of software design patterns.

Table of Contents

Executive Summary / Overview

Company Overview and Product Scope

Global Crossing Conferencing (GCC) is a division of Global Crossing, which specializes in audio, video, and web-based conferencing for small, medium, and large businesses. GCC has partnered with other conferencing-based companies to offer web-based conferencing services. GCC's video and audio conferencing products have been completely developed in-house, with the exception of the physical audio and video-based bridge devices, which actually coordinate conference calls at a physical level. This project will focus on the audio portion of a conference call, which happens to be the most used conferencing service generating more revenue many times over web and video-based conferencing within GCC.

Audio Conferencing Bridge Definition

In order to coordinate an audio-based conference call, there is one major piece of hardware needed, an audio conferencing bridge. An audio conferencing bridge is a physical device that accepts telephony connections from many different service providers and combines all lines into a single call. Each telephony connection accepted by the bridge is associated with a corresponding software-based port. Each port, which corresponds to exactly one telephony connection is then conjoined into a single call all participants are able to listen to, as well as interact with. Figure 1 below is a diagram depicting the relationship between the end-user devices and the actual physical bridge. The telephony network is the base of the audio conferencing bridge network and telephony, cellular phones, regular phones, and VOIP supported computer systems are

allowed to connect to the physical bridge device. Once a conference call has begun, a caller can invoke bridge commands directly by use of their phone key pad. These operations include muting a call, un-muting a call, call transfer, and many others. It is also important to note that these audio conferencing bridges are capable of supporting conference calls in the hundreds of thousands and crossing international boundaries.

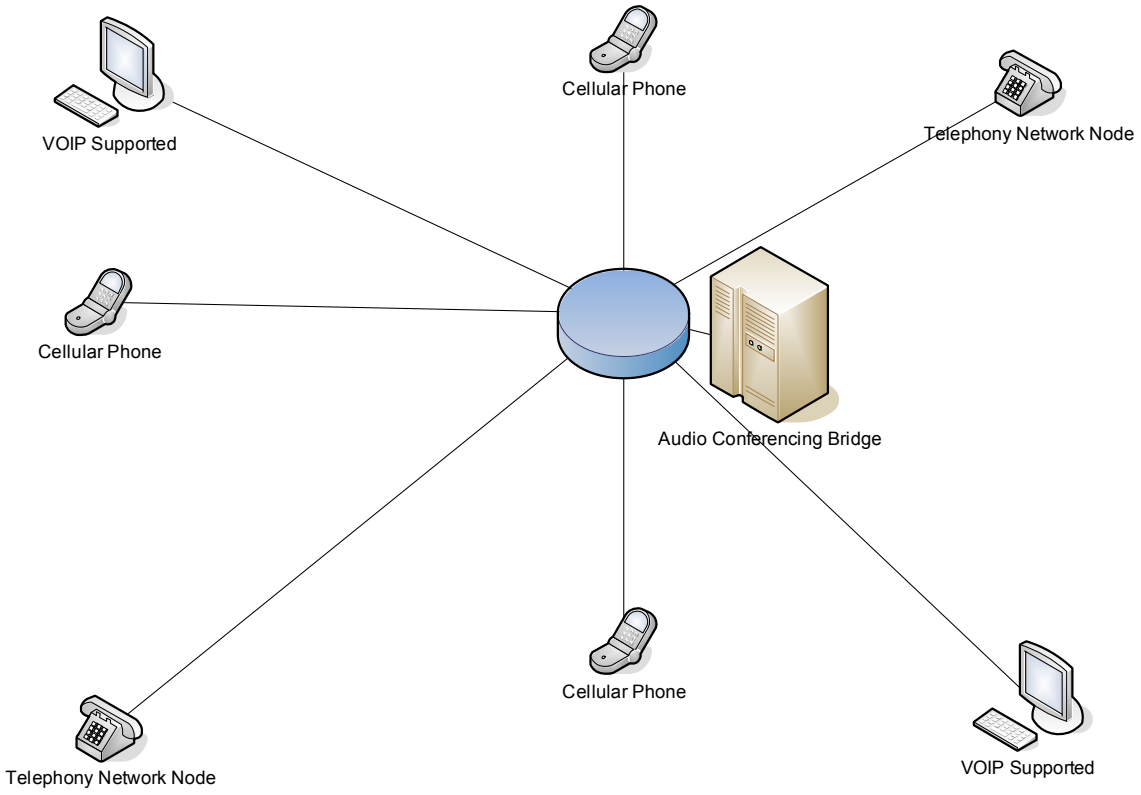


Figure1: Audio conferencing bridge network architecture.

Audio Conference Call Moderator and Participant Definitions

There are two types of callers that are accepted into an audio conferencing bridge: moderators and participants. A participant is defined by the assignment of a single caller to a software-based port on the physical bridge. Participants have the ability to mute and

un-mute their line as well as signal a moderator for questions and comments. The moderator is defined as having all qualities of a participant, plus having the ability to mute all participant lines, hand-off the call to another participant, transfer the call, keep the call open after the moderator has left the meeting, as well as signal a GCC operator to help with call coordination and advanced call functions.

An audio conference call can be restricted to specific participants as well as an open-call, which any participant could join either with or without GCC operator intervention. If the call is to be restricted, it is common for a company to send GCC a "conference call participant pre-list." This pre-list is uploaded to the bridge environment prior to the starting of a call. When a participant calls into the bridge, a GCC operator intercedes to confirm the participant is on the participant pre-list before the participant is accepted into the conference call.

Global Crossing Conferencing Operator Roles

The GCC operator is a Global Crossing call center employee that has the ability to intercede into any conference call to help facilitate the call from the time the call was created on the audio bridge and concluding with when the last participant disconnects from the audio bridge. The GCC operator is responsible for uploading conference call participant pre-lists, adding, updating, and removing participants from a conference call, and sending participant post-call reports to the end customer, which in this case is a moderator of the call in every occasion.

Project Definition and Scope

GCC will be adding a new audio conferencing bridge to its product service line-up and development is needed on existing applications to support the new audio conferencing bridge within the GCC environment. This new bridge will require new development in billing, operations, marketing, and many other facets of the organization. This thesis will focus on part one of a two-part operations development initiative. The operations part requires development in two major systems: reservations and participant call management. The reservations system manages conference call creation and maintenance. These operations include adding, updating, and removing conference calls, restricting participant requests, and making any "notes to the operator" regarding the call to take place.

The participant call management operations system is used for adding, updating, and removing participants, as well as, participant reporting, participant pre-list uploading, and participant prioritization setup. Participant prioritization setup is used to prioritize participants according to importance during the question and answer session of a conference call. The participant call management operations system will be the focus and scope of this thesis project.

Participant List Manager Application Definition

Currently, the GCC operator uses an in-house developed application to facilitate participant additions, updates, and deletions, as well as, participant pre-list uploading and call reporting; this application is called "List Manager." (LM) LM manages lists of conference call participants and allows the operator to interact with the physical bridge

directly from their personal computer. LM gives the operator the ability to add, update, and delete participants, as well as the ability to report on participant-based details after the conference call, which includes fields: connect time, disconnect time, location, company, first name, last name, and more. (Levin, 2005, p.2)

List Manager Application (Legacy Architectural Overview)

The original version of LM currently hosted in production supports a single bridge type and is a two-tier application framework, which consists of a single forms application installed on an operator personal computer and an Informix database back-end. The current LM application communicates with the audio bridges through the use of shared database tables. LM writes directly to a single database table, which then is picked up by the bridge directly; any changes made to the database tables containing participant information take effect immediately on all audio-based bridges currently available in GCC.

The two-tier architecture that was originally implemented for LM requires database specific drivers as well as database specific settings to be hosted on the operator's personal computer. The installation for the original version of LM is quite intensive for IT professionals to maintain because not only does the software have to be upgraded on all personal computers hosting the LM application, but they also have to configure software ports, database drivers and settings, as well as the LM application itself on the personal computer to host the application environment. This architecture has made it difficult to change database settings in anyway since all operator personal computers would have to be updated with the new settings and if any architectural

changes are required on either the database server or in the application environment, IT professionals have to repeat the long installation process to perform the update. There are currently over five hundred operator personal computers that host the LM application and updating all personal computers with changes has been both problematic and time consuming for all involved.

List Manager Application (Proposed Architectural Overview)

A business need has arisen to add an additional bridge type to LM. Along with the additional bridge, management wants to integrate this new application into an existing .NET framework and completely do away with the stand-alone Delphi version of LM. The new features applying to all bridge types supported will include enhanced reporting capabilities, mass participant upload capabilities, and increased participant information available to the operator through the LM GUI.

The new LM application, titled "List Manager 3.0," is a complete application rewrite and will use a Model-View-Controller (MVC) implementation of n-tier architecture for application design, but will have to keep the same look and feel as the original version of LM to cut down on training requirements for the operators during release. (Levin, 2005, p.4)

An architecture is needed that would provide ease of future application enhancements as well as database server changes. In order to facilitate these requirements, a thin-client will be developed containing only view and validation code, while the bulk of the business model will be hosted inside of a web service on a central server. The web service will accept all thin-client connections simultaneously and be able

to conduct database persisting as well as direct bridge communication through the new bridge vendor's API. Abstracting the business model from the actual installed client on the operator's personal computer will allow for database settings changes and additional bridge types to be added without having to modify the operator's installed version of LM. The only conditions that will require a new operator client installation is the addition or change to the operator GUI.

In addition to the thin-client and web service, there is also a need to add a server process hosted as a separate application from the web service that calls methods out of the web service in a timed fashion to do audio-bridge and local database synchronization. Although the currently supported bridge type talks directly to the same database tables as the original version of LM, this is no longer completely true with the new Pactolus bridge. (Pactolus is the name of the manufacturer of the new physical bridge type being added to GCC.) Synchronization of participant lists will have to take place between GCC's local conferencing database and the actual bridge. This synchronization process, hosted as a server process, will look-up all conference calls being conducted in a given time period, when the call is started by a moderator, the synchronization server process will run every two minutes, constantly taking participants added to the bridge and synchronizing these participants with GCC's local database. Although you would think that LM would be able to update both the bridge and the database during the LM session, this is not the case. In addition to the LM application, the GCC operator also has an additional application, which can modify participants directly on the bridge and this application is provided by the bridge vendor to conduct calls when the call begins, so LM

will not be the only point of change for participant lists, thus requiring a synchronization process to ensure that the data is consistent across the complete architecture.

Obstacles / Constraints

There are a few notable constraints that will need to be reviewed as part of the application design. According to the requirements documentation provided, the LM application must keep the same look and feel as the original LM application to cut down on training costs and operator confusion.

There must be a complete abstraction of business logic from the forms installed client to be hosted on the operator's PC. Ensuring that the client no longer possesses business logic will ensure that IT professionals and software engineers will be able to update and upgrade the application framework and the database framework without having to push out another client installation to all operator personal computers.

In addition to abstracting business logic from the operator personal computer installation, a suitable installer compatible with Microsoft Systems Management Server 2.0. (SMS) is also required. SMS is an enterprise application, which allows IT professionals to push out personal computer installations remotely as well as maintain a Microsoft Windows network.

The new LM application must be written using .NET technologies and must be capable of integrating into an existing .NET application framework by use of a .NET assembly plug-in. This requirement will require two separate builds of LM, a standalone application version to be installed on a personal computer, as well as a plug-in version, which will be installed into an existing .NET application framework hosted on a Citrix

Metaframe server. The .NET plug-in version of LM must give the end-user the capability to print and save reports back to the calling computer running the Citrix client.

There are two notable obstacles involved with this project that has to do with application performance and real-time data synchronization. Since a synchronization process will be necessary to provide data consistency throughout the environment, mainly between the actual bridge and an Informix database, there is a question regarding the frequency of the updates and how that will impact an operator's ability to conduct a real-time conference call. The other notable obstacle is application performance and scalability in a large environment. The web service will have to support up to seventy-five simultaneous requests. Load testing will have to be performed and performance statistics confirmed with a business analyst and a GCC operations call center supervisor before application sign-off will be given.

Summary

Global Crossing's conferencing division (GCC) specializes in hosting and conducting conference calls for small, medium, and large businesses. GCC develops software to conduct and mange these conference calls, which take place on a physical device called a conferencing bridge.

A business need has arisen, which will require the participant list manager application (LM), an application that maintains conference call participant information, to be rewritten to integrate with existing .NET systems, support additional features, as well as support the new Pactolus bridge type. The new application architecture will be

designed using an MVC implementation of n-tier architecture to facilitate easy application upgrades and changes.

Chapter 1: Research and Methodology

Reference and Research Approach

The annotated bibliography referenced at the end of this text, can be broken down into two major categories of interest: architecture and implementation. In the sections to follow, both categories will be examined and corresponding reference support for methodologies and implementations presented and implemented within this project. The remainder of this chapter will focus on the overall architectural methodologies implemented throughout the List Manager .NET software solution. The specific methodologies and implementations used per .NET software project are examined in detail in the chapters to follow.

LM .NET / Citrix Requirement

Global Crossing Conferencing (GCC) has been migrating legacy applications to .NET component applications as the possibility arises; the migration is almost always a full application rewrite. GCC has found this migration important in delivering a higher level of service to its end-users. GCC creates this higher level of service with a .NET forms application, which resides on a Citrix Metaframe server. The Citrix Metaframe server allows for multiple end-users from various physical sites to use a single entry point to access multiple application instances. The .NET forms application hosted on the Citrix Metaframe server is called the “Ready-Access Admin GUI.” This front-end application allows for other .NET enabled applications to be plugged into its infrastructure through

.NET assemblies. Through this requirement, GCC has defined the technology to be used in this project, ".NET". (Levin, 2005)

N-tier/MVC Architecture Overview

Architectural research was performed to implement software design patterns within an n-tier application framework. The "Model-View-Controller" (MVC) design pattern allows software to be designed and architected to fit the n-tier development methodology, which abstracts user interface logic from business process, "This is a fundamental design pattern for the separation of user interface logic from business logic." (Microsoft Corporation, 2004)

"N-Tier architecture refers to the architecture of an application that has at least 3 "logical" layers -- or parts -- that are separate." (Yang, 2001) An n-tier architected application framework consists of many tiers of software abstraction and is unlimited in the tiers that could be defined. This logical abstraction of business concepts into tiers promotes reusability of code components as well as ease of maintenance; Figure 2 below is a diagram depicting a common n-tier framework implementation using MVC.

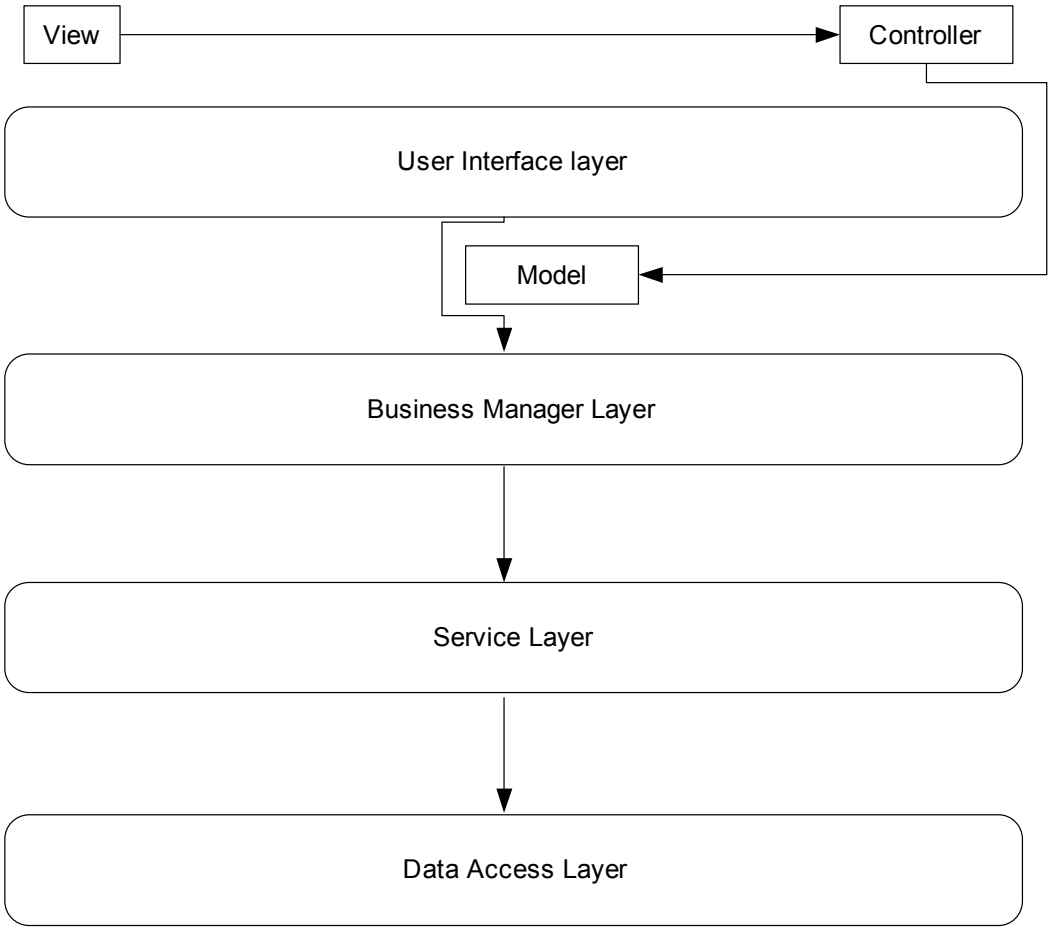


Figure 2: N-tier software architecture / MVC architecture overlap.

Notice that the "view" and "controller" portion of the MVC design pattern are part of the user interface tier of an n-tier methodology and that the "model" component of MVC contains all subsequent tiers thereafter. There is a clear separation between the user interface tier, business tier, service tier, and data access tier in that each tier's collection of class components only has visibility to the direct subsequent descendant tier components, but has no knowledge of any components in the tiers above it or below it. Ultimately, the "user interface" tier only has knowledge of the "business manager" tier components and the "business manager" tier components only have knowledge of the "service tier" components and this pattern is replicated throughout all tiers following the

same rules; this methodology promotes class decoupling. The following sections in this text will examine the purpose of each tier in this basic n-tier framework. N-tier architecture represents an unset number of possible tiers in an application system, but the MVC pattern focuses largely on the n-tier implementation presented and so does this project, so the scope of the layers defined in this methodology will be limited to what was actually implemented in LM.

The "user interface tier" of n-tier architecture contains software components used to present and validate views to the end user of the system. In the MVC design methodology, the "view" and "controller" components are located within the "user interface tier" of n-tier architecture. End-users will submit data through end-user graphical user interfaces, which are then translated into "view" classes in the underlying application code. The "view" component then passes the data back to a corresponding "controller" component for data validation. If the data is valid, control is then passed to the "model" tier in MVC for business processing and data persisting.

The "model" portion of the MVC design pattern is used for performing business process related tasks, such as data persisting and manipulation. The "model" in MVC is partitioned into three subsequent layers: manager layer, service layer, and data access layer. (Yang, 2001)

The component tier in MVC that actually does the business processing is the "manager tier," which exists within the "model tier" of MVC. The "manager tier" accepts

domain tier objects or simple data types as arguments from the controller tier and makes programmatic business decisions based on these incoming dynamic parameters.

The "service" tier of n-tier architecture is often times the layer that performs the actual processing from within the application, with the exception of data access, which is delegated to the next and final layer in our n-tier architecture, the "data access" layer. The "service" tier makes decisions on how and where data is to be persisted and also often times will operate on that data before returning the results back up to the end-user of the application.

The last and final tier this text will examine is the "data access" tier. The "data access" tier handles the formulations of queries and the direct communication with the database back-end. The "data access" layer is responsible for formulating requests to be sent to the database, receiving those requests, and handling any errors that might occur while the request is being processed.

LM .NET Solutions / Projects

Microsoft Visual Studio 2005.NET, the IDE that was used to develop the LM application components, allows a developer to have a single solution, which is a logical ordering of code that may contain application projects beneath it. The LM application framework was created using a single .NET Solution, called "ListManager." The "ListManager" solution contains four separate application projects:

1. List Manager Thin-Client (LMTC): Thin-client end-user application.

2. List Manager Domain Library (LMDL): .NET dynamic link library containing class definitions for the application domain.
3. List Manager Web Service (LMWS): ASP.NET web service application to host the LM API.
4. List Manager System Service (LMSS): System service application that runs according to set time intervals that calls LMWS methods to synchronize back-end data.

Each .NET project hosted under a single .NET solution will create its own binary executables or dynamic link libraries to be used and defined by its own memory space and physical files.

Ready-Access Admin GUI Framework / .NET Assemblies

.NET Assemblies are flat files, which are represented with a ".dll" file extension. (Jillellamudi, 2004) The ".dll" file extension stands for "dynamic link library" (DLL) and this file type is Microsoft platform specific. Unlike traditional dynamic link libraries, .NET assembly versions of the .dll do not have to be registered within the windows registry for use; these libraries are often times directly referenced by the application or hosted in the .NET Global Assembly Cache (GAC). The Microsoft .NET Framework allows for entire forms, web, service, and API-based applications to be fully or partly contained within a .NET assembly file; this technology allows a developer to write code in a language such as C#, but embed the application into a larger framework that was created in Visual Basic or C++. (Dietrich, 2004)

Tiers / Software Namespaces

Methodology tiers are translated from the design phase to the implementation or coding phase of the project with the use of logical software namespaces; namespaces are a logical ordering of code components. (Jones, 2003) Each tier implemented in the software design process of a software project will have a corresponding namespace definition, which implements the functionality of the corresponding classes that are referenced by it. Software namespaces in general contain class definitions and other namespace definitions within a software application. Namespaces can contain other namespaces in so that a developer could model the "model" namespace to correspond with the "model" tier in MVC. This "model" namespace would then contain subsequent namespaces that would relate to a "manager" namespace, "service" namespace, and "data access" namespace respectively.

Contribution to the Field

The LM application system and this thesis project contributes to the field of computer science in that this project promotes and implements coding design methodologies, which are current and aid in promoting object-oriented benefits, such as, code re-use, ease of maintenance, and scalability of the application. The LM project is a leading example of how a software application system should be designed in today's advancing technological world. Not only does the LM application system boast the latest in methodologies implemented, but this project also uses advanced concepts, such as socket-level programming, dynamic class loading, and web service API hosting.

Summary

Research was conducted in a systematic way ensuring the use of credible sources and more then a single source on related subject matter to confirm the methodology represented. The annotated bibliography section of this project follows two major themes: architecture and implementation. Architectural research was conducted on advanced methodologies, such as design patterns and n-tier methodology using MVC. Implementation research was conducted on advanced concepts such as socket-level programming.

GCC has placed a requirement on this project that enforces the use of Microsoft .NET development technologies to produce .net assembly-based applications, which can be plugged into a larger .NET hosting application framework. (Levin, 2005) This framework allows an end-user to enter this larger system from a single entry point and navigate the end-user through a plethora of applications that all have separate memory spaces, but appear to be operating as a single application to the end-user of the system.

The LM application system was designed using an MVC implementation of n-tier architecture. N-tier architecture is the application design methodology, which abstracts software concepts into separate logical layers implemented by logical software namespaces.

LM will contribute to the field of computer science with the implementation of good design methodologies and advanced concepts used. Good methodologies create stable, scalable, and maintainable application systems.

Chapter 2: “List Manager Domain Library” Software Project

“List Manager Domain Library” (LMDL) Project Overview

The “List Manager Domain Library” (LMDL) project defines application domain properties-based classes, which are classes that define informative objects about the application being created. These objects are used to communicate business entities down through the tiers or software-based packages in our MVC application framework. In a legacy 2-tier application system methodology, it was common to lump the application domain classes into the software application to use the classes directly. In this project, there was an opportunity to abstract the application domain properties classes out of the software project and place them into their own physical and logical library. This library then allows multiple external software applications to now use the same application domain class definitions; this library helps to facilitate cross-application communications through technologies like web services.

LMDL defines twelve separate classes and two additional software namespaces to be used within the LM application framework, but this text will focus on two core class definitions that define the majority of data used within the LM framework and a domain exception class definition, which facilitates exception handling between the thin-client, web service, and system service. Separating the application domain class definitions into its own .NET assembly allows a developer to import these classes into any other external software project that could use an application domain class definition of a moderator, participant, or any of the other ten domain class definitions available in the LMDL. This methodology and approach supports code-reuse in that these class definitions at the very

least would not have to be created in both the web service and thin-client application software projects.

The LM application framework defines application domain classes, "CResvChairInfo" and "CResvPartInfo." The "CResvChairInfo" class defines private members and public methods that correspond to conference call moderator specific details and "CResvPartInfo" corresponds to conference call participant specific details. Each one of these classes define members that correspond with details that a participant or moderator would have in regards to a conference call, such as, first name, last name, address, priority, reservation numbers, etc. In essence, LMDL will allow the passing of one complex object that contains all information that corresponds to a moderator or participant respectively and this information can then be used throughout the entire framework without any further data manipulation or parameter passing.

Figure 3 below is a diagram depicting all the private members to both the "CResvChairInfo" class and the "CResvPartInfo" class; these two classes are the heart of the List Manager solution and are used in every project for object-to-object communications.

```
public class CResvChairInfo
{
   private Int32 confStatus = -1;
   private Int32 serviceProviderID = -1;
   private string confStatusDesc = "";
   private string startDate = "";
   private string endDate = "";
   private string startTime = "";
   private string endTime = "";
   private Int32 compId = -1;
   private string compName = "";
   private string chairName = "";
   private string chairPhone = "";
```

```
public class CResvPartInfo
{
      private string moderator = "";
      private string companyName = "";
      private string emailAddress = "";
      private string primaryPhone = "";
      private string faxNumber = "";
      private string customField1 = "";
      private string customField2 = "";
      private string customField3 = "";
      private string customField4 = "";
      private string otherInfoNote = "";
      private string numPeopleInRoom = "";
      private char attended = 'N';
      private Int32 partId = -1;
      private Int64 pacPartId = -1;
      private string surName = "";
      private string firstName = "";
      private string lastName = "";
      private string partType = "";
      private string partTitle= "";
      private string location = "";
      private string phoneNo = "";
      private Int16 priority = 9;
      private string connectTime = "";
      private string disconnectTime = "";
```

Figure 3: "CResvChairInfo" and "CResvPartInfo" domain class excerpts

Take note that the participant fields and chair person fields referenced in the LM requirements documentation (Levin, 2005) mirror many of the fields of the "CResvPartInfo" class and the "CResvChairInfo" class respectively defining the data, which our application will be displaying, manipulating, and storing. One of the great benefits to object-oriented design and development is the ability to mirror business processes during implementation. This relationship between the requirements and these two data classes clearly depicts the business model mirroring that is being transferred over into the implementation phase of this software project.

In addition to the two core class definitions that define the majority of data being passed, manipulated, and persisted in the LM application system, there is also a class

definition used for handling exceptions and transmitting these exceptions from the web service to the thin-client and from the web service to the system service, called "CListManagerSoapException." The "CListManagerSoapException" class definition accepts a string of arguments representing the error message, error number, error source, and whether to log the error or not; take note that instead of having multiple logging entries in the exception handler portions of code, the exception is responsible for the logging, thus requiring only a single block of code to perform error logging throughout the application framework. (Application logging is done on the web service tier and is not performed on any applications connecting to the web service because the exception is generated from within the web service itself; additional exception handling has been implemented to handle exceptions that occur from the calling clients.) Figure 4 below is a sample code excerpt of the "CListManagerSoapException" class depicting the logging method call; the List Manager Web Service uses the open source Apache product, "Log4NET," to facilitate the logging feature of the web service.

```
public class CListManagerSoapException: SoapException
{
   public CListManagerSoapException() { }

   public SoapException raiseSoapException(string classGenErrorSource, string errMsg, int errNum,
string errSource, bool log, bool serverFault)

   {
      XmlQualifiedName faultCodeLocation = null;

      switch (serverFault)
      {
         case false:
            faultCodeLocation = SoapException.ClientFaultCode;
            break;
         case true:
            faultCodeLocation = SoapException.ServerFaultCode;
            if (log == true)
               CLoggingServiceHelper.writeLogMessage(errMsg, "ERROR", classGenErrorSource);
            break;
      }
```

Figure 4: "CListManagerSoapException" code excerpt depicting logging calls and fault choices.

```
2006-10-20 13:03:07,054 [1] WS-CResvPactolusBridgeConnDao - LOG ENTRY STARTED**********
2006-10-20 13:03:07,064 [1] WS-CResvPactolusBridgeConnDao - AMS\mo003024 generated the following log entry.
2006-10-20 13:03:07,064 [1] WS-CResvPactolusBridgeConnDao - A connection attempt failed because the connected party
did not properly respond after a period of time, or established connection failed because connected host has failed to respond
2006-10-20 13:03:07,064 [1] WS-CResvPactolusBridgeConnDao - LOG ENTRY ENDED**********
```

Figure 5: "CListManagerSoapException" generates the above log message recorded in a flat text file.

The above "CListManagerSoapException" code excerpt generates the above log file as depicted in figure 5. When a Soap exception is thrown, the exception could have been generated on the calling client or on the hosting web service. In figure 4, the "ServerFault" boolean parameter captures the source of the exception, whether client or server generated. If "ServerFault" is "true,", indicating a server-based web service error, then logging is performed, otherwise the exception is not recorded to the log file, but still generated and displayed ultimately as an end-user calling client dialog.

"List Manager Domain Library" Project Methodology Implemented

Each software project within the LM application solution conforms to its own implementation of MVC methodology and this implementation is defined both by logical separation of code into software namespaces as well as the decoupling of objects to conform to the methodology as discussed in chapter 1 of this text.

LMDL produces a single dynamic link library file, which extends an application's capabilities through the use of a public API offered by the compiled version of the LMDL. This library's sole purpose is to provide domain definition classes to other software applications. The domain definition classes do not process data, so a full MVC

implementation for the LMDL doesn't make sense; a standard namespace structure is used to simply break-up the domain application class definitions into logical parts.

Summary

The "List Manager Domain Library" (LMDL) project defines application domain properties-based classes, which are classes that contain informative information about the software application domain. LMDL ultimately produces a single dynamic link library, which can be accessed by multiple external software projects to minimize duplicate coding in all tiers of the LM application framework.

LMDL contains properties-based classes and does not contain view or business-based logic, so using the MVC design pattern doesn't make much sense. MVC is used to abstract business logic from view logic and neither exists in this software project.

Chapter 3: "List Manager Web Service" Software Project

"List Manager Web Service" (LMWS) Project Overview

The "List Manager Web Service" (LMWS) software project handles all the business processing for the "List Manager .NET solution." LMWS is an ASP.NET web service that offers a public API to calling clients. This public API offers methods to persist data well as retrieve data from different types of data sources. All the data for the LM application system is built and transformed within this application process and then sent back to the calling client in the form of a SOAP response.

The legacy version of LM, now being replaced by this current project, uses a 2-tier methodological approach to application development. All the business logic, view, and validation code was placed in a thick-client and this thick-client communicated with the database back-end directly. The legacy version of LM required customized IBM Informix drivers to be installed on all clients to run the legacy LM application. This configuration overhead produced numerous headaches for information technologists attempting to maintain the environment either from a software perspective or from a systems deployment perspective.

The method used to combat the custom configuration necessary on all clients running the LM GUI software and make the software more maintainable was to abstract the business logic out of the thick-client solution and place it into a separate web service application hosted in a centralized environment, but available to a decentralized end-user group. The methodology chosen to abstract the business logic out of the thick client

enables a software engineer to add additional supporting bridge types to the web service code and not ever have to modify the new thin-client installed within the Citrix environment as well as on end-user personal computers. The next sections of this text will focus on the methodology used to create LMWS as well as advanced concepts implemented unique to this specific project.

"List Manager Web Service" Project Design Methodology

LMWS implements many layers of the MVC design pattern, but still customized for this specific project's purpose. A web service does not have a "view" and therefore the "view" pattern in MVC has been omitted out of this software design. There is a single service file, which acts as the controller of the web service and this service file accepts requests from calling clients and funnels those requests through the business model layer of the application. Figure 6 shown below depicts the architectural breakdown of the LMWS and the relationship between the LMWS and the List Manager Thin-Client.

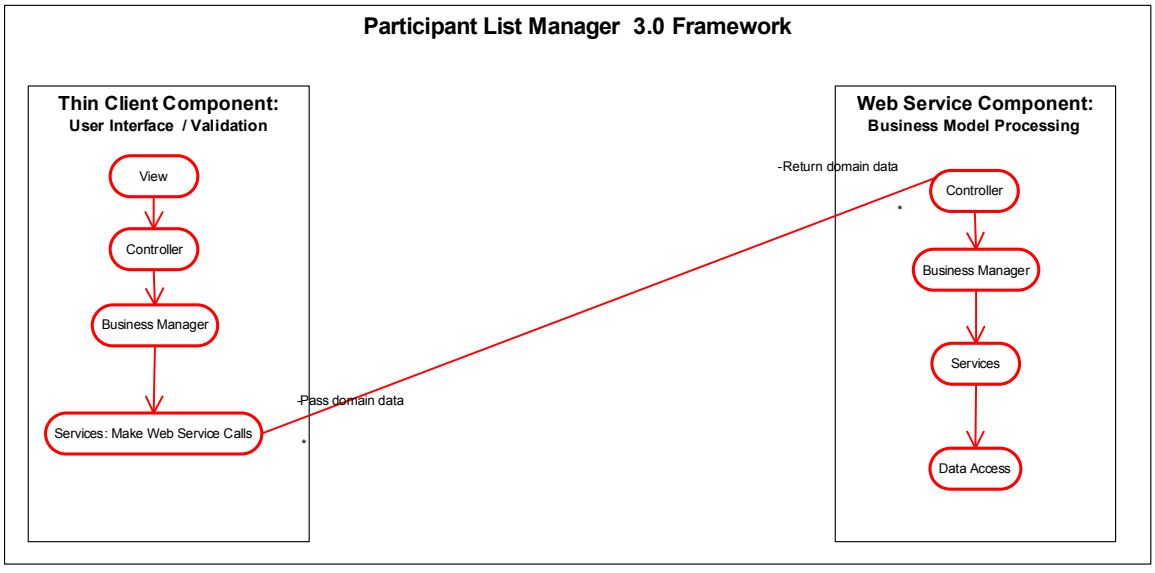


Figure 6: LM web service and thin-client n-tier methodology.

Maintainability and ease of updating the List Manager .NET solution in any one of its components is a key concept for this project because the new bridge type vendor will be releasing bug fixes and new API methods on a frequency of about once a month. Since API changes will be frequent, it is important to allow for application updates to be added as easily as possible. Physical and logical n-tier architecture breaks down the application into smaller self-contained pieces, so that software engineering can update components of the application and not have to update the entire application itself. The "Factory" design pattern adds another level to this concept by allowing components within the application framework to be updatable without having to deploy an entirely new version of code. Figure 7 below depicts the LMWS class diagram broken down into layers; take not of the factory design pattern and interface design patterns used. The following section illustrates the use of the "Factory" design pattern.

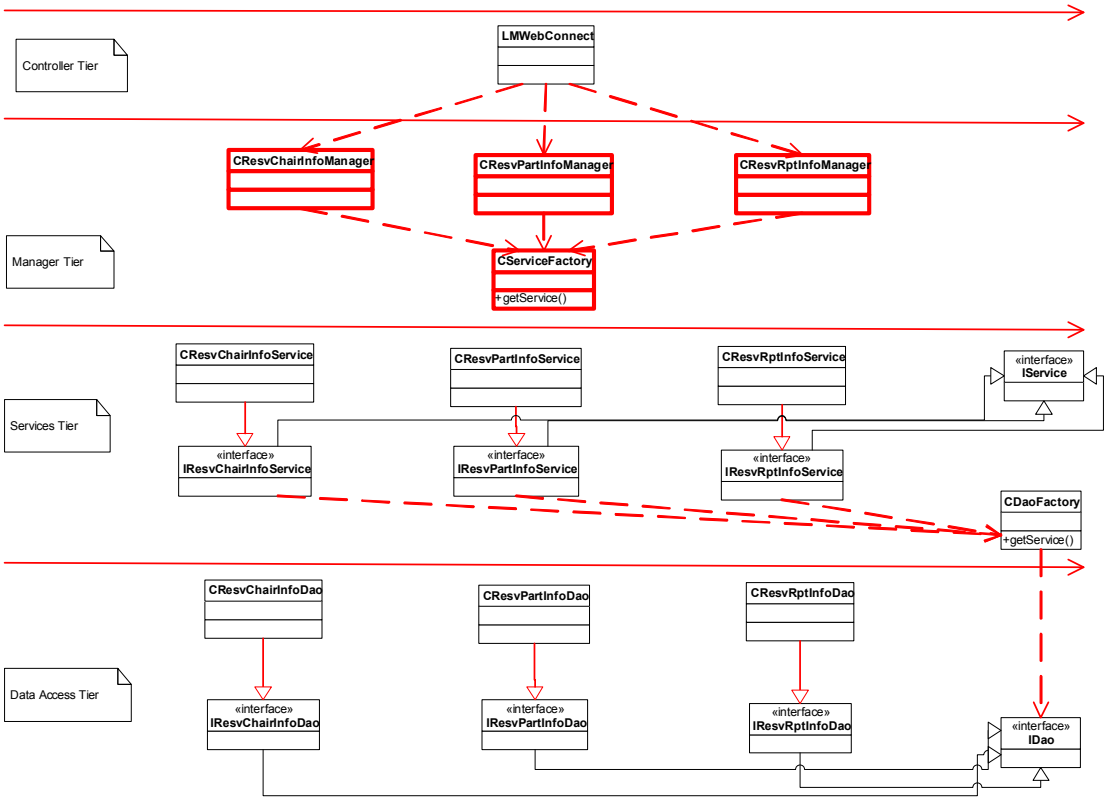


Figure 7: LM web service class diagram

Factory Design Pattern

The implementation of the MVC methodology, as depicted in figure 7, creates a level of abstraction and class decoupling so that software engineering can easily add new functionality to the web service as well as maintain existing code easily. In order to further promote ease of maintenance and add the ability to add further software components later to the application without having to do a full regression test, the LMWS project implements the "factory design pattern." (FDP)

FDP is a design pattern that allows for "dynamic class loading." (Trott & Shalloway, 2002) Dynamic class loading is a coding technique used to dynamically load

classes at runtime. The benefit to this feature is the ability to provide the application with additional software component definitions through a properties file, which is read by the application during runtime. In essence, a developer could add new functionality and replace existing functionality of the application without having to recompile the entire web service application itself. This gives the certification department the ability to test certain components of an application without having to regression test the entire application every time an application change is made or feature added.

The LMWS implements the factory pattern at the "service" and "data access" tiers. The majority of business model processing is done at the "service" and "data access" tiers of the application. If software engineering wanted to create the ability to add an additional audio conferencing bridge type later, all a developer would have to do is replace or add an existing "service" tier class definition with a new definition that has implementation code to call the new "data access" class, which then communicates with the new audio conferencing bridge.

The LMWS combines different design patterns together to produce the appropriate factory behavior specific to this application. The Interface design pattern is used to abstract the public method definitions used in object-to-object communication. (Trott & Shalloway, 2002) In the case of LMWS, the factory classes return an interface to the class invoking instantiation through the factory. By creating interfaces to our "service" and "data access" classes, we create a legal contract between a class and its corresponding interface. This contract determines what return types are allowed from a class's public methods as well as what type of parameters can be passed into a class's

corresponding public methods. This enables a developer to change a specific class's implementation and as long as the developer doesn't violate the contract between a class definition and its corresponding interface, there will not be any concern about affecting other classes, which instantiate and use this class's method definitions.

Figure 8 below illustrates the use of FDP in LMWS; this is one of two factory classes available in the factory namespace and each factory corresponds to a specific tier in our n-tier architecture. The "CServiceFactory" class and the "CDaoFacotory" class service the "service" and "data access" tiers respectively. Each factory returns a custom generic interface, which is inherited by all interfaces in that corresponding tier.

In figure 8 below, the "CServiceFactory" class accepts a string type, representing the class name to dynamically load at runtime. The "getService" method returns a generic "IService" interface; all interfaces created in the "Service" tier will need to inherit the generic "IService" interface, so that the factory will be able to return an object of any type from that respective tier.

```
namespace ListManagerWS.Model.Factory
{
   public class CServiceFactory
   {
      private static CServiceFactory f = new CServiceFactory();

      public static CServiceFactory newInstance()
      {
         return f;
      }

      public IService getService(string serviceName)
      {
         Type type = Type.GetType(ConfigurationManager.AppSettings[serviceName]);
         object objService = Activator.CreateInstance(type);
         return (IService)objService;
      }
   }
}
```

Figure 8: Factory design pattern implemented in LM web service.

The factory classes in LM acquire their "service names" from the "web.config" parameters file. (The "web.config" parameters file is a generic parameters file used in all Microsoft ASP.NET-based projects to provide application settings at runtime.) The entries in the "web.config" file are referenced by specific classes in the LMWS at application runtime and these parameters are depicted in Figure 8 below.

```
<add key="IResvChairInfoService" value="ListManagerWS.Model.Services.WebConnect.CResvChairInfoService"/>
<add key="IResvPartInfoService" value="ListManagerWS.Model.Services.WebConnect.CResvPartInfoService"/>
<add key="IResvRptInfoService" value="ListManagerWS.Model.Services.WebConnect.CResvRptInfoService"/>
<add key="IResvPactolusBridgeConnService" value="ListManagerWS.Model.Services.WebConnect.CResvPactolusBridgeConnService"/>

<add key="IResvPactolusBridgeConnDao" value="ListManagerWS.Model.Dao.DaoConnect.CResvPactolusBridgeConnDao"/>
<add key="IResvChairInfoDao" value="ListManagerWS.Model.Dao.DaoConnect.CResvChairInfoDao"/>
<add key="IResvPartInfoDao" value="ListManagerWS.Model.Dao.DaoConnect.CResvPartInfoDao"/>
<add key="IResvRptInfoDao" value="ListManagerWS.Model.Dao.DaoConnect.CResvRptInfoDao"/>
```

Figure 9: "Web.config" factory parameters.

The above "web.config" excerpt in figure 9 illustrates the naming convention used to attain classes in code; take note that the key names stipulated by the value of "add

key=" string reference the actual interface name definition. The value returned by any one of these keys is the specific location to the class definition file itself. By not hard-coding these parameters in code, the developer now has the ability to recompile these classes and reinsert them for bug fixes and updates without affecting the rest of the surrounding code aiding in to fulfill the application requirement for ease of updates and maintenance.

Socket Programming

The LMWS communicates with the audio conferencing bridge through an XML-based API and this API is hosted on a physical bridge itself. The bridge makes its API available through exposing of a software-based socket. This socket accepts serialized XML requests regarding the operation of the bridge and the conference calls that the bridge is currently hosting. Figure 10 below is a code excerpt from LMWS depicting a block of socket communication code.

```
namespace ListManagerWS.Model.Dao.DaoConnect
{
   public class CResvPactolusBridgeConnDao : IResvPactolusBridgeConnDao
   {
      private static readonly ILog log = LogManager.GetLogger(typeof(CResvChairInfoDao));

      private TcpClient socket = null;
      private string webBrokerIpAddress = CBridgeConnectionSettings.WebBrokerIpAddress;
      private string webBrokerPortNumber = CBridgeConnectionSettings.WebBrokerPortNumber;
      private string webBrokerAppIpAddress = CBridgeConnectionSettings.WebBrokerAppServIP;
      private string webBrokerAppPortNumber = CBridgeConnectionSettings.WebBrokerAppPortNumber;
      private string webBrokerAppSpId = CBridgeConnectionSettings.WebBrokerAppServSpId;
      private IPAddress clientIpAddress = null;
      private string clientHostName = "";

      private string envOpenTag = "<SOAP-ENV:Envelope xmlns:SOAP-ENV=\"http://schemas.xmlsoap.org/soap/envelope/\" " +
                                  "SOAP-ENV:encodingStyle=\"http://schemas.xmlsoap.org/soap/encoding/\">";
      private string envCloseTag = "</SOAP-ENV:Envelope>";
      private string headerOpenTag = "<SOAP-ENV:Header>";
      private string headerCloseTag = "</SOAP-ENV:Header>";
      private string bodyOpenTag = "<SOAP-ENV:Body>";
      private string bodyCloseTag = "</SOAP-ENV:Body>";
```

Figure 10: "CResvPactolusBridgeConnDao" class excerpt: image 1

Figure 10 above depicts the class variable declarations used for the "CResvPactolusBridgeConnDao," which is a "data access" tier class used to communicate with the new bridge. The "webBroker" declarations accept values from a properties object, which is loaded from an "app.config" file at runtime. The "webBroker" parameters contain values, which allow the LMWS to communicate with the bridge, such as IP address and application port number. The parameters in the second half of the above class declaration contain XML-based tags used to develop the SOAP envelope necessary to transfer XML messages to the receiving server.

Figure 11 below illustrates the "createClientHashKey" method, which is also from the "CResvPactolusBridgeConnDao," class definition. The "createClientHashKey" method generates a somewhat random value, which is used to identify the calling requests from other requests submitted simultaneously.

```
private long createClientHashKey()
{
   string clientHashKey = "";
   long clientTransBase = -1;

   try
   {
      clientHostName = Dns.GetHostName();
      IPHostEntry ipEntry = Dns.GetHostEntry(webBrokerIpAddress);
      clientIpAddress = (IPAddress)ipEntry.AddressList.GetValue(0);
      clientHashKey = clientHostName;
   }
   catch (Exception ex)
   {
      if (clientIpAddress == null || clientIpAddress.ToString().Length == 0)
      {
         Random rand = new Random();
         int randInt = rand.Next(1000000);
         clientHashKey = randInt.ToString();
      }
      else if (clientHostName == null || clientHostName.Length == 0)
      {
         clientHashKey = clientIpAddress.ToString();
      }
   }

   clientTransBase = (long)clientHashKey.GetHashCode();
   if (clientTransBase < 0)
      clientTransBase *= -1;

   return clientTransBase;
}
```

Figure 11: "CResvPactolusBridgeConnDao" class excerpt: image 2

Figure 11 illustrates the use of a homegrown hash key generator, which attempts to obtain a client IP address along with a computer name to formulate a hash key, which can be sent to the receiving server within the SOAP envelope. If a client name and IP address cannot be determined, the use of a random number generator is used to formulate the hash key.

Figure 12 below illustrates the "deleteParticipant" method, which is invoked to delete a participant from the new bridge type. Take note that an XML string is formulated

using the class variables declared in Figure 11 as well as the specific tags necessary to perform the participant deletion; a participant is uniquely identified by the XML tag, "<participantID>."

```
public string deleteParticipant(string reservationNum, int[] delRowsList)
    {
      string response = "";

      for (int i = 0; i < delRowsList.Length; i++)
      {
        createClientHashKey();

        string removeParticipantStr = string.Concat(envOpenTag, headerOpenTag, appTargetIPTag,
                    appTargetPortTag, trxnIdTag, headerCloseTag, bodyOpenTag,

        "<m:", "RemoveConferenceParticipantRequest", " xmlns:m=\"http://schemas.pactolus.com\">",
        "<serviceProviderID>", webBrokerAppSpId, "</serviceProviderID>",
        "<customConferenceID>", reservationNum, "</customConferenceID>",
        "<participantID>", delRowsList[i].ToString(), "</participantID>",
        "</m:RemoveConferenceParticipantRequest>", bodyCloseTag, envCloseTag);

        try
        {
          socket = new System.Net.Sockets.TcpClient(webBrokerIpAddress, Convert.ToInt32(webBrokerPortNumber));

          //System.IO.StreamWriter temp_writer;
          System.IO.StreamWriter streamWriter = new System.IO.StreamWriter(socket.GetStream(), System.Text.Encoding.Default);
          streamWriter.AutoFlush = true;
          streamWriter.WriteLine(removeParticipantStr);

          StreamReader streamReader = new System.IO.StreamReader(socket.GetStream(), System.Text.Encoding.Default);
          response = streamReader.ReadLine();
        }
        catch (System.Exception ex)
        {
          throw new CListManagerSoapException().raiseSoapException("WS-CResvPactolusBridgeConnDao", ex.Message, 5, ex.Source, true, true);
        }
      }

      return response;
    }
```

Figure 12: "CResvPactolusBridgeConnDao" class excerpt: image 3

After the XML string has been formulated, the socket is opened to the destination server and the XML string is transferred and a response received. If an error occurs in the bridge communication process, a "CListManagerSoapException" object is thrown and the error logged on the web server hosting the web service.

Summary

The List Manager Web Service (LMWS) contains all the business logic for the List Manager .NET solution; this business logic includes preparing data and contacting of data sources. LMWS allows for the changing of data source information on a single server and also allows for application patches and updates, without affecting the calling client applications.

LMWS was modeled using the MVC design pattern and implements advanced concepts such as, socket programming and dynamic class loading. LMWS secures its communication with the calling client with Microsoft Kerberos technology, which is an encryption method only available from within a Microsoft Active Directory network.

Chapter 4: “List Manager Thin-Client” Software Project

“List Manager Thin-Client” (LMTC) Project Overview

The “List Manager Thin-Client” (LMTC) project within the “List Manager .NET solution” contains all the source code to produce an LM thin-client. The LM thin-client application displays views, performs data validation, and prepares the data to be sent to the List Manager Web Service for business processing. LMTC can be used on any workstation that supports the .NET framework 2.0 runtime environment and requires zero configuration outside of a dynamic parameter that can be set in the application configuration file to change the web service address currently being used. Please refer to appendix B of this text for screenshots and application process information for the LMTC. (Olsen, 2006)

“List Manager Thin-Client” Project Design Methodology

LMTC was built using the MVC design methodology, but does not include the “data access” tier because data access is an action of the List Manager web service. LMTC ends with the service tier, which is used to transform the data into the appropriate application domain object, which is then used within the service layer to send and receive data communication with the LMWS.

The LMTC implements the factory design pattern and the interface design pattern in the exact same way that LMWS implements these design patterns. Chapter 3 of this text details the technical implementation used for the factory and interface patterns. The LMTC implements dynamic class loading and interface abstraction at the service tier of

the application. Allowing the "service" tier of LMTC to be extensible gives the ability to modify the web service interface without affecting the "view" or "controller" portions of the LMTC application. Figure 13 below is a figure detailing the architectural breakdown of LMTC; the methods and attributes of these classes have been omitted for readability.

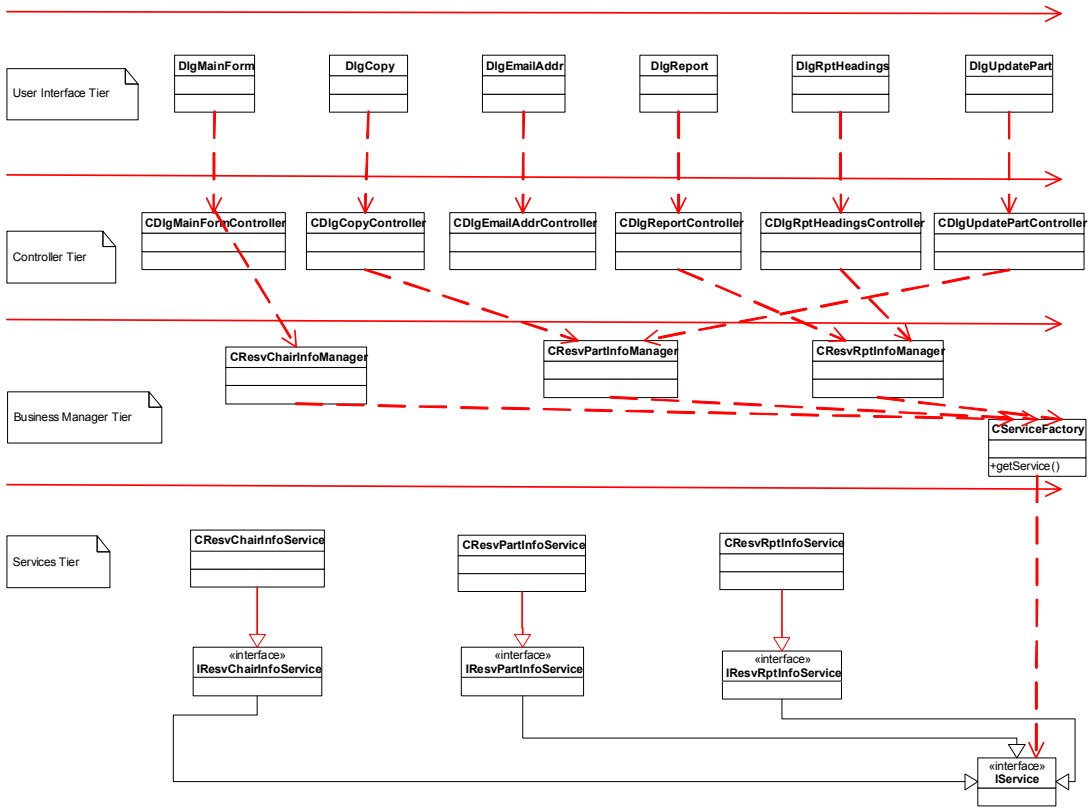


Figure 13: LM thin-client class diagram.

Summary

The LMTC software project produces both a single executable that can be deployed to end-user personal computers as well as single dynamic link library, which can be plugged into a larger .NET application framework. LMTC's primary goal is to

display graphical user interfaces to end-users, collect data, and send that data off to be processed by the List Manager web service.

LMTC implements a complete MVC design for n-tier architecture encompassing both a "view," "controller," and "model" tier components in its implementation. Dynamic class loading was added to the "service" tier of the application to be able to expand and change the web service interface without having to fully regression test the other components of the application. Refer to appendix B of this text for screen captures, data modeling, and process flows for LMTC. (Olsen, 2006)

Chapter 5: “List Manager System Service” Software Project

“List Manager System Service” (LMSS) Project Overview

The “List Manager System Service” (LMSS) project produces a Microsoft Windows system service library, which is referenced by the windows subsystem to perform an ongoing task, in this case, making calls to the LMWS to perform data synchronization with the bridge API and local database instance. Since a web service only runs when a client submits a request and then completes after the request has been fulfilled, it cannot fulfill the requirement on its own to synchronize data at a given time-interval at least not in an acceptable fashion.

A platform system service runs all the time and can run based on time intervals, but this is not the only benefit to creating a platform system service to do data synchronization. The platform system service can be setup to restart itself after an application failure as well as a system failure and this does not have to be done programmatically, but is set within the platform configuration control panel. These normal behaviors that are inherited with the creation of a platform system service both make it reliable and robust.

“List Manager System Service” Project Design Methodology

MVC is a software design pattern that abstracts end-user views from the underlying business model. LMSS does not contain end-user views, but does conform to n-tier architecture with the implementation of an application controller, manager, and service tiers. The LMSS controller tier is made up of a single service class, which makes

calls to the descendant manger tier. The manager tier then passes the calling arguments to the service tier where the web service calls are actually made to synchronize data. Figure 14 below depicts the n-tier architectural make-up of LMSS.

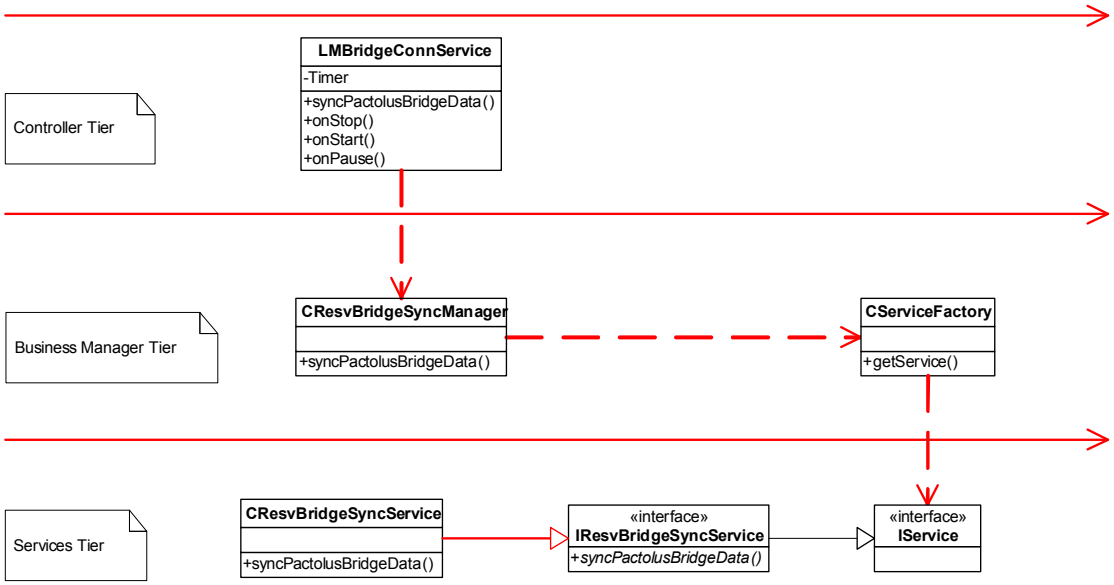


Figure 14: LMSS architectural class diagram

Figure 14 depicts the tiers implemented within the LMSS application. A view tier is omitted because this application is a background process without user dialogs. The data access tier is omitted because the web service handles all data access from within the List Manager .NET solution. The omission of the business modeling in LMSS allows LMSS to completely focus on timed-interval API calls and nothing else.

Summary

The "List Manager System Service" LMSS project is a platform system service that runs at all times on the platform that hosts it. A system platform service is an ideal choice for this type of process because it allows the application to recover from both

system and application level failures without any additional coding; the parameters are set in the platform system control panel.

LMSS does not conform to the MVC design pattern rules because by definition it does not abstract user views from the business model. LMSS does however conform to n-tier architecture and does implement a "manager," "service," and "controller" tiers. The "service" tier is the location of the web service calls, which actually make the web requests to synchronize data and the "controller" tier handles the timed-interval process, while the "manager" tier just directs the timed-interval requests from the "controller" tier to the "service" tier.

Chapter 6: Project History / Software Development Life Cycle

Overview

GCC is pursuing an extension of their audio conferencing bridge product lines to be used to host audio conference calls world wide. This product extension requires changes throughout the entire business sector, including billing, operations, marketing, and many other facets of the organization. The project manager of this project manages all facets of the project from idea conception to the product maintenance phase of the project. The following sections of text will focus on the history of LM and the life cycles utilized to implement the LM software project. Although the LM project is a subproject of a larger initiative to integrate a new audio conferencing bridge type within the GCC business environment, the scope of this text is the LM software project itself and focuses on the methodologies used to create the LM application framework. The following sections will look at the history of the LM project and the business methodologies used to see this project through to completion.

LM Project History

The software project conception date for LM is March 15, 2005. (Levin, 2005) The conception date is determined by the date in which software engineering receives the appropriate requirement documentation necessary to formulate an implementation of a software project. The requirements documentation is ultimately the company's sign-off as a whole that the business is going to invest the funding necessary to implement the conceived business idea. The ultimate project deliverable date is October 13, 2006 and

this is the ultimate production release date being communicated to GCC end customers, although software engineering has an ultimate project deliverable date of July 15, 2006, which is determined as the date, in which the software needs to be fully functional in the GCC certification environment.

LM Software Development Life Cycle

The LM software project uses the "object-oriented life cycle" (OOLC) approach to software design. OOLC follows a waterfall life cycle model, but implements iterations throughout the life cycle that span hard boundaries. Although it is not uncommon for the design and implementation phases to overlap, a software engineer and supporting business team must defend against the "CABTAB" approach to development, "Code a bit, test a bit." (Schach, 2002) There is an inherent danger in using the object-oriented life cycle in that the methodology can break down into unorganized patterns of development and testing. This lack of discipline within a software project can potentially hide large architectural design flaws until they are encountered later on in the project, instead of right up front in the design phase.

The LM software project was carefully placed into the OOLC methodology, taking special care to ensure that methodologies implemented are in fact beneficial to the LM project itself and not just extra time taken for sake of practice. Figure 15 below figure depicts the software development life cycle used within the LM project. Take note of the iterative approach applied to the water fall boundaries.

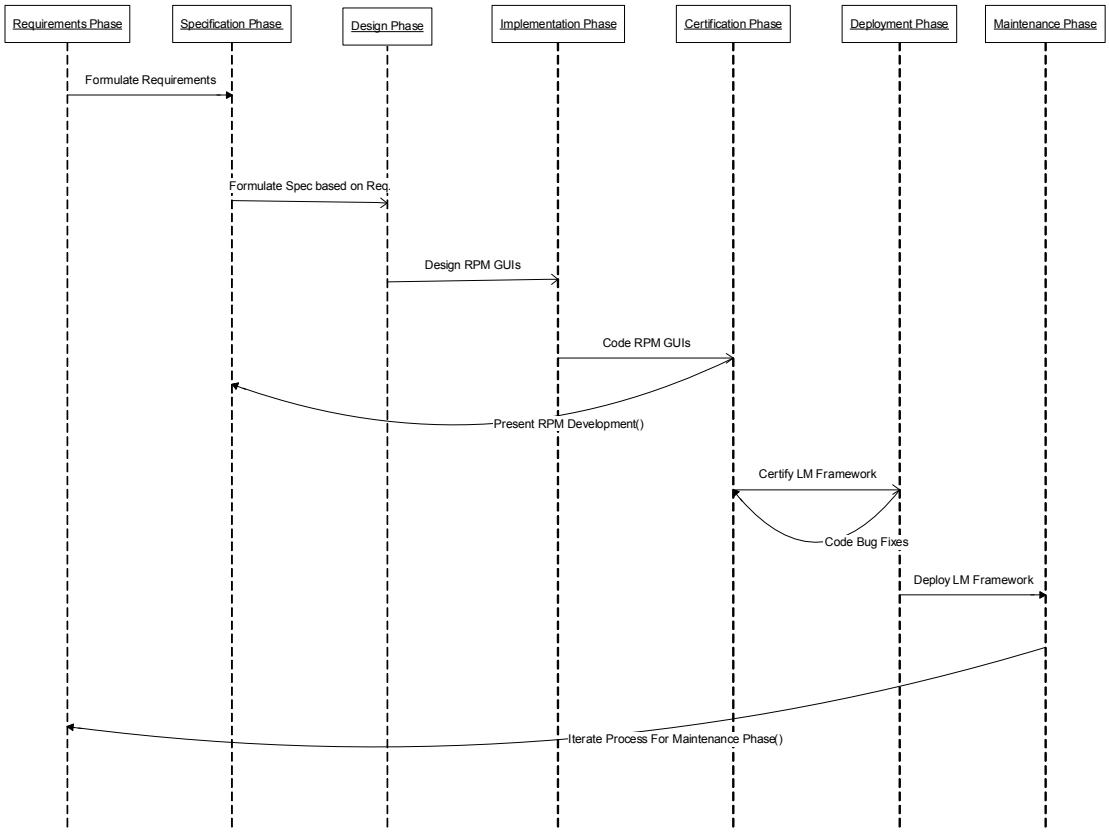


Figure 15: Object-Oriented Life Cycle used in LM

Figure 15 depicts the object-oriented software development life cycle used in the LM software project. Notice that our OOLC implementation mimics the traditional waterfall life cycle approach with specific points of iteration. The following sections of text describe the different phases of the OOLC used in LM as well as the relationships and iterations between the phases.

Requirements Phase

During the requirements phase of LM, a business analyst conducts interviews with business personnel as well as technical personnel to define the LM application requirements; a full version of the requirements documentation is available in Appendix A of this text. (Levin, 2005) After information was collected and requirements documentation written, the documentation was turned over to software engineering for requirements acceptance. During the requirements acceptance phase of the project, software engineering collaborates internally and externally through the business analyst to clarify any specific points in the requirements documentation as well as to determine a certification release date.

Specification / Design / Implementation Phase

The specification, design, and implementation phase of this project are collectively implemented in iteration. The specification phase of the software life cycle defines the graphical user interface and end-user functionality of the software application through the "Rapid Prototyping Model." (RPM)

(RPM) was used to define the specification phase of this project. RPM is a methodology, which defines methods to produce a working subset of an application to demonstrate the functionality of the system. (Schach, 2002) The end-user thin-client graphical user interface was the first portion of the application framework created. Creating a working end-user interface, which can be demonstrated to the end-users of the system, allows them to envision what the application is going to look like; it is at this time that end-user feedback is collected on the new application model and any glaring

problems are corrected. The specification phase is married to the design and implementation phases in that both modeling diagrams and code was written to implement the RPM for the specification phase of this project.

Iteration using RPM was very beneficial in that many great features were added to the system, which made sense, but maybe more business benefits were seen through the reduction of legacy features no longer being used by the end-user base. Although the additional enhancements and the reduced features may cancel themselves out in this project, GCC has always prided themselves in creating software that is tailored to end-user needs making the software more usable and efficient for the ultimate users of the software project.

The design phase is defined by architectural diagrams depicting the state of the final application framework to be created. These diagrams include business diagrams, UML modeling diagrams, and any other documentation related to the final blue print of the application to be created.

The implementation phase of the software life cycle is the coding phase of the application creation process and is defined by source code compiled to generate a working application framework. The implementation phase is where the bulk of the work is performed to create the application system.

Certification Phase

During the certification phase of the software development life cycle, certification plans are written to test the functionality of the LM application framework; all test plans used for the LM project were performed through the LM thin-client application, which is

truly the controller of the entire system spanning three physical boundaries. The certification department for GCC allotted eight weeks for certification testing on the new LM application framework. This time period included both acceptance testing and pilot testing. Figure 16 below depicts one of the certification plans written by a GCC operations supervisor, testing the application functionality and depicting the bugs found in the application during the first certification iteration.

No	**Test**	**Status**	**Expected Results**	**Notes**
1	Search for Reservation Number	**Pass**		
2	Search for Reservation Number with list Preloaded	**Pass**		
3	Load an excel Prelist (must follow loading rules)	**Pass**		
4	Update one entry using Update button	**Pass**	The change is accepted.	
5	Update one entry using double-click	**Pass**		
6	Add one entry using Add button	**Pass**	All information appears	
7	Delete one entry using Delete button	**Pass**	Receive message requesting if I wish to delete.	

			Deleted after pressing yes.	
8	Delete several entries using CTRL and mouse	**TBD**	Receive message requesting if I wish to delete including the amount of lines. Deleted after pressing yes.	Deletion occurs. Total number of lines being deleted does not show on warning message. Mentioned to Maurice, he will look into it.
9	Delete several entries using click and drag.	**TBD**	Receive message requesting if I wish to delete including the amount of lines. Deleted after pressing yes.	Deletion occurs. Total number of lines being deleted does not show on warning message. Mentioned to Maurice, he will look into it.
10	Copy existing list to second unused reservation	**Pass**	Should copy Complete total lines from existing list as unattended.	
11	Able to clear reservation using Clear button	**Pass**	Screen should appear blank.	
12	Able to reopen session from Recent Reservation Numbers	**Pass**	Should reappear by simply highlighting the	

			line.	
13	Able to sort by First Name	**Pass**		
14	Able to sort by Last Name	**Pass**		
15	Able to Sort by Company Name	**Pass**		
16	Able to Sort by Location	**Pass**		
17	Able to Sort by Phone No	**Pass**		
18	Able to Sort by Attended	**Pass**		
19	Able to Sort by Priority	**Pass**		
20	Report Menu defaults to Attended	**Fail**		I attempted to open the menu while my Conference Details read Attended, as well as Complete List. When opening the reports menu, Report Type was appearing by Complete Participant List by default
21	Report Menu defaults to Portrait	**Pass**		
	Changing Headings			
22	Able to select all fields	**Pass**		
23	Sort By Defaults to Last Name	**Pass**		
24	Change Sort by to Location	**Pass**		

25	Change Sort by to Company	**Pass**		
26	Change Sort By to Last Name	**Pass**		
27	Viewing the list by Complete List	**Pass**		
28	Update Participant to appear as attended	**Pass**		
29	Viewing the list by Attended List	**Pass**		
30	Viewing the list by No Show List	**Pass**		
31	Previewing List shows Total Lines	**Pass**		
32	Email list	**Pass**		

Figure 16: LM certification test plan. (Caporale, 2006)

The above certification plan in figure 16 was used by the pilot test group to conduct repetitive tests on processes and functionality within LM. When a bug was found, it was so noted along with any details for the developer to help duplicate the issue. The developer would release a fixed version of the software and the entire process would start again.

Deployment Phase

The deployment phase of LM involves all software components of the LM framework to be deployed to production systems. Production deployment took place on

October 13, 2006 and included deployments for all systems in the LM project as well as all other facets of the business that needed modification to support the new Pactolus-based bridge type.

Recall from previous chapters in the text, that the LM application framework is four separate physical components. The List Manager Domain Library (LMDL) consists of a single dynamic link library, which will be deployed with software projects that require it, so no development routine was devised to deploy the LMDL. The LM Web Service application (LMWS) will be deployed by software engineering by deploying the appropriate web service and its dependencies through a publication wizard available through the IDE; software engineering is granted access temporary to production systems during deployment windows on October 13, 2006. The LM Thin-Client (LMTC) and LM System Service (LMSS) applications will be deployed using a universal installer created by software engineering through InstallShield DevStudio technology. InstallShield DevStudio is a development tool that allows software engineers to encapsulate their software into standard platform installer routines, which will allow for application deployment in a heterogeneous environment as well as increase compatibility with products like Microsoft Systems Management Server (SMS); a systems administrator will use the LMTC installer and LMSS installer to deploy these components of the framework to their respective destinations.

Maintenance Phase

The maintenance phase of LM consists of routine changes for the first six months of operation to update the Pactolus bridge type API with additional supporting methods

to increase system performance. The LM application framework was designed using MVC methodology and implemented the Factory design pattern to aid with ease of application maintenance. In the maintenance phase of this application framework, additional features and updates can be made to components within the LM application framework without having to perform full regression testing. The concepts implemented in this project promote component decoupling, minimizing the impact to processes in the code, which are not being modified.

Summary

The LM application framework uses OOLC methodology of application development. The OOLC methodology is an iterative version of the waterfall life cycle model. The OOLC used in this text implemented the requirements, specification, design, implementation, certification, deployment, and maintenance phases of the waterfall life cycle, iterating the specification, design, and methodology phases using RPM.

The requirements phase consisted of interviews, research, and requirements documentation acceptance. The specification, design, and implementation phases utilized an RPM modeling iterative approach for development. The certification phase was defined by acceptance testing and pilot testing. The deployment phase consisted of deploying all four application components to production systems using a combination of web publishing available through the Microsoft .NET IDE and universal installer-based packaging. The maintenance phase consists of feature enhancements, API enhancements, and any other changes to the system. The maintenance phase is just another iteration of

the software development life cycle starting with the requirements phase to define the changes or additions to be made to the LM application framework.

Chapter 7: Lessons Learned

Lessons Learned Overview

The LM project has posed many challenges, both technical and inter-personal. This project from initial thought to completion took nine months. From the initial requirements phase to the production rollout phase, the LM project has been riddled with changing requirements due to a lack of understanding by the business sector of the new processes that need to be implemented. The following sections of this text will examine specific problematic issues of the LM project and what could be done differently in future projects to avoid these challenges.

Call Center Supervisor Interview

During the initial requirements phase of the LM project, interviews were conducted with a single GCC operations supervisor, which is an individual that manages the GCC operator end-user base. These interviews ranged from enhanced feature requests to customer care processes on how they intend to use LM with the new Pactolus bridge type.

The GCC operations supervisor interview went well, with the exception of a specific interview question, “Are there any systems besides List Manager that will modify participant details on the bridge?” The GCC operations supervisor stated that the LM application was to be the only application that will modify participant information on the bridge. The call center supervisor assumed that this new Pactolus bridge type will follow the same business processes as the legacy bridge type already in use.

The new Pactolus bridge type vendor informed GCC's business analysts that their understanding of the technical process of participant information modification on the bridge is not correct. Pactolus only allows participant modifications from their proprietary API before a conference call actually starts. Modifications to participant information after the call starts will take place utilizing a specific Pactolus operations console. (The Pactolus operations console allows a customer care representative to perform conference call operations during a call, but is not functional until a conference call begins and is disabled when the call has ended.) The GCC operatations supervisor's intentions were to not have the GCC operator add, modify, or remove participants using this operations console and force a business process to use the LM application for all participant modifications. The LM application has an enhanced GUI specific to customer care business needs and reporting capabilities, which are not possible using the Pactolus bridge console application, so it seemed reasonable to the GCC operations supervisor at the time. This single process change increased development time by twenty-five percent, since an additional system service application had to be written to perform data synchronization utilizing the List Manager Web Service application. (Refer to Chapter 6 of this text for a complete application definition of the List Manger System Service application.)

The cause for this unforeseen issue was inadequate documentation by the Pactolus bridge vendor. The business analysts wrote requirements for the LM application based on the documentation available at the time, which did not clearly define the relationships between the Pactolus operations console and GCC's new List Manager application.

The issue has been documented by the project team and future audio conferencing bridge vendors will be asked to detail any processes between the bridge operation console and home-grown development preformed against the new bridge vendor's proprietary API.

Pactolus API (New Audio Bridge Type)

The new Pactolus audio conferencing bridge vendor builds a customized API for its larger customers and the API given to Global Crossing was largely determined by business analysts working for Global Crossing. There were two notable issues with the Pactolus API: incorrect/incomplete API technical documentation provided by Pactolus and not all requirements written for Pactolus by Global Crossing business analysts encompassed all technical processes needed to build a scalable application using the Pactolus API.

Pactolus released new versions of its API on a monthly basis while GCC was already in its development phase. GCC was given a complete API guide from Pactolus to develop code against, although their API was not fully completed yet and new versions of the code were being released to GCC from Pactolus on a monthly basis. Pactolus API releases and changes caused major headaches for the entire project team. Many technical process changes to LM took place over the course of nine months of development and some of these process changes required changes in implementation and application design mid-way through development. Countless hours were wasted due to poor documentation and inconsistent API releases.

In the future, software engineering will be pushing for accurate documentation up front and will make a critical point of it to the vendor. Although the ultimate project deadline was met and delays were not incurred, the cost of labor for doing this project exceeded expectations, although didn't put the project at risk of failure.

Scope Creep

Interviews were conducted with a customer care supervisor detailing general feature enhancements to the LM application, not including the new Pactolus bridge type enhancements. After the requirements were accepted by software engineering, software engineering preformed "rapid application development" (RAD) to produce user interfaces with no additional back-end functionality. Once the RAD development was completed, customer care supervisor sign-off was accepted.

During the implementation phase, numerous versions of the LM thin-client were demoed for all GCC operations supervisors and although they were satisfied with the general content agreed upon, additional recommendations were made for feature enhancements. The additional feature enhancements were relatively easy for software engineering to perform, so the department committed on changing the requirements to include some additional enhancements. Shortly after software engineering committed to the additional feature enhancements, the Pactolus process flaws were noted as defined in the above section, "Customer Care Supervisor Interview." These unforeseen technical process issues in conjunction with the additional feature enhancements, which software engineering committed to, pushed the time limits and work was performed around the clock for a short period of time to ensure that the project did not fall behind schedule.

In the future, software engineering will be more reluctant to commit additional feature enhancements unless absolutely necessary. Although the additional requirement changes did not take that much additional development time, it was almost part of a system of events that could have put the LM project at risk.

Software Certification

Global Crossing's software certification department has went through reductions in workforce since the start of the LM project and application end-users are now being asked to perform the duties that were once performed by software certification engineers. Although the end-user base is an excellent test bed for application certification to some degree, the end-user base is not trained in software certification test plans and therefore, there was a lack of structure during the certification process of this project.

The consequences of end-user base testing have been an increased reliance on a job well done by the software engineering department and an increased reliance on the end-user base for certification testing. Pilot test groups are groups of end-users that have agreed to invest time in testing the new application being certified. Pilot groups test applications after the application has already been released from certification and therefore generally have high expectations when they receive the software to be certified, even though they are now receiving the software one iteration before pilot testing.

In order to combat the problems with end-user base testing, software engineering has had to re-educate GCC operations to note that the software they will be testing will in fact have bugs in it and will not be ready for production as they would normally anticipate. In addition to changing customer care's expectations, software engineering

will be providing the end-user base in future projects with software certification test plans for the application being presented. Although it is always beneficial to have an individual or department write test plans for an application from an objective perspective, under the current circumstances, this is the best solution GCC can implement until funding becomes available to replace lost certification resources.

Summary

The LM application has endured many obstacles and unforeseen issues, both technical and inter-personal. Changing technical and business requirements were common throughout the List Manager Application Life-Cycle and additional processes will be put into place for software engineering and business to combat changing requirements from the end-user base as well as from technical vendors.

The majority of technical obstacles revolved around the Pactolus API; the Pactolus API allows an application to control an audio conferencing bridge programmatically. New releases of the API were deployed monthly and functionality written in the original documentation was subject to change, impacting any code written against the API.

New business processes noted by the project team will combat these issues with future vendors. These new processes detail the relationship between the audio bridge operations console and a connecting API thin-client.

GCC's certification department has gone through recent turnover and customer care pilot users are being asked to enter the project one phase earlier then normal to perform certification testing. GCC's software engineering department has gone through

challenges to change customer care's expectations of the software received from a certification released version to a version that still has some bugs in it. In the future, software engineering is going to aid customer care in application certification by writing the test plans necessary for the pilot users to do good solid testing.

Works Cited

Anonymous. (2004). *User interface process (UIP) application block– version 2.0.* Microsoft Corporation.

Caporale, Cliff. (2006). *Certification Plan: List Manager.* Global Crossing, Inc.

Dietrich, Hans. (2004). *Step by Step: Calling C++ DLLs from VC++ and VB – Part 3.* The Code Project.

Jillellamudi, Ramakrishna. (2004). *Introduction to .NET Assemblies.* DNZone.com

Jones, Bradley. (2003). *Microsoft .NET Glossary.* Developer.com.

Levin, Daria. (2005). *Requirement: List Manager.* Global Crossing, Inc.

Olsen, Maurice (2006) *List Manager Technical Specification Document.* Global Crossing, Inc.

Schach, Stephen. (2002). *Object-oriented and classical software engineering.* McGraw-Hill Companies, Inc.

Yang, James. (2001). *What is n-tier architecture?* Developer Fusion.

Annotated Bibliography

Aitken, Peter. (2002). Creating windows services in .NET

DevX.com. Date Accessed: 10/29/2006. Retrieved from:

http://www.devx.com/dotnet/article/7001

Reason: Referenced for creating a windows service on the server-tier to access the web service for syncing capabilities of data in timed- intervals.

Aitken, Peter. (2006). Multithreading in .NET 2.0: The ThreadPool class. DevSource.

Date Accessed: 10/15/2006. Retrieved from:

http://www.devsource.com/article2/0,1895,2019360,00.asp

Reason: Researched multi-threading concepts to implement multi-threading in the web service tier for enhanced performance of DB uploads.

Allen, Paul. (2001). *Realizing e-business with components.*

London, England: Pearson Education Limited.

Reason: Referenced to enforce MVC design pattern in multi-tier physical application architecture.

Anonymous. (2005). Reflection and dynamic class loading.

Microsoft Corporation. Date Accessed: 10/18/2006. Retrieved from:

http://msdn2.microsoft.com/en-us/library/ms227224.aspx

Reason: Referenced to implement dynamic class loading in both the client and the web service tier of the application infrastructure.

Anonymous. (2006). Windows identity class.

Microsoft Corporation. Date Accessed: 10/27/2006. Retrieved from:

http://msdn2.microsoft.com/en-us/library/system.security.principal.windowsidentity.aspx

Reason: Referenced to gather user login information for logging purposes on the web service tier.

Anonymous. (2006). How to create a setup project for a windows service in Visual C# .NET and in Visual C# 2005

Microsoft Corporation. Date Accessed: 10/29/2006. Retrieved from:

http://support.microsoft.com/kb/816169/

Reason: Referenced for creating an installer for a windows service on the server-tier.

Anonymous. (2004). User interface process (UIP) application block– version 2.0

Microsoft Corporation. Date Accessed: 10/29/2006. Retrieved from:

http://msdn2.microsoft.com/en-us/library/ms979213.aspx

Reason: Referenced for creating MVC designed applications in .NET.

Anonymous. (2003). A guide to building enterprise applications on the .NET framework.

Microsoft Corporation. Date Accessed: 10/29/2006. Retrieved from:

http://msdn2.microsoft.com/en-us/library/ms954601.aspx

Reason: Referenced to help design an enterprise application framework using service oriented architecture.

Anonymous. (2006). How to automate Excel by using Visual C# to fill or to obtain data in a range by using arrays.
Microsoft Corporation. Date Accessed: 10/20/2006. Retrieved from: http://support.microsoft.com/kb/302096/
Reason: Referenced to implement Microsoft Excel report generation and data importing for the forms client application.

Bourisaw, Mark. (2004). Generating Microsoft Excel reports in .NET
C# Corner. Date Accessed: 10/29/2006. Retrieved from: http://www.c-sharpcorner.com/UploadFile/bourisaw/ExcelReportsInNet11092005001455AM/ExcelReportsInNet.aspx?ArticleID=3e6ed057-5306-4c8e-84fd-0fde37848b2c
Reason: Referenced to implement Microsoft Excel report generation and data importing for the forms client application.

Chappell, David, & Kirk, Steve. (2002). Application design guidelines: From n-tier to .NET. Microsoft Corporation. Date Accessed: 10/29/2006. Retrieved from: http://msdn2.microsoft.com/en-us/library/ms978384.aspx
Reason: Referenced to aid in building a multi-tier architected application framework.

Chariter, Robert. (2002). Creating an extensible windows service

15 seconds. Date Accessed: 10/27/2006. Retrieved from: http://www.internet.com/icom_cgi/print/print.cgi?url=http://www.15seconds.com/Issue/021007.htm

Reason: Referenced for creating a windows service on the server-tier to access the web service for syncing capabilities of data in timed- intervals.

Chong, Frederick, & Clark, Jim, & Morris, Max, & Welsh, Dave. (9/2005). Web service solution design: Developing a solution design for web services in the northern electronics scenario. Microsoft Corporation. Date Accessed: 10/27/2006. Retrieved from: http://msdn2.microsoft.com/en-us/library/ms954617.aspx

Reason: Referenced to study and architect a web service that models the business scenarios that the web service will support.

Dietrich, Hans. (2004). Step by Step: Calling C++ DLLs from VC++ and VB – Part 3. The Code Project. Date Accessed: 11/20/2006. Retrieved from: http://www.codeproject.com/dll/XDllPt3.asp

Reason: Supporting reference for calling .NET assemblies from a heterogeneous .NET programming language environment.

Janczuk, Tomasz. (2006). Protect your web services through the extensible policy framework in WSE 3.0. Microsoft Corporation. Date Accessed: 10/19/2006. Retrieved from:

http://msdn.microsoft.com/webservices/default.aspx?pull=/msdnmag/issues/06/02/wse30/default.aspx

Reason: Referenced to develop a secure web service methodology.

Jones, Bradley. (2003). Microsoft .NET Glossary.

Developer.com. Date Accessed: 11/18/2006. Retrieved from:

http://www.dnzone.com/ShowDetail.asp?NewsId=698

Reason: Referenced for .NET assembly research and reference.

Jillellamudi, Ramakrishna. (2004). Introduction to .NET Assemblies.

DNZone.com. Date Accessed: 11/18/2006. Retrieved from:

http://www.dnzone.com/ShowDetail.asp?NewsId=698

Reason: Referenced for .NET assembly research and reference.

Kurniawan, Budi. (2002). Using .NET sockets

ONDotNet.com. Date Accessed: 10/26/2006.

Retrieved from: http://www.ondotnet.com/pub/a/dotnet/2002/10/21/sockets.htm

Reason: .NET socket programming is one of the technical aspects needed to enable server-to-server communication.

Meier J.D., Mackman Alex, Dunner Michael, and Vasireddy Srinath. (2006). Building secure ASP.NET applications: Authentication, authorization, and secure communication.

Microsoft Corporation. Date Accessed: 10/13/2006. Retrieved from:

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnnetsec/html/SecNetch10.asp

Reason: Authentication and security web service research and design.

Mitchell, Scott. (2004). An extensive examination of web services: Part 9

4 Guys from Rolla.com. Date Accessed: 10/17/2006. Retrieved from: http://aspnet.4guysfromrolla.com/articles/071404-1.aspx

Reason: Referenced to develop a secure web service methodology.

Panchal, Nimesh. (2003). Implementing MVC design patterns in .NET

C# Corner. Date Accessed: 10/27/2006.

Retrieved from: http://www.c-sharpcorner.com/Code/2003/Feb/MVCDesign.asp

Reason: Referenced for creating MVC designed application in .NET

Parmar, Chandrakant P. (2005). Understanding threading in .NET Framework. C# Corner. Date Accessed: 10/17/2006. Retrieved from: http://www.c-sharpcorner.com/Code/2005/April/Thread.asp

Reason: Researched multi-threading concepts to implement multi-threading in the web service tier for enhanced performance of DB uploads.

Patil, Vishal Kumar. (2005). CodeSnip: Handling SOAP exceptions in web services.

ASP Alliance. Date Accessed: 10/19/2006. Retrieved from: http://aspalliance.com/727_CodeSnip_Handling_SOAP_Exceptions_in_Web_Services

Reason: Referenced to develop a methodology to handling exceptions thrown by the web service tier.

Poddar, Saumendra. (2003). Introduction to .NET web services.

The Code Project. Date Accessed: 10/27/2006Retrieved from:

https://secure.codeproject.com/dotnet/intro2websvc.asp

Reason: Referenced to aid in creating a .NET web service.

Prosise, Jeff. (2003). Build secure web services with SOAP headers and extensions.

Codeguru.com. Date Accessed: 10/19/2006. Retrieved from:

http://www.codeguru.com/columns/experts/article.php/c5479/

Reason: Referenced to implement and pass custom SOAP headers for security implementation.

Ramana, G.V. (2006). How to get Window NT logged user name using ASP.NET.

The Code Project. Date Accessed: 10/10/2006. Retrieved from:

http://www.codeproject.com/useritems/How_to_NT_User_Name.asp

Reason: Referenced to gather user login information for logging purposes on the web service tier.

Richter, Charles. (1999). *Designing flexible object-oriented system with UML.*

Indianapolis, Indiana: Macmillan Technical Publishing.

Reason: Referenced for UML class diagramming and use-case modeling.

Schach, Stephen. (2002). *Object-oriented and classical software engineering.*
New York, New York: McGraw-Hill Companies, Inc.
Reason: Referenced to implement object-oriented life cycle approach in application development.

Smon, Uros. (2001). Multithreading in .NET.
The Code Project. Date Accessed: 10/27/2006. Retrieved from:
http://www.codeproject.com/dotnet/multithread.asp
Reason: Researched multi-threading concepts to implement multi-threading in the web service tier for enhanced performance of DB uploads.

Spano, John. (2005). Multithreading in VB.NET
DevCity.NET. Date Accessed: 10/27/2006. Retrieved from:
http://www.devcity.net/Articles/160/1/article.aspx
Reason: Researched multi-threading concepts to implement multi-threading in the web service tier for enhanced performance of DB uploads.

Strawmyer, Mark. (2006). Multithreading in .NET applications.
Developer.com. Date Accessed: 10/27/2006. Retrieved from:
http://www.developer.com/net/asp/article.php/2202491
Reason: Researched multi-threading concepts to implement multi-threading in the web service tier for enhanced performance of DB uploads.

Trott, James R., & Shalloway, Alan. (2002). *Design patterns explained: A new perspective on object-oriented design.*

Indianapolis, Indiana: Addison-Wesley.

Reason: Referenced for Factory design pattern modeling with the web and client tier.

Turner, Richard. (2005). Performance of ASP.NET web services, enterprise services, and .NET Remoting. Microsoft Corporation Date Accessed: 10/28/2006. Retrieved from: http://msdn.microsoft.com/webservices/choosing/default.aspx?pull=/library/en-us/dnwebsrv/html/asmxremotesperf.asp

Reason: Web service performance was a concern for the technical team and research was conducted to design a scalable and efficient methodology.

Yang, James. (2001). What is n-tier architecture?

Developer Fusion. Date Accessed:11/14/2006. Retrieved from: http://www.developerfusion.co.uk/show/3058/2/

Reason: Referenced for n-tier architecture methodology.

Appendix A

Project Name	Document Name
Event Bridge Replacement	**Requirement: List Manager**

Author	Origin Date	Last Updated	Status
Daria Levin	***3/15/2005***	***2/26/2007***	***Draft***
Interviewees: Cliff Caporale; Bhattacharjee, Raja			

Requirement Description

Modification History

Sign-off Date: *Date that users and developers signed off on the requirement.*

Date	*Name*	*What has been changed and in which section of the document the changes were made.*
4.26.06	D. Levin	Updated to reflect that we will also remove the Text File button. Also added two notes to the issues section based on Maurice's analysis.

Synopsis

As part of the Event bridge replacement project, we are building an API between List Manager and Pactolus to pull and report in participant information. We are also adding functionality to List Manager to support new Pactolus only fields.

Description

Note to tester: List Manager will be migrated to a .net environment. Please test in Citrix and local PC to make sure functionality is the same. The .net changes will otherwise be seamless.

List Manager functionality and GUI changes

1. List Manager should pull data from Pactolus for a current conference (just as Allegro does today)
2. List Manager must pull the following new fields from Pactolus back to List Manager and must accommodate the following new columns in the main screen:
    - Fax Number
    - email
    - Other Information Note
    - Number of people in room
    - Custom Field label 1
    - Custom Field label 2
    - Custom Field label 3

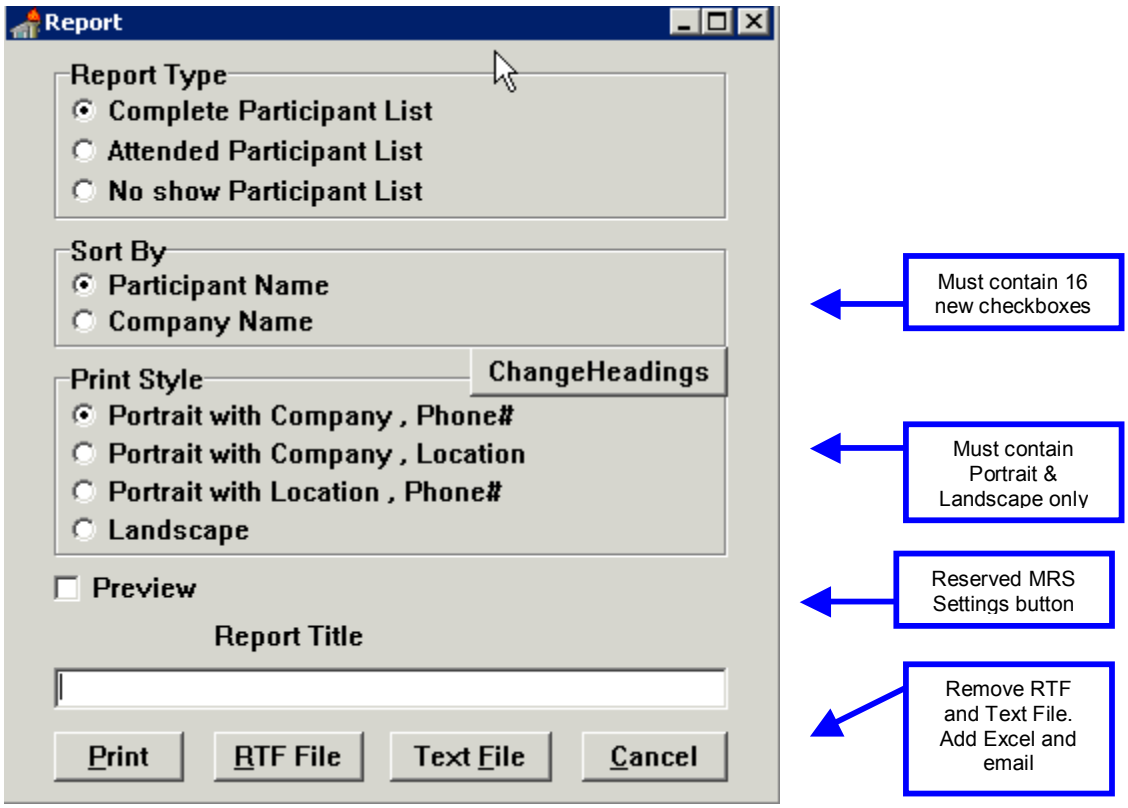
- Custom Field label 4
- *Connect time**
- *Disconnect time**
- *Priority (this comes in the pre-list)*

* MRS must flag whether these two fields are required for a given call.

NOTE: These new fields will not apply to Allegro. We propose adding a message to MRS if a user is checking off a 6th service stating: “For more than six selections, the call cannot be booked on Allegro.”

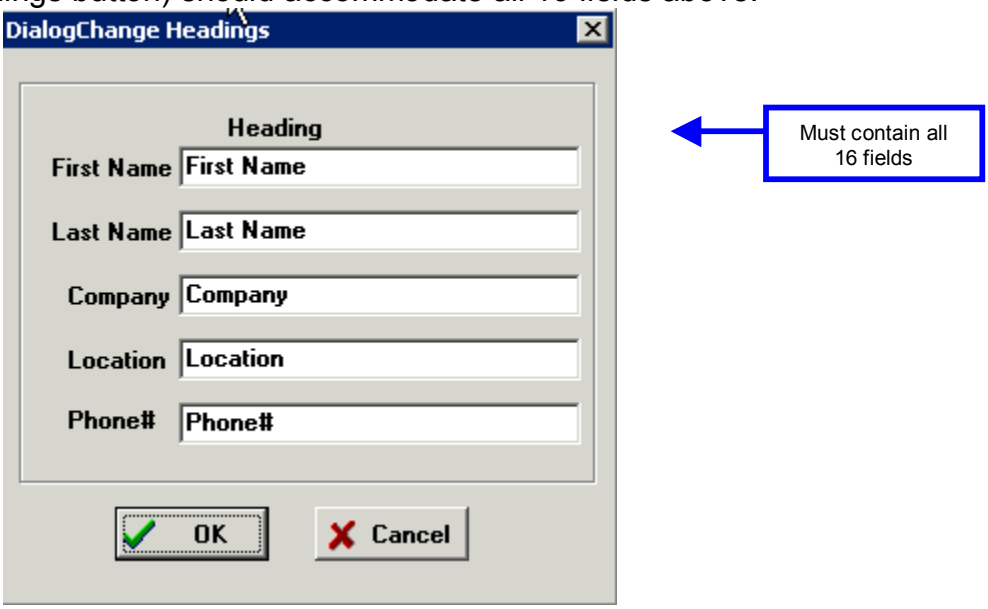
3. We need a character limit increase in List Manager (Pactolus supports: first name: 20 characters, last name: 30 characters, remaining fields all 30 characters)
4. If Customer Care is printing a list from a call conducted on Pactolus, the report screen will look different (for a call conducted on Allegro, there will be no change other than the two new print button options). The Report screen will have the following changes:

    4.1. the Sort By group box must display all possible fields as checkboxes:
        1. First Name
        2. Last Name
        3. Company name
        4. Location
        5. email address
        6. Caller's Phone Number
        7. Fax Number
        8. Other Information Note
        9. Number of people in room
        10. Custom Field label 1
        11. Custom Field label 2
        12. Custom Field label 3
        13. Custom Field label 4
        14. Priority
        15. *Connect time**
        16. *Disconnect time**



4.2. the user should be able to check one ore more checkboxes to select which columns should print on the report

4.3. The DialogChange Headings box (that opens when you click ChangeHeadings button) should accommodate all 16 fields above.



4.4. The Print Style group box should only contain the Portrait or Landscape options (the radio buttons above will determine what columns print).

5. The Report screen must have two new buttons: “Excel” and “e-mail”

5.1. If Excel is clicked, List Manager will export a comma delimited file (good for large reports)

5.2. If e-mail is clicked, a box opens with a field to enter one e-mail address.

5.2.1. By default, the window must be pre-populated with lists@cfer.com.

5.2.2. The user must have the ability to overwrite the default e-mail with another.

5.3. Remove the RTF button

5.4. Remove the Text File button

5.5. The four button changes above (two new buttons and two removed) must apply **to both Allegro and Pactolus** calls.

6. The Report screen must have a buttons labeled "Reserved MRS Settings"

6.1. List manager should pull all reserved settings from the Participant List window

6.2. When the Reserved Settings button is clicked, a window opens containing all of the 16 possible settings (listed above) and display what was selected in MRS for this reservation via Read-only checkboxes

7. Remove the PR Firm field from the list manager main screen (never used)

List Manager API changes

8. List Manager must send pre-list information to Pactolus using the following method (fields in blue are new fields we're adding to MRS with this release).

From AddParticipantToConferenceRequest

Pactolus API Field Name	Required	Transfer to Pactolus? (Y/N)	Corresponding MRS Field Name/Setting
serviceProviderID	**Yes**	Y	1000
customConferenceID1	**Yes**	N	Reservation #
Moderator	**Yes**		FALSE
dialOut	**Yes**		No
initializeMutedflag	**Yes**		
subscriberFlag	**Yes**		
companyName	No		Caller's company name (from Event profile)
FirstName	**Yes**		Caller's first Name (from Event profile - splitting first

			& Last name)
MiddleName	No		No
LastName	**Yes**		Caller's last Name (from Event profile - splitting first & Last name)
emailAddress	No		**e-mail address**
phoneNumber1	No		Caller's Phone Number
phoneExt1	No	N	
phoneType1	No	N	
phoneNumber2	No	N	
phoneExt2	No	N	
phoneType2	No	N	
faxNumber	No		Caller's Fax Number
location	No		City State/Province
partyPasscode	No	N	
Priority	No		*Determined and contained in pre-list*
custom1	No		**Custom Field**
custom2	No		**Custom Field**
custom3	No		**Custom Field**
custom4	No		**Custom Field**
nbrPeopleInRoom	No		**Number of people in room**

Constraints/Assumptions/Issues

- There is no change to the Allegro experience in List Manager other than seeing the new columns on the main screen and the three button changes in the Report options.
- OPAL will not show any of the new fields. Customers tend to view OPAL during an ongoing conf to see who's on and use it to help the Comm line with Q&A. Leaderview will eventually replace Opal. We can tell customers they will only view current 5 columns (plus priority) through OPAL. A complete list will be sent after the call. (better than not using OPAL at all if call is on Pactolus)
- If we correct the prelist data (from the original value), Pactolus doesn't keep any history of what the original entry was. This is better since it eliminates the need to correct duplicates.
- Can we make changes to the list from List Manager during the call? No, the changes won't be reflected in the GUI. We can only refresh the info by logging out and then back into the conference. What does this mean for us? Any changes that need to be made during the call will be made directly from Pactolus. Any other changes we will make in List Manager at the end of the call, not during.

- Customer Care accesses LM from Citrix to correct and input info, but they print from the desktop application. For some reason, the desktop version does not accept corrections.
- We must keep text file capabilities in case OPAL is down. We can remove RTF. Updated 4/26/06: we can also remove text file since the Excel report will use the same purpose (in fact, the text files are transferred to Excel today).
- 4/26/06: The “Next” button in “List Manager” is supposed to take you to the next reservation number, but this functionality appears to have never been implemented because the “Next” button is never enabled. It will be removed.

Development Owner

The principal Developer(s) who coded this requirement.

Analysis

END.

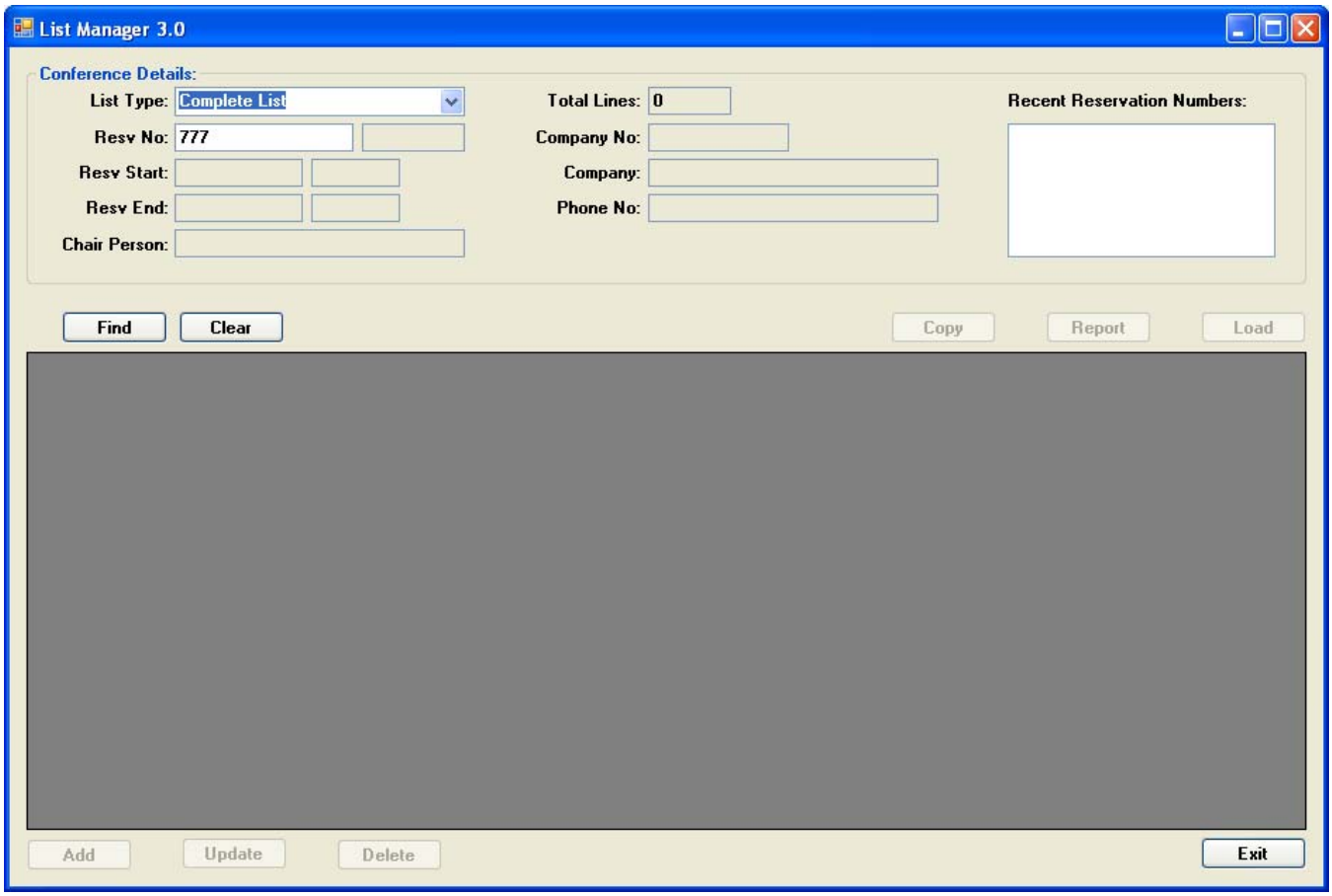
Appendix B

<List Manager Technical Specification Document>

Table of Contents

Dialogs / Field restrictions

List Manager (Main) Dialog.



Field Name: "List Type" Drop-Down

Maximum Length: N/A

Input Type: Drop-Down

Special Considerations: This field is always enabled and allows the user to change the participant list on the fly from "Attended," to "No Show," or "Complete List."

Field Name: "Resv No" TextBox

Maximum Length: Unlimited

Input Type: TextBox: Numeric (ONLY) Exception thrown otherwise.

Special Considerations: This field becomes disabled to the user when the "Find" button is selected. In order to reactivate this field, the "Clear" button must be de-pressed.

Field Name: "Resv Start" TextBox

Maximum Length: N/A

Input Type: TextBox

Special Considerations: This control is always disabled and displays the reservation's start time when List Manager has been populated. This field is converted from UTC to the current time zone of the conference.

Field Name: "Resv End" TextBox

Maximum Length: N/A

Input Type: TextBox

Special Considerations: This control is always disabled and displays the reservation's end time when List Manager has been populated. This field is converted from UTC to the current time zone of the conference.

Field Name: "Chair Person" TextBox

Maximum Length: N/A

Input Type: TextBox

Special Considerations: This control is always disabled and displays the chair person's full name when List Manager has been populated.

Field Name: "Total Lines" TextBox

Maximum Length: N/A

Input Type: TextBox

Special Considerations: This control is always disabled and shows the total number of participants in the participants data grid when List Manager has been populated.

Field Name: "Company No" TextBox

Maximum Length: N/A

Input Type: TextBox

Special Considerations: This control is always disabled and shows the chair person company Id when List Manager has been populated.

Field Name: "Company" TextBox

Maximum Length: N/A

Input Type: TextBox

Special Considerations: This control is always disabled and shows the chair person company when List Manager has been populated.

Field Name: "Phone Number" TextBox

Maximum Length: N/A

Input Type: TextBox

Special Considerations: This control is always disabled and shows the chair person phone number when List Manager has been populated.

Field Name: "Recent Reservation Numbers" ListBox

Maximum Length: N/A

Input Type: ListBox

Special Considerations: This control holds a user-selectable list of reservation numbers that were used during this session of List Manager. When List Manager is closed, the list is cleared.

Field Name: "Find" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will populate the List Manager application with the current reservation number. When selected, "Resv No" control becomes disabled.

Field Name: "Clear" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will clear the List Manager application of previous reservation information. When selected, "Resv No" control becomes enabled; "Load," "Report," "Add," and "Update" controls are disabled.

Field Name: "Copy" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will prompt the "copy to" dialog.

Field Name: "Report" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will prompt the report dialog.

Field Name: "Load" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will prompt an open/save dialog to select a flat .xls file to import into the List Manger application.

Field Name: "Add" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will prompt the add/update dialog with a blank template allowing the user to add a participant to the participant data grid.

Field Name: "Update" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will prompt the add/update dialog with the participant info. that was selected from the participant data grid.

Field Name: "Delete" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will delete a participant from the participant data grid and database.

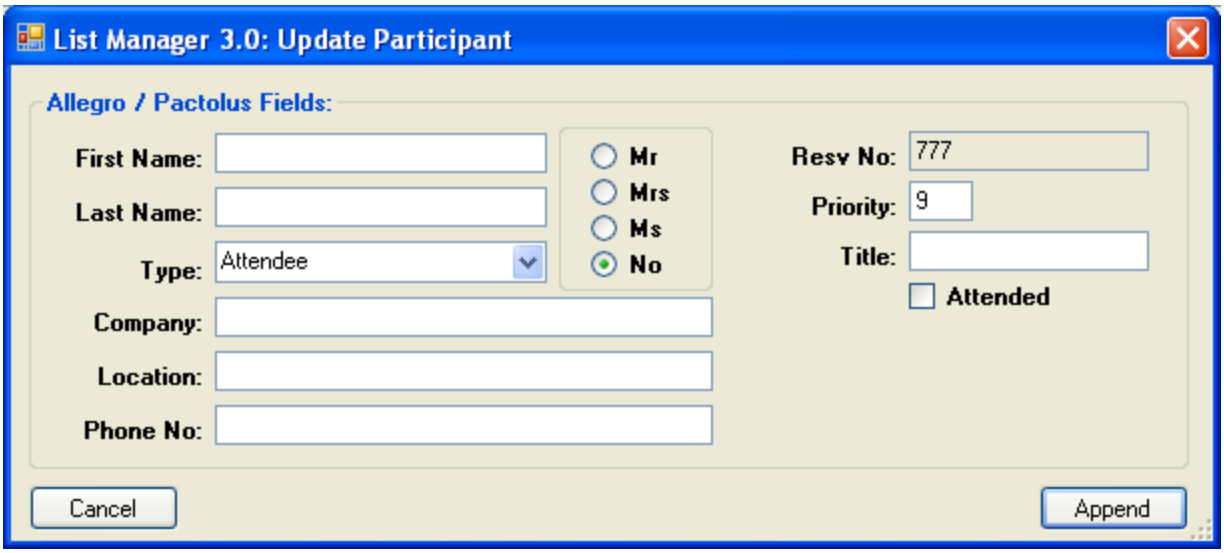
Field Name: "Exit" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will exit the application completely.

List Manager (Add/Update) Dialog.



Field Name: "First Name" TextBox
Maximum Length: 30
Input Type: TextBox
Special Considerations: This field stores the first name of the participant to attend. This field is required for the "Append" button to be enabled.

Field Name: "Last Name" TextBox
Maximum Length: 30
Input Type: TextBox
Special Considerations: This field stores the last name of the participant to attend. This field is required for the "Append" button to be enabled.

Field Name: "Company" TextBox
Maximum Length: 30
Input Type: TextBox
Special Considerations: This field stores the company name of the participant to attend.

Field Name: "Sir Name" RadioGroup
Maximum Length: 30
Input Type: RadioGroup
Special Considerations: This field stores the sir name of the participant to attend.

Field Name: "Type" Drop-Down
Maximum Length: 30
Input Type: Drop-Down

Special Considerations: This field stores either "Attendee" or "Moderator."

Field Name: "Company" TextBox
Maximum Length: 30
Input Type: TextBox
Special Considerations: This field stores the company name of the participant to attend.

Field Name: "Location" TextBox
Maximum Length: 30
Input Type: TextBox
Special Considerations: This field stores the location of the participant to attend.

Field Name: "Phone No" TextBox
Maximum Length: 30
Input Type: TextBox
Special Considerations: This field stores the phone number of the participant to attend.

Field Name: "Resv No" TextBox
Maximum Length: N/A
Input Type: TextBox
Special Considerations: This field is always disabled and shows the current reservation number to be modified.

Field Name: "Priority" TextBox
Maximum Length: 1
Input Type: TextBox (Numeric ONLY)
Special Considerations: This field saves the priority of the participant to attend.

Field Name: "Title" TextBox
Maximum Length: 10
Input Type: TextBox
Special Considerations: This field saves the title of the participant to attend.

Field Name: "Attended" Checkbox
Maximum Length: N/A
Input Type: Checkbox
Special Considerations: If checked, the participant will show as attended the conference, otherwise, the participant status is updated as a "No Show."

Field Name: "Append / Update" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will display "Append" to add a participant to the existing participant list or "Update" if an existing participant is to be updated.

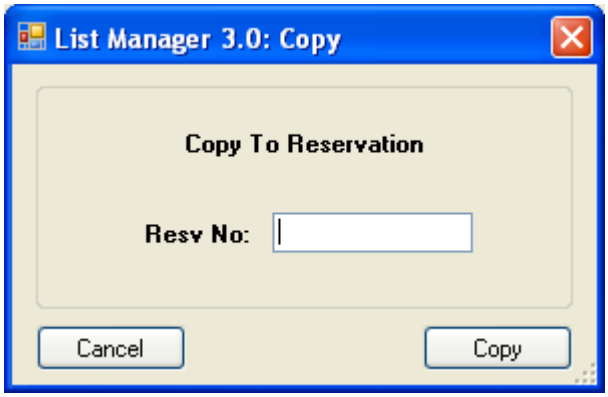
Field Name: "Cancel" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will close the add/update dialog without applying any changes.

List Manager (Copy To) Dialog



Field Name: "Resv No" TextBox

Maximum Length: N/A

Input Type: TextBox (Numeric ONLY)

Special Considerations: This field corresponds to the destination reservation number that the current participant list will be uploaded to.

Field Name: "Copy" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will copy the current reservation participant list to another reservation number. When the "Copy" button is selected, the destination reservation number is checked to verify that the destination reservation number is of the same company as the source reservation number. If the companies differ, a warning is thrown, but the user is allowed to proceed with the copy anyway.

Field Name: "Cancel" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will close the add/update dialog without applying any changes.

List Manager (Delete) Dialog

Field Name: "Yes" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will delete the selected participant from the participant data grid and DB.

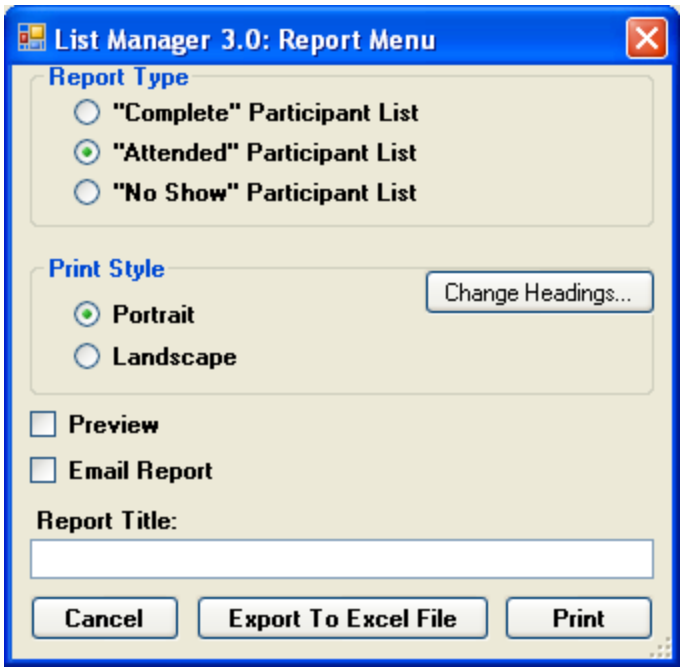
Field Name: "No" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will close the deletion dialog without changes to the participant list.

List Manager (Report) Dialog



Field Name: "Report Type" RadioGroup

Maximum Length: N/A

Input Type: RadioGroup

Special Considerations: If "Complete Participant List" is selected, the spreadsheet is generated with the complete participant list. If "Attended Participant List" is selected, the spreadsheet is generated with ONLY the attended participant list. If "No Show Participant List" is selected, the spreadsheet is generated with ONLY the no show participant list.

Field Name: "Print Style" RadioGroup

Maximum Length: N/A

Input Type: RadioGroup

Special Considerations: If "Portrait" is selected, the spreadsheet is generated with the portrait view. If "Landscape" is checked, the spreadsheet will be generated in a landscape view.

Field Name: "Preview" Checkbox

Maximum Length: N/A

Input Type: Checkbox

Special Considerations: If checked, causes the excel spreadsheet to be opened through the "Export To Excel File" process.

Field Name: "Email Report" Checkbox

Maximum Length: N/A

Input Type: Checkbox

Special Considerations: If checked, the user will be prompted for an email address during the "Export To Excel File" process.

Field Name: "Report Title" TextBox
Maximum Length: None
Input Type: TextBox
Special Considerations: This field saves the user-selected "Report Title" for generating a report.

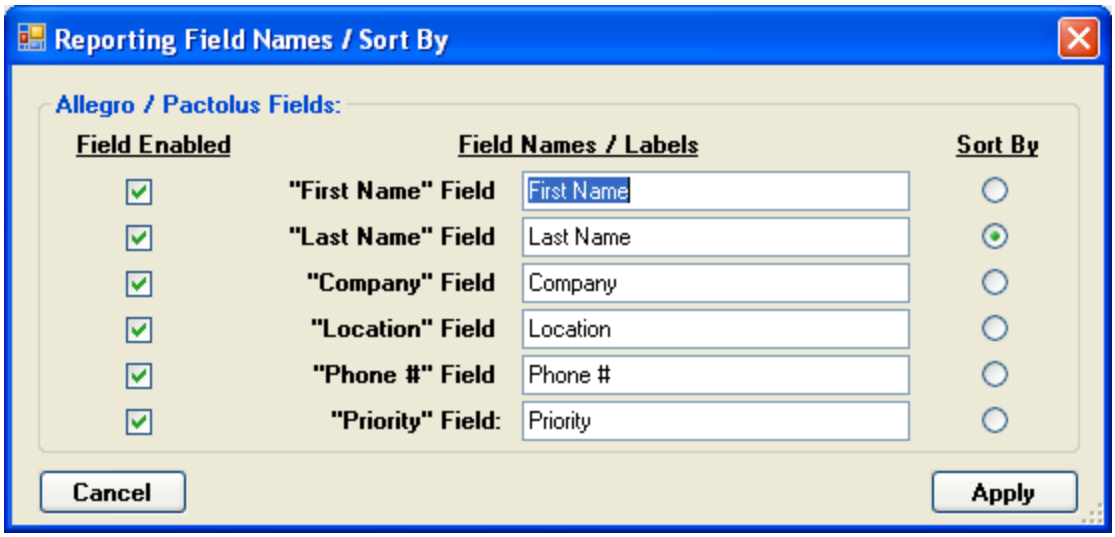
Field Name: "Change Headings…" button
Maximum Length: N/A
Input Type: Button
Special Considerations: This button will display the "Change Headings" dialog.

Field Name: "Cancel" button
Maximum Length: N/A
Input Type: Button
Special Considerations: This button will close the "Report" dialog without generating a report.

Field Name: "Export To Excel File" button
Maximum Length: N/A
Input Type: Button
Special Considerations: This button will export the participant list to an excel spreadsheets with the selected options from the "Report" dialog.

Field Name: "Print" button
Maximum Length: N/A
Input Type: Button
Special Considerations: This button will print the excel spreadsheet report to the default printer.

List Manager (Report Headings) Dialog



Field Name: "Sort By" RadioButton(s)

Maximum Length: N/A

Input Type: RadioButton

Special Considerations: These radio buttons cause the report to be sorted by the corresponding field. Only one radio button can be selected at a time.

Field Name: "Field Enabled" CheckBox(s)

Maximum Length: N/A

Input Type: CheckBox

Special Considerations: These checkboxes enable the corresponding field textboxes and radio buttons.

Field Name: "Field Names / Labels" TextBox(s)

Maximum Length: 30

Input Type: TextBox

Special Considerations: These fields save corresponding field titles for excel reports.

Field Name: "Apply" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will change the field headings for the report, but NOT change anything in the DB.

Field Name: "Cancel" button

Maximum Length: N/A

Input Type: Button

Special Considerations: This button will close the "Report Headings" dialog without generating a report.

Dialog PROCESSES

"Find" Process: (Populates List Manager with reservation info.)

Enter Resv No: → Select "Find" Button. → END.

"Copy" Process: (Copies current participant list to new reservation number.)

Find Process + → Select "Copy" Button. (Prompt "Copy" Dialog) → Enter Resv No: → Select "Copy" Button. → Return to "Main" Dialog. → END.

"Load" Process: (Loads a flat file into the participant list data grid and DB.)

Find Process + → Select "Load" Button. (Prompt Open/Save Dialog) → Enter Open Location.→ Select "Open" Button. → Return to "Main" Dialog. → END.

"Add / Update" Process: (Add / Update an existing participant within the participant data grid / DB)

Find Process + → Select "Add / Update" Button. (Prompt Add / Update Dialog) → Enter Participant Information→ Select "Append / Update" Button. → Return to "Main" Dialog. → END.

"Delete" Process: (Delete an existing participant from the participant data grid / DB)

Find Process + → Select participant to delete from participant data grid. → Select "Delete" Button. (Prompt "Are you sure" MessageBox) → Select "Yes" Button→ Return to "Main" Dialog. → END.

"Report" Process: (Generate a report in the form of an excel spreadsheet with chair / participant info.)

Find Process + → Select "Report" Button. (Prompt Report Dialog) → Select Reporting Options→ Select "Export To Excel" Button. → Return to "Main" Dialog. → END.

END.