Real-Time Rating Engine Interface

By

Erik Pfingsten

Submitted to
the Faculty of the Information Engineering Technology Program
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Engineering Technology

University of Cincinnati
College of Applied Science

June 2003

Acknowledgements

I would like to thank the following people for their assistance: Erik Zigman for suggesting the idea for this project and providing the guidance and resources necessary to complete it. Allan Logan for taking the time to explain in detail the many facets of Bluespring's Rating Engine and the Batch Framework. Brian Gehrich for setting up the equipment I needed to test the software. Karl Treier for providing general technical knowledge and answering every question I asked him.

Dedication

I would like to dedicate this project to my Fiancée, Heather. Without her support, I would never have the time to work on this project and finish school.

Table of Contents

List of Figures

Abstract

The Real-Time Rating Interface project is designed to relieve the operational problems inherent in running Bluespring Software's billing process as well as provide new business possibilities. Currently, Bluespring Software's billing process is a serial set of programs with data files as inputs and outputs. This makes it hard to scale the process across multiple processors or multiple machines. The real-time interface solves this problem by allowing a user to insert records from different sources into the bill process and have them run until the end (i.e., bill generation) without extensive manual operations. Decoupling the Rating Engine from the Batch process also enables the possibility of selling Rating Services to customers as a standalone application.

Real-Time Rating Engine Interface

1. Statement of the Problem

1.1 Background

Bluespring Software is a small software company with a comprehensive Billing Support and Operations Support Software suite, *PriorityCS*. This software focuses on the telecommunication market but is flexible enough to be used in a variety of businesses such as financial institutions or cable companies. The main components of the suite include an Order Entry/Account Management application with versions for the Web and Windows platforms, a sophisticated workflow system, a billing application and a management application used to configure the entire suite. The suite is designed to manage the entire customer contact lifecycle, from account signup to provisioning to account management to bill creation and presentation.

In traditional telecommunications billing applications, the introduction of a new product would mean that the telecommunications company would have to write a new order entry program, write the programs needed to provision the product and then write code to bill for the product (5). Bluespring's software architecture is based on metadata. This approach means that a customer can create a new product and set up everything needed to provision and bill for the product without having to re-engineer and re-code the order entry and backend billing system (for a detailed discussion of *PriorityCS'* Architecture see Appendix A.).

The billing application is actually a series of programs that run in a batch environment.[A] Account information and usage data[B] is pulled from a database, or other outside sources through a mediation program[C], as the first step of the process. All of the

usage is aggregated into an input file and that file is then fed into each program in the billing process. Each program in turn takes that file, processes each record and outputs another file that is fed to the next program in a serial fashion. This process happens on a predetermined cycle: usually daily, weekly, or monthly. One of the first programs in the series is the Rating Engine. The Rating Engine receives a Usage Record and calculates the amount to charge the subscriber for the event described.

1.2 Problem

There are two problems with the current method of storing usage until the end of a cycle and running in a batch setting. The first problem is that for some types of services, such as pre-pay calling cards, real-time rating is necessary to keep a customer from going over his/her allotment of minutes (5). If a company only runs a bill cycle every month, and only calculates the actual usage at that time, pre-paid services could be used indefinitely during that billing period. From a Customer Service perspective it is also beneficial to provide customers with as up-to-date information as possible. It becomes difficult for a customer to keep track of minutes used (on a cellular phone, for instance).

The second problem is that it is operationally difficult to run multiple instances of a program in a batch environment when there is only one input file (5). If the company running the software wanted to take advantage of multiple machines or multiple processors it would require the input file to be split up, and all the pieces to be kept track of while each job was running. Then all the various output files would have to be recombined for the next program in the billing application. This would be a hard task for a system operator to accomplish in the limited amount of time usually allocated to

generate bills (usually limited to a window of a few hours every week or month). If the usage data can be rated in real time, it would:

- Allow customers to keep better track of their usage.
- Enable a company to keep track of customer usage (and limit them to the total usage they are paying for per month).
- Enable end of month bills to be generated quicker (since the end of cycle bill process would only need to rate incremental charges, apply taxes, bill one-time charges and generate the bill).
- Enable easier scalability to multiple processors/multiple machines (since the process could be decoupled from a single data entry source).

A real time interface to the rating engine also presents a new opportunity to generate business for Bluespring. The rating process is one of the most complex parts of a billing system. Building a rating engine is not something that most telecommunications companies would want to undertake. In addition, Bluespring's metadata architecture and plugin design would allow the rating configuration components to be decoupled from the main application. This would allow Bluespring to offer its rating engine as a standalone service (5).

2. Description of the Solution

2.1 User Profile

Two types of users interact with the Real-time Rating interface: Configuration Designers and System Operators.

2.1.1 Configuration Designer

Most of the work handled by the Rating Engine Interface is automated, in the form of Web Service transactions; however, one type of user would use the Configuration utility. The Configuration Designer is responsible for two functions: Defining Usage

Records and Defining Rate Plans. The learning curve for both of these functions is long and requires someone with previous knowledge of the Telecommunications Rating area as well as detailed knowledge of the rating and billing plans and services offered by his or her company.

2.1.2 System Operator

The Rating Interface has the ability to have files sent to it to process large batches of Usage Records. An automated process could handle this or it could be accomplished by a system operator manually uploading files to the Web Service. This person needs to have basic computer skills. Manually uploading a file is accomplished by a small Web Service consumer application that is responsible for letting the user browse to the file and then uploading it.

2.2 Design Protocols

This project covers the technical areas of Programming, Networking and Database. The main impact of this project is in infrastructure and Application Programming Interface changes and does not incorporate a large amount of user interface. All components and applications for this project were written using Microsoft's VB.NET and utilize several of the key technologies of the .NET platform, including Web Services and .NET Remoting, Windows Services, Message Queuing and COM+ Services.

The Real-Time Interface replaces several of the infrastructure components that currently are used in the batch-processing environment. The normal flow of data in a batch cycle is from program to data file to program (see Figure 1.).

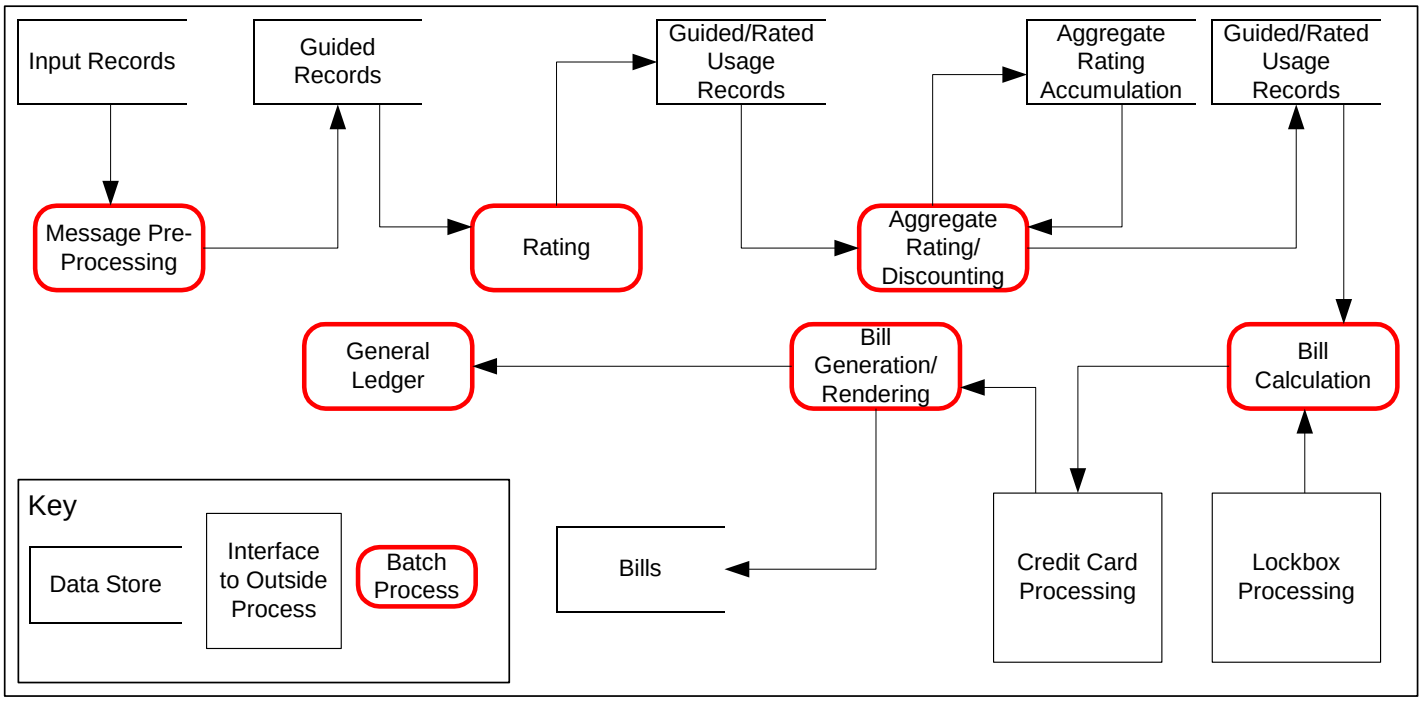

Figure 1. Abstract View of the Current Batch Process

The Real-Time Rating Interface takes the place of the Standard Batch Executable for Rating. The Standard Batch Executable has numerous hooks to file handlers, and error handlers that allow it to work with the batch cycle. The Real-Time Rating Interface provides a Web Service and remotely callable object (that can be called via a consumer application) to push Guided Usage records into a Message Queue (see Figure 2.). The Wrapper Application then reads those Usage Records and invokes the Rating Engine. The Wrapper Application pulls its database login information from an XML file. Since the database login is handled within the Wrapper Application, the Web Service does not have to handle database connection information. The Wrapper Application creates an instance of the Rating Engine object and uses that to process each record.

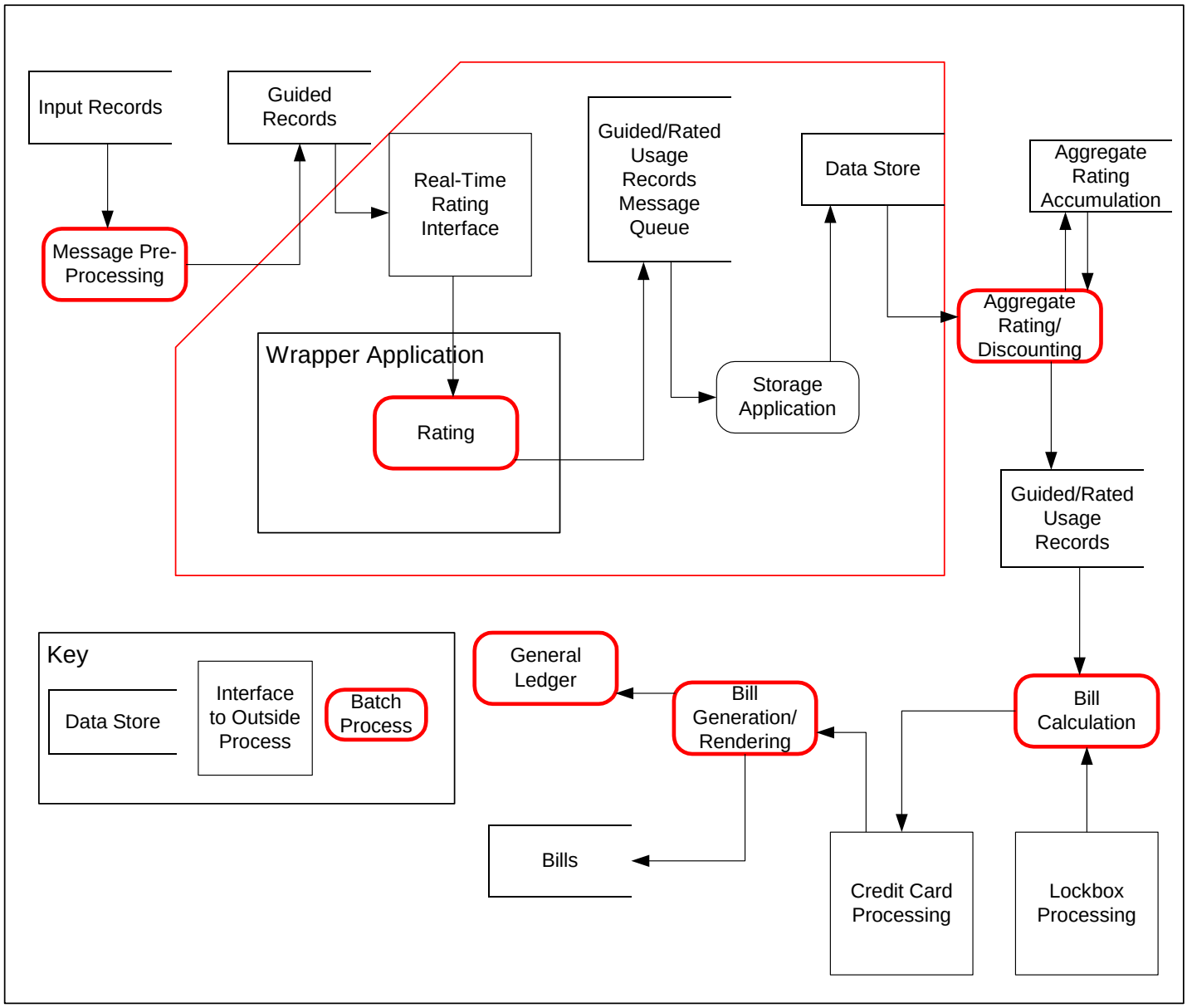

Figure 2. Abstract View of Batch Flow with Real-Time Rating

Once each record is processed, it is pushed back out to Message Queue and pushed into a SQL Server table (see Figure 3. for the table schema) via the Storage Application. The Web Service then queries against the Data Store to transmit the rated Usage Records back to the origination point of the original Usage Record. Once the Usage Records have been read back out by the Web Service, they are moved to an archive table. This ensures that the data table does not grow unchecked.

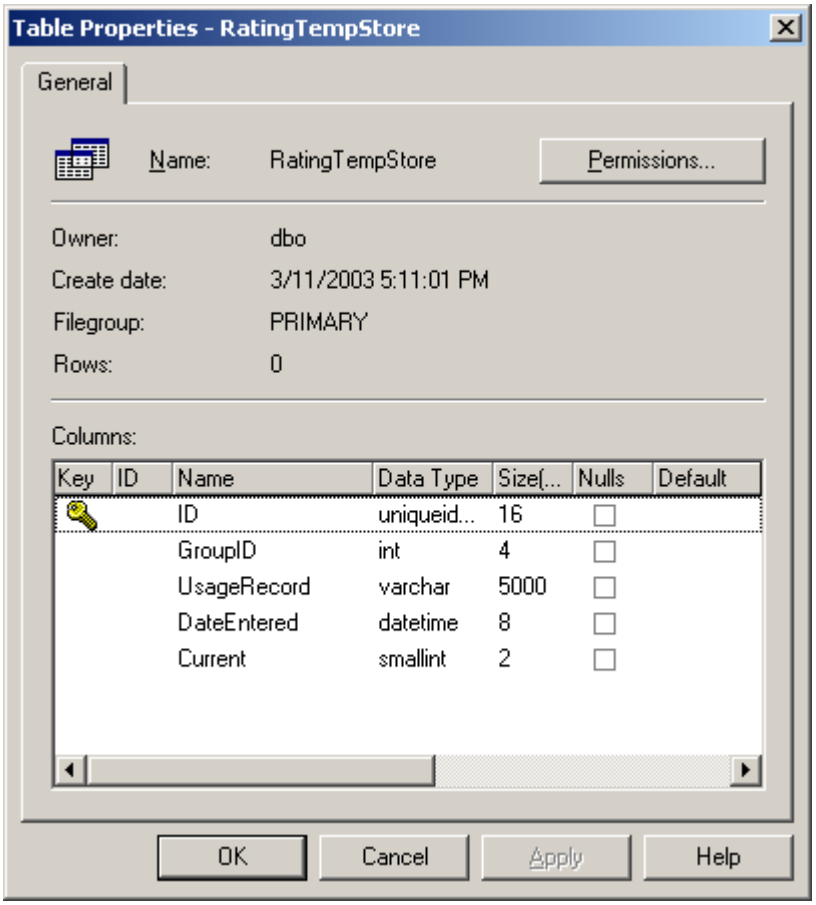

Figure 3. Database Table to Hold Rated Records

As shown in Figure 4., the Wrapper Application and Storage Application interact directly with Message Queue. Since .NET Web Services and .NET Remoting Objects do not maintain state, they both interact with Message Queue through an object that is hosted within Component Services. This allows the Message Queue object to maintain its connection to Message Queue during regular use, instead of being forced to close and reopen the Message Queue to send each individual message received through the service. The Message Queue Object opens its designated queue upon activation and maintains a reference to that queue during its lifetime within COM+.

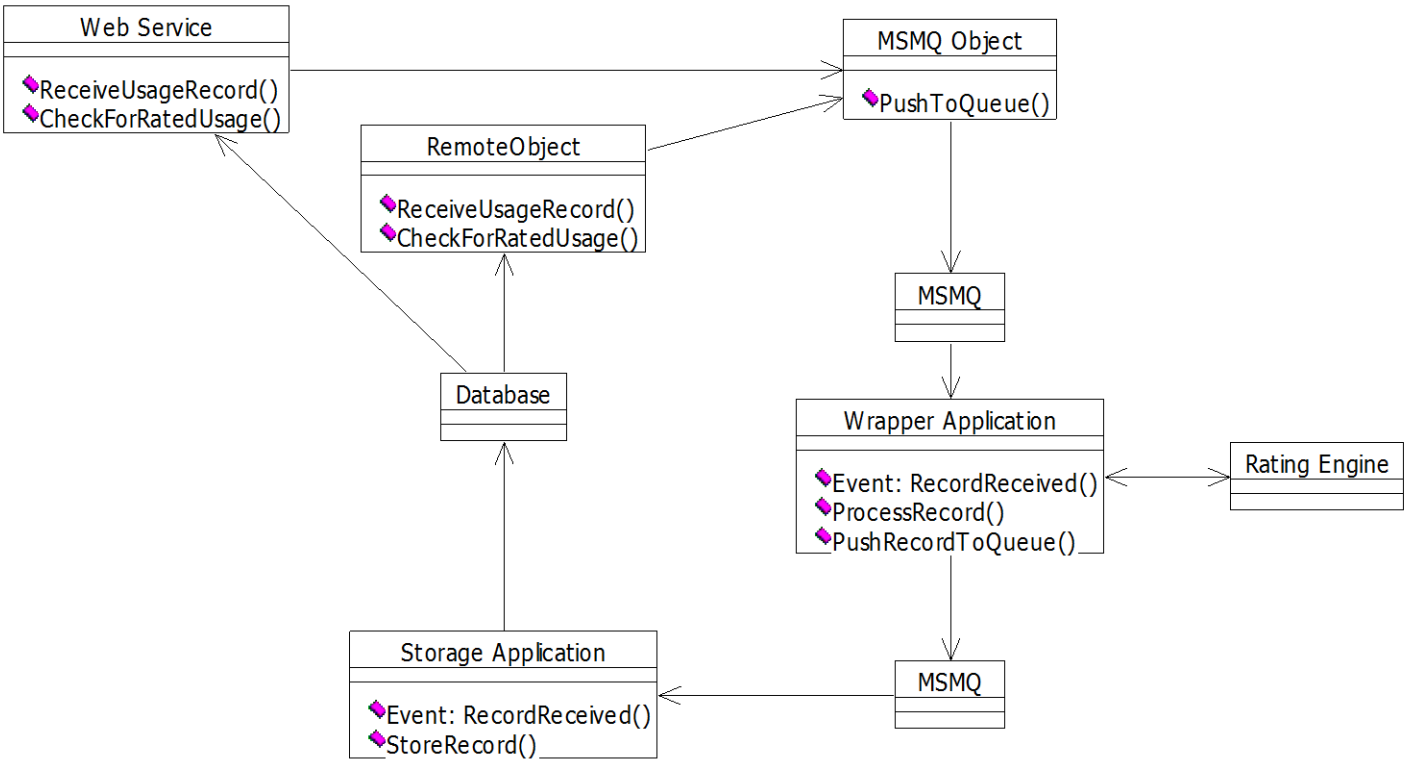

Figure 4. Object Diagram

3. Deliverables

There are five main components to the Real-Time interface in two areas:

3.1 Rating Engine Web Service Interface

The Rating Engine Web Service Interface is responsible for receiving individual Usage Records or groups of Usage Records and transmitting them to the Wrapper Application. The Web Service Interface consists of a Web Service and .NET Remoting Object used to receive Usage Records. The interface to Microsoft Message Queue is provided by a .NET Class Assembly hosted within a COM+ Application (Windows Component Services).

The Web Service provides a public interface that is platform independent due to its use of SOAP transactions. This means that a client can be designed in any

programming language on any operating platform to interact with the Web Service. Since it communicates over the web protocol (port 80), it is also firewall friendly, since port 80 (for web serving) is one of the few protocols allowed on most firewalls (2).

A .NET Remoting Object is also used to provide a public interface. This method is not platform independent or firewall friendly, however it does offer the chance for better performance since it can use direct TCP/IP to communicate between client and server (2).

The Message Queue object is hosted within Component Services to provide interaction between the public interface and Message Queue. Since the .NET Remoting Services and Web Services are inherently stateless, this provides a method for maintaining state (and the message queue connection) with object pooling. This allows the object to keep a reference to the actual message queue open, instead of having to open a reference to the message queue each time it needs to send a message (2).

3.2 Wrapper Application

The Wrapper Application is a .NET Service that monitors the message queue and passes each Usage Record into the Rating Engine Object. The Rated Usage Record is returned to the Wrapper Application, which then pushes the rated Usage Record back to another Message Queue. The Storage Application reads this queue and pushes the record into semi-permanent storage in the database. From there the rated Usage Records can be handled in two different ways. For now, the messages stay in the table until it is queried by the Web Service again to retrieve records. This is how the Interface currently operates as a standalone service. Another possibility is that instead of being pushed into the database at all, the rated Usage Records could be pushed out to a file to be picked up by

the next program in the batch cycle. Though this functionality is not implemented in this project, it would not be hard to add this ability.

4. Design and Development

4.1 Budget

Although there weren't any hardware or software purchases for this project (all development/testing took place with existing hardware and software), the approximate costs for the hardware and software in a professional setting is shown in Figure 5.

2 Dell Dimension 8250 workstations (Pentium 4-2.4 GHz Processors, 512 MB RAM)	$1,500 x 2 = $3,000 (www.dell.com)
2 Dell PowerEdge 1600 SC Servers (2 Pentium 4-2.0 GHz Xeon Processors, 1 GB RAM)	$2,000 x 2 = $4,000 (www.dell.com)
Microsoft Visual Studio.NET	$1,031 (www.buy.com)
Microsoft SQL Server 2000 (10 client license)	$2,061 (www.buy.com)

Figure 5. Hardware/Software Budget

4.2 Timeline

The timeline for this project was as follows:

- February 11 – 19: Technical Design of project
- February 19 – 25: Programming of MSMQ Interface component
- February 26 – March 4: Programming of Web Service
- March 5 – 12: Unit Testing Web Service and MSMQ Component
- March 14 – 20: Programming .NET Remoting Object
- March 21 – April 11: Initial Programming of Wrapper Application
- April 11 – 25: Unit Testing Wrapper Application
- April 25 – May 5: Programming of Storage Application

- May 5 – May 10: Unit Testing Storage Application
- May 10 – May 17: Web Service Method to retrieve records
- May 17 – May 20: Unit Testing Retrieve Method
- May 20 – May 25: QA Testing

5. Proof of Design

This section will show how the project met the goals of this project. The first part of the project was the actual Web Service Interface. Using .NET Web Services, a public interface that is not dependent on any one platform was created. Since Web Services are invoked via SOAP calls (see Figure 7.), a consumer can be written in virtually any language. Due to the ease of invoking Web Services from .NET, I wrote a consumer using VB.NET (Figure 6.). Using this client, I successfully connected to the Web Service to send the individual Usage Records. The first method call of the Web Service is

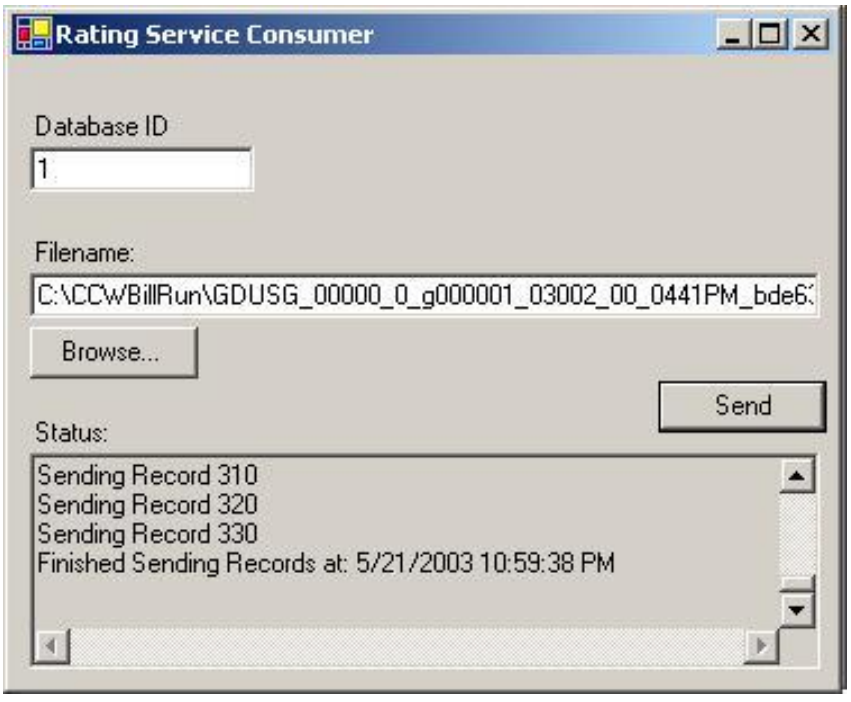

Figure 6. Usage Record Web Service Client

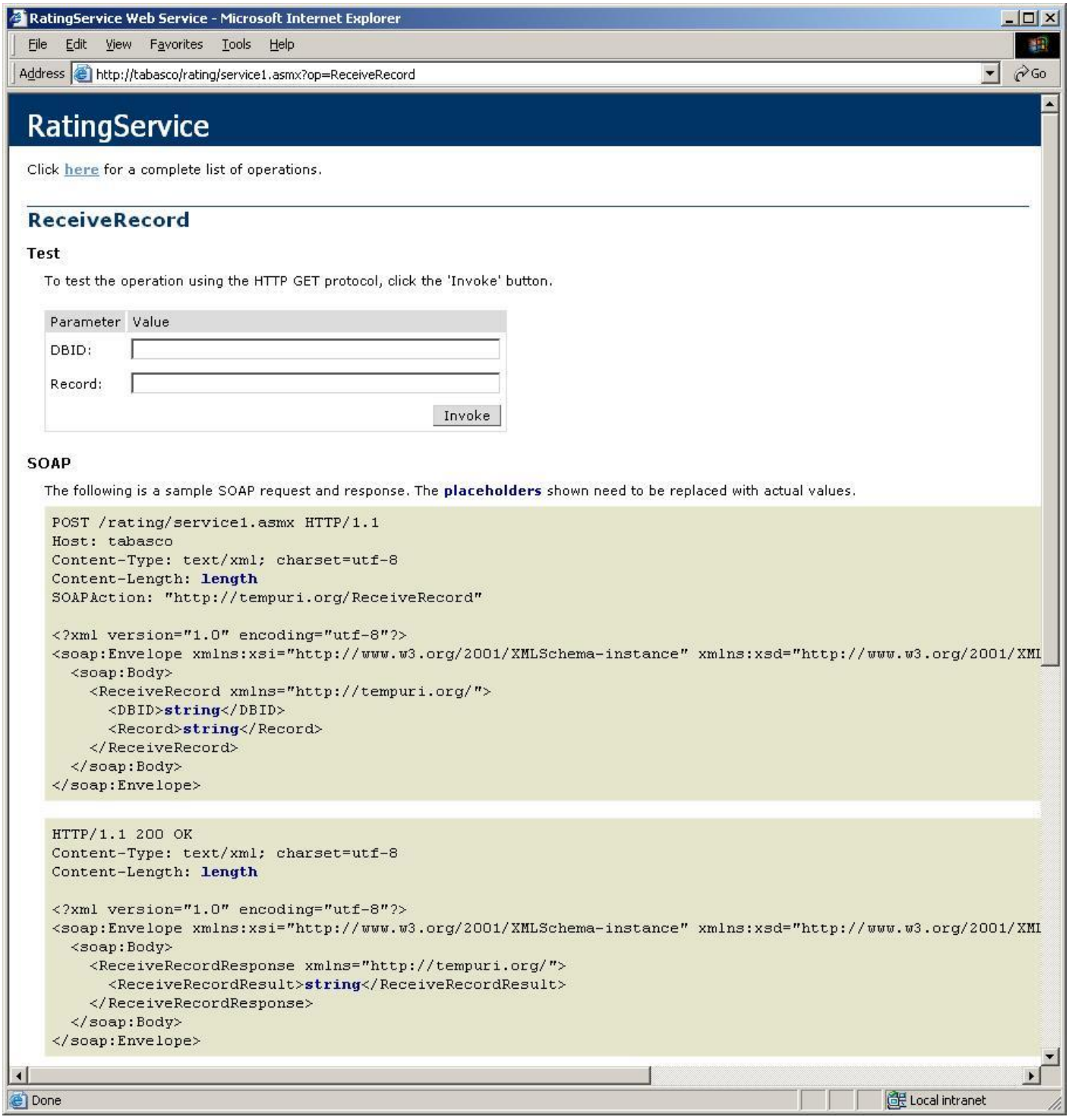

Figure 7. Web Service SOAP Transaction

ReceiveRecord. This method creates a MessagePump object to push the records into Message Queue. The Web Service successfully published each Usage Record into the

correct Message Queue (Figure8. and 9.) where it was processed by the Wrapper

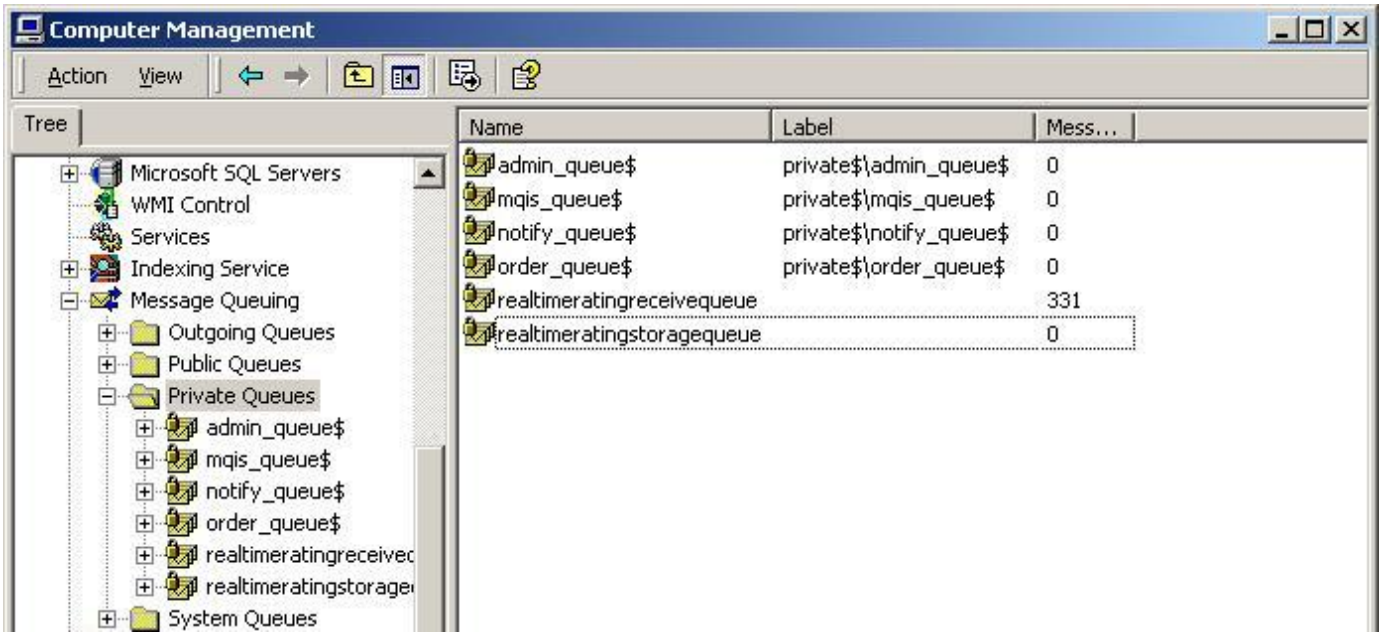

Figure 8. Message Queue with Usage Records

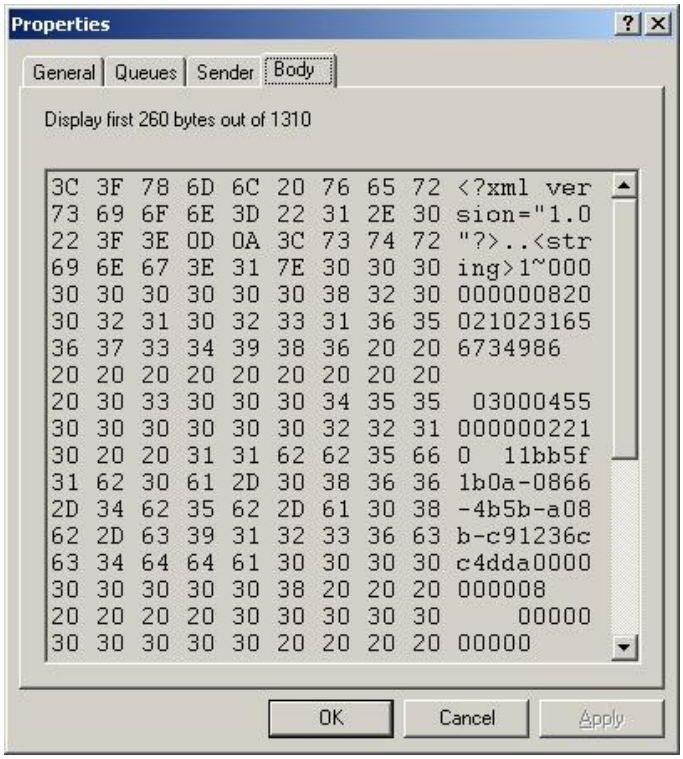

Figure 9. Format of the Usage Record Message within Message Queue

Application (Figure 10.). As each record was processed by the Wrapper Application, a variable number of rated Usage Records were output to the storage queue. Each message output to the storage queue was then processed by the Storage Application and a record was inserted into the database (Figure 11.).

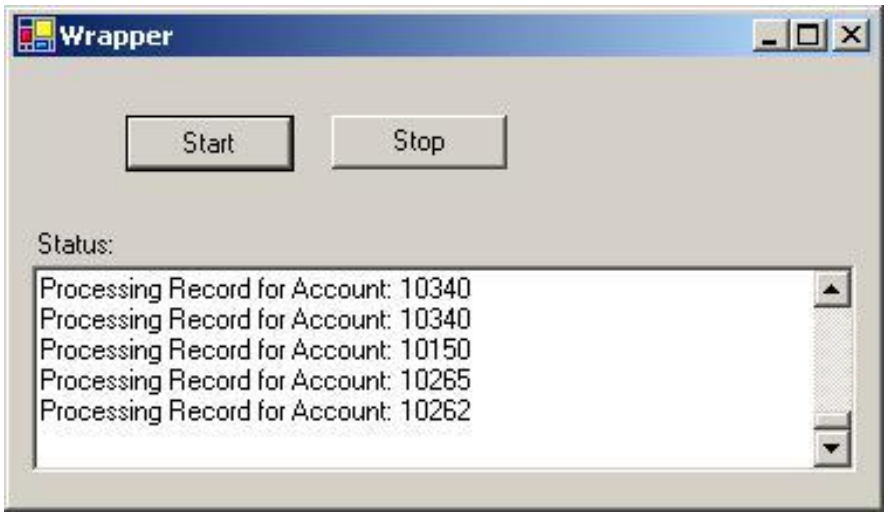

Figure 10. Demo Version of the Wrapper Application

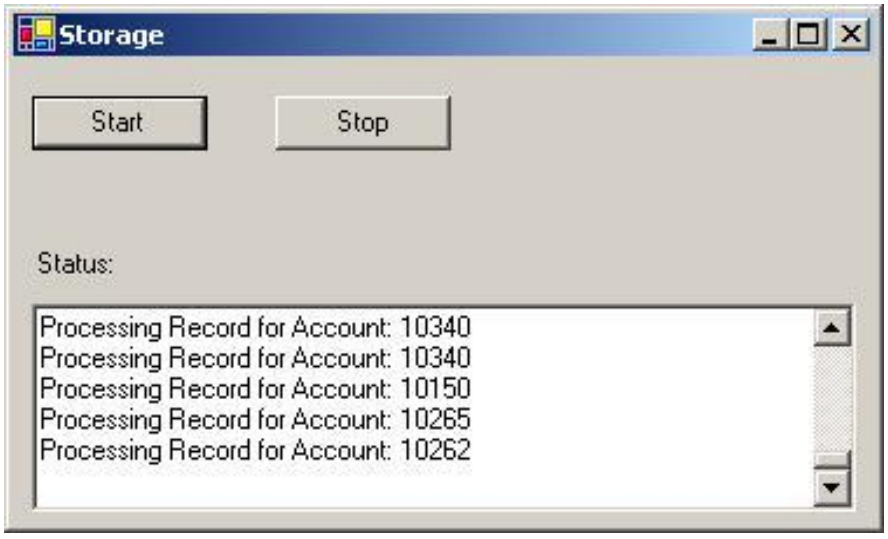

Figure 11. Demo Version of the Storage Application

Once the records are inserted into the database, they can be retrieved by the second Web Service Method, GetUsage (Figure 12.). This method call receives a Database ID and

Account ID and retrieves any records from the database with a matching Account ID (Figure 13).

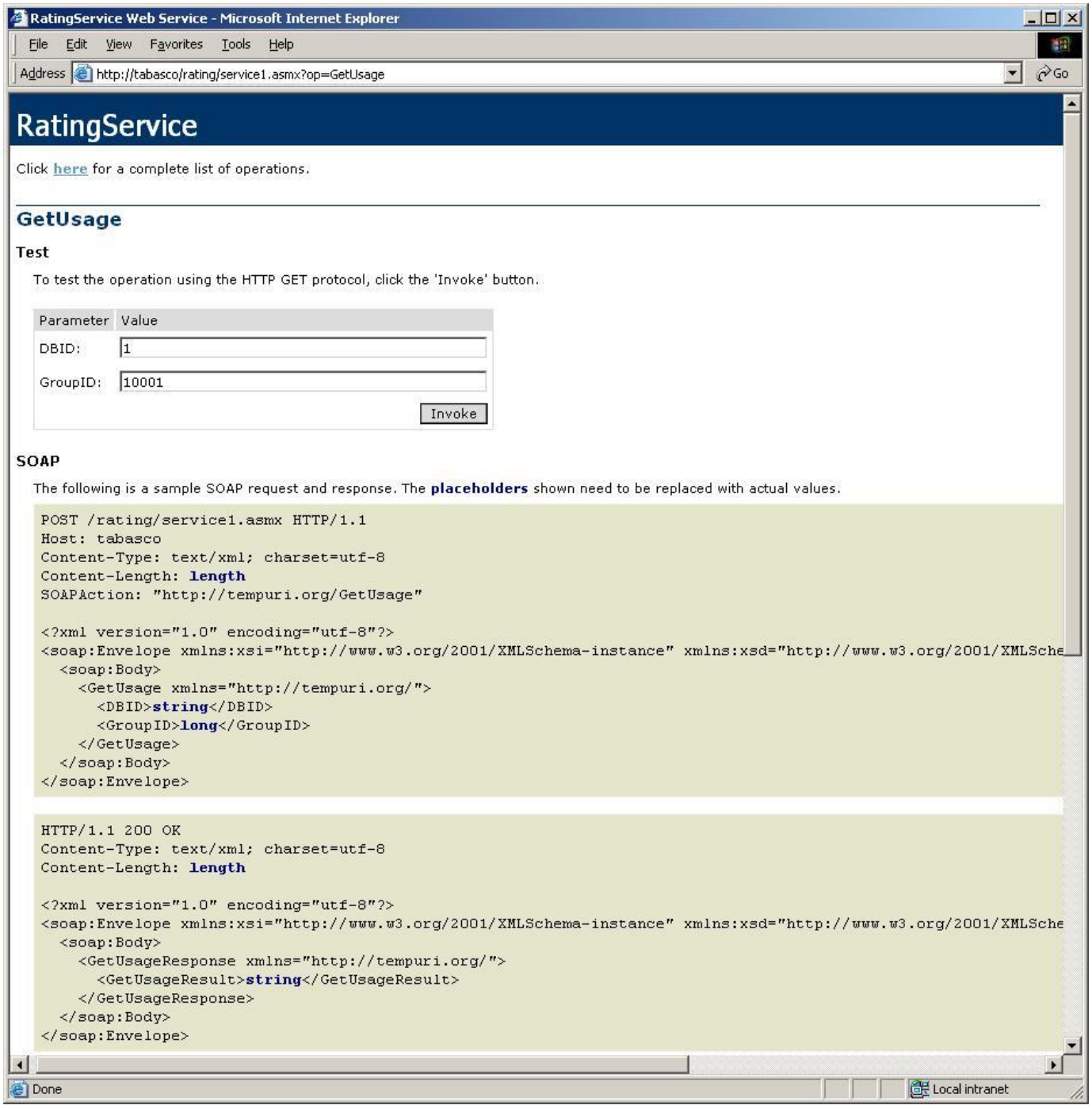

Figure 12. SOAP Calls to Retrieve Rated Records with the GetUsage Method

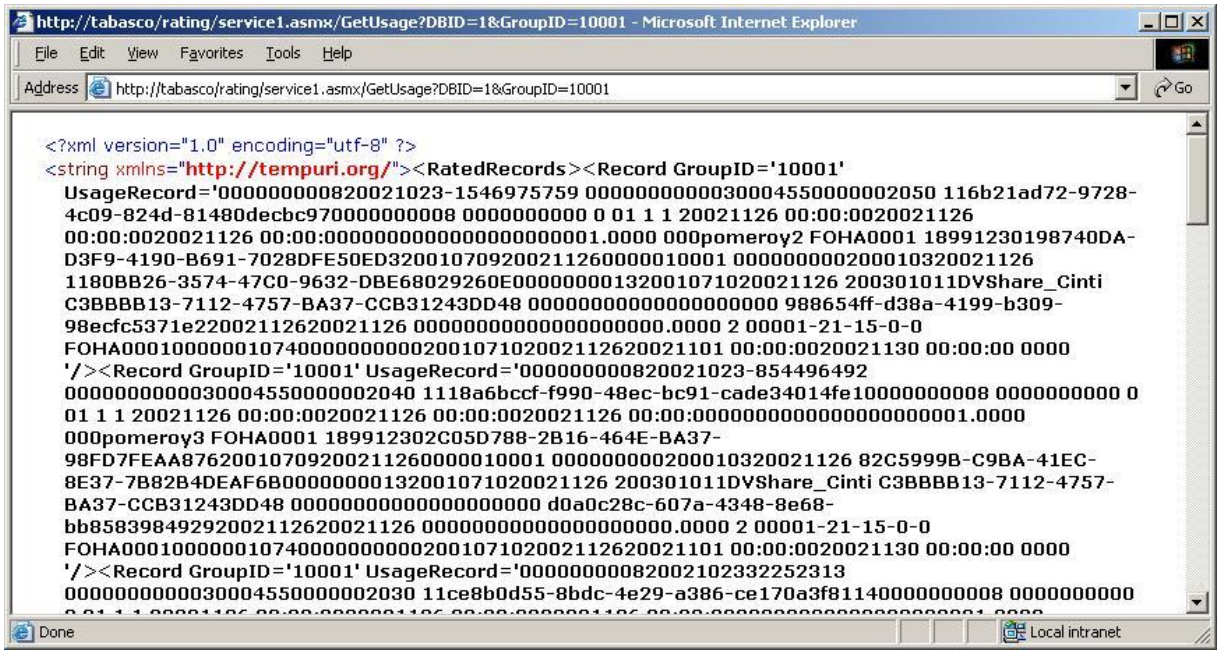

Figure 13. Results from the GetUsage Method Call

6. Testing

Testing for this project happened in two phases. Unit testing took place throughout the development process as each component was created. This was a necessity to ensure that each component worked as expected and that any minor syntax issues and hard crashes were fixed quickly. All unit testing took place on the original development machine (a Pentium 3 800MHz laptop with 512 MB RAM).

QA testing took place on two Dell Dimension Desktop computers with the Dell Inspiron Laptop serving as the Database server. Both computers were configured as Windows 2000 servers with Internet Information Services and Microsoft Message Queuing installed. One of the computers was also running SQL Server 2000. Both Machines have Pentium 3 500 MHz processors with 256 MB RAM. The Inspiron was used as the Database server (instead of the Dell Dimension) to more closely simulate a real world environment where the database would be on a dedicated server.

The purpose of the QA testing phase was to ensure that all components of the Real-Time Rating Interface worked properly outside of the development environment. Installing the .NET components involved copying the executable files and configuration files to the proper place on the machine. The MessageQueue object also had to be registered with Component Services with the REGSVCS command. The Web Service files had to be copied to the Inetpub\wwwroot directory and set up as an IIS Application to work properly. Installing the .NET components was the easy part.

A larger problem arose due to the extensive use of Visual Basic 6 objects and the need to use .NET Interop. When creating each project in .NET, copies of the required VB 6 components are copied to the directory where the final executable or assembly is compiled. This does not always ensure that it will work however. After attempting to run the Real-time Interface unsuccessfully, I had to run the Installation program for *PriorityCS* to get the Visual Basic Components installed correctly. Once this was done, the entire Real-Time Interface process was successfully executed.

A third phase of testing was originally planned to measure performance. This phase would have taken place on actual server grade computers to test the real-world performance level of the entire project. Unfortunately, due to budgetary contractions at Bluespring Software, spare hardware was not available. It was determined that testing the project during "downtime" was too big of a risk on production hardware and a decision was made that performance levels could be sufficiently extrapolated from the QA Testing environment.

The original performance goal was to rate records at the rate of 100 per second per CPU. This was determined to be the rate that previous performance testing on

Bluespring's Rating Engine had achieved (4). Since server grade hardware was unavailable, the performance testing was performed on the Dell Dimension computers and comparable numbers were extrapolated to approximate server level hardware. The performance of the system as a whole was somewhat disappointing. While the Web Service was able to receive records at the rate of approximately 30 per second (with one consumer), the overall performance fell to approximately 8 records per second including rating, storage, and retrieval.

There are several possible reasons to explain this performance. The first is obviously the hardware. The original measurement of 100 records per second was achieved on high performance server hardware at a testing lab in Cincinnati. The specifications for the Rating server included a server with 2 GB of RAM, a Pentium 4 XEON CPU and high speed SCSI hard disks. The Database in these performance tests was hosted on a 6 way Pentium 4 CPU server, with 4 GB of RAM and high speed SCSI hard disks. As well, the network they were running on had no other traffic to cause latency or bandwidth issues. The performance tests I ran were on desktop level machines running in the corporate LAN.

Another possible reason is also hardware related, but has to do with memory requirements for Message Queue. Since the message queues used were strictly memory based, 256 MB of RAM was probably insufficient to achieve optimal performance (3, p. 9). 256 MB RAM is really the minimum needed to run Windows 2000 Server and according to the Microsoft Performance Whitepaper, message rates as high as 22,000 per second when a system is configured with more RAM (3, p. 12).

A third possible explanation is the use of Interop to access the Rating Engine itself. In performance tests done by Microsoft, there was almost a 50% performance hit when using COM Interop in certain circumstances instead of native .NET code (1). Regardless of these explanations, further testing and code optimization is going to be performed to try to improve performance.

7. Conclusions and Recommendations

7.1 Conclusions

This project was created at the request of Erik Zigman (CTO, Bluespring Software) to fulfill two major goals. The first goal was to improve operational capabilities and scalability for the Bluespring Billing Process. This project demonstrates these improvements by using Microsoft's Message Queuing infrastructure and Windows Services for the batch application. The second major goal of this project was to provide a working example of a Real-Time Rating model that could be used as a prototype to sell service based rating services. The creation of the Web Service Interface to submit Usage Records and to retrieve them demonstrates this capability.

7.2 Recommendations

Though the result of this project (and the original goal) was a working prototype, more could be accomplished as a long-term goal. Selling Real-Time rating as a standalone service and the full operational improvements for the Billing Process could be handled as separate projects.

The separation of the Rate Page Designer and Usage Record Layout interfaces from the Administration Management Console would be necessary to sell Real-Time Rating as a standalone product. Currently the Usage Record formats are stored in the

database and cached to disk during execution. Selling Real-Rime rating as a standalone service would require that Usage Record object first look for the cached version on disk, thus allowing a customer to submit their Usage Record layout as a plain file. This would only make it necessary to connect to the database to get actual rating information.

Operational improvements could be had over the entire Billing Process by converting the current file based billing executables to .NET Windows Services using Message Queue. Though there are some programs in the Billing Process that require sorted records and would require storage in something other than Message Queue, the ability to scale across multiple machines outweighs this potential bottleneck.

The major hurdle to all this is, of course, performance. I do not believe it is impossible to achieve the 100 record per second goal. I believe that starting off with a working prototype is a major achievement to demonstrate the feasibility of the project and that performance results could readily be achieved through the use of server level hardware and the use of better performance metrics (such as the Performance Monitor tool) as a guide to optimizing the code.

Appendix A.

Bluespring Software

P·r·i·o·r·i·t·y·CS

Logical Architecture Overview

Release 1.6

Bluespring Software, Inc.
1128 Main Street, Ste 300
Cincinnati, Ohio 45202

Tel: 513.794.1764
Fax: 513.794.1724
Web: www.Bluespringsw.com

bluespring™ SOFTWARE

Bluespring Software, Inc.
www.Bluespringsoftware.com sales@Bluespringsoftware.com
Corporate Headquarters
Cincinnati
1128 Main Street
Suite 300
Cincinnati, OH 45202

1. Logical Architecture

1.1 Introduction to the Logical Architecture

Bluespring Software's Priority Communications Solution (Priority CS) system is built on cutting edge technologies, and employs leading software design philosophies such as object-orientation, distributed processing, and meta-data driven data stores.

2. Hardware and Software Development Trends

The software industry in general is moving in the direction of distributed component object models, following standards such as Microsoft's COM/DCOM and OMG's CORBA, and distributed processing systems. Regarding telecommunications customer care and billing systems, Bluespring Software is one of the companies leading the effort to deliver product based upon these new architectures and design principles.

The traditional approach to development of software applications were monolithic systems run on mainframe or UNIX variants. With the advent of PC technology the next iteration of software applications development was client server-type applications. The current phase can best be described as "n-tier" computing, which includes the abstraction of presentation, business logic and data management layers from each other in order to improve performance and reuse of requirements, design and application source code. The next phase appears to be distributed systems, which have been discussed for a long time and are just now coming to fruition due to the technologies being introduced to support n-tier systems. From messaging systems to distributed object models, technologies are easing the development of distributed systems. Some distributed processing advantages are also realized with clustering. True clustering has existed on the DEC VAX VMS architecture since the 1970's and has only recently been introduced with varying degrees of success on the client/server or PC computing platforms. The key benefits of clustering are performance and up-time/fail over.

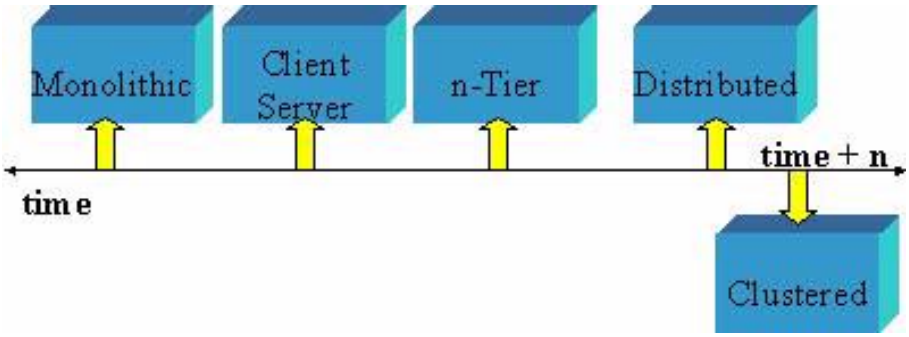

Figure 1 – Evolution of Software Application Development

3. Architecture and Design

Traditionally, systems were hard coded resulting in longer development times. These systems are typically mainframe based and are well represented in the RBOC world.

The next generation of systems was comprised of table driven systems. Most of the BACC vendors existing today employ this approach. These systems have features and functions based on table values, and are typically more flexible and easier to maintain. Key benefits include reduced time to market and reduced cost when compared to the legacy development world. More often than not, these systems were built on midrange or client server platforms.

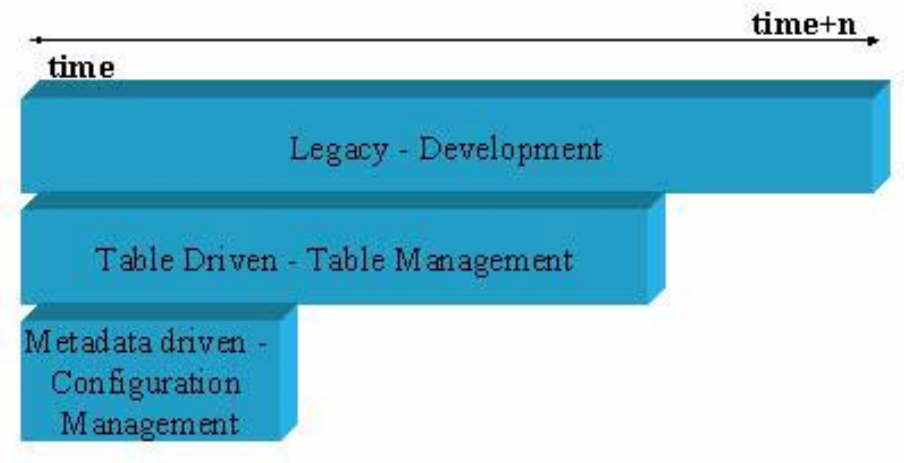

Figure 2 – Comparison of Time to Market

3.1 Defining Meta Data

Meta data, *n.* [from L. meta-; Gr. meta- and L. data] Data that characterizes other data in a reflexive way, e.g., data about data. Analogous to words about words. In data processing, it is definitional data that provides information about or documentation of other data managed within an application or environment. For example, meta data would document data about DATA ELEMENTS or ATTRIBUTES, (name, size, data type, etc) and data about RECORDS or DATA STRUCTURES (length, fields/columns, etc) and data about DATA (where it is located, how it is associated, ownership, etc.). Meta data may include descriptive information about the context, quality and condition, or characteristics of the data.

Meta-data driven systems are a paradigm shift from traditional Customer Care and Billing systems. Changes made to a definition system result in new features, new entry fields and functions. This shifts the world from a table management system to a more flexible configuration management system, resulting in reduced development time, and improved time to market. Time to implement features and enhancements in billing and customer care systems has made significant leaps with each advancement in application software.

As an example, adding a single discrete element, such as hair color, to a system and having it impact rating and bill format requires different change efforts on each system. The following table shows the effect on typical systems:

Legacy	*Table Driven*	*Meta Data*
Database change	Database change	Bill Format Changes
Database conversion	Database conversion	
User Interface Changes	User Interface Changes	
Rate System Changes	Bill Format Changes	
Bill Format Changes		

Figure 3 – Complexity of changes to add elements

Priority CS is an n-tiered distributed order entry, customer care, and billing platform employing relational databases, component applications, and Internet web technologies. The back office processes (message processing, rating, etc.) have been designed to fit within Priority CS' Messaging Architecture. Those functions are controlled and monitored via the Priority CS Operations Management System.

The system leverages the tiers or layers to abstract application functions from technical layers thus reducing dependence on any vendor. In addition, the n-tiered approach abstracts the presentation layer from the business logic, and the business logic from the database layer.

4. Priority CS Architecture, Release 1

A logical view of Priority CS' Architecture is presented in Figure 4. This section will describe each of the eight areas indicated in the diagram as they are used in Priority CS. The eight major areas of Priority CS are:

4.1. Relational Data Base Management System
4.2. Meta-data Repository
4.3. Object Layer
4.4. User Interface Components
4.5. User Application Layer
4.6. Process Flow
4.7. Messaging Layer
4.8. Operations Application Layer

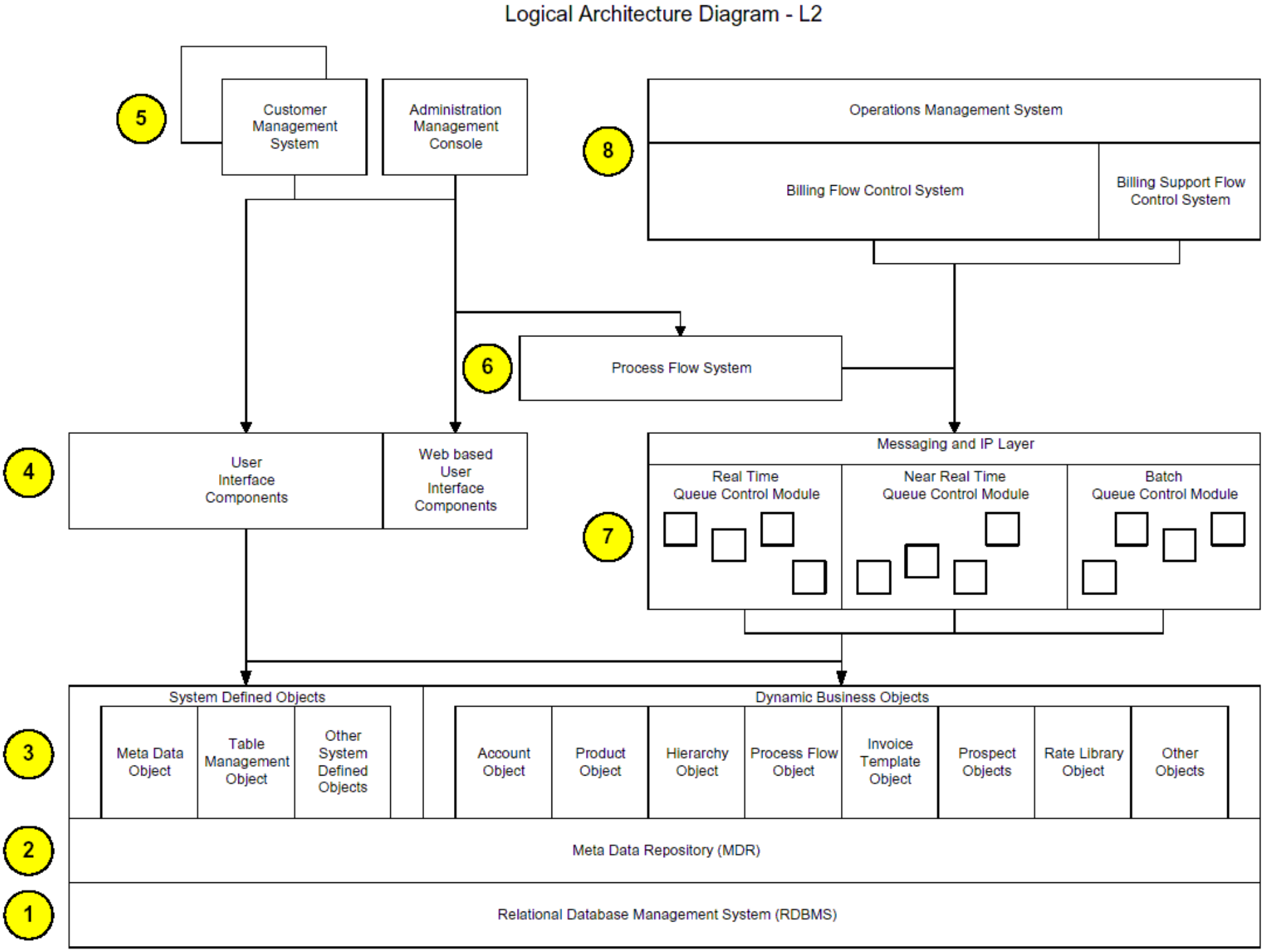

Figure 4 – Priority CS Logical Architecture

As shown in the diagram above:

4.1 Relational Database Management System (RDBMS)

Fundamental to Priority CS' architecture is the RDBMS. It acts as the repository for all information. Priority CS has been developed using Microsoft SQL Server version 6.5 or higher. However, extreme care was taken during the development to ensure that database portability would be maximized. To ensure this portability, Priority CS accesses the database using ODBC and has not made use of MS SQL specific triggers and stored procedures. As a result, porting to other ODBC compliant databases, such as Oracle, Informix and Sybase, should be accomplished with minimal effort and no application development or impact on business rules.

4.2 Meta Data Repository

Built on top of the RDBMS is Priority CS' meta-data repository. Priority CS derives flexibility from its meta-data repository. The goal of the meta-data is to reduce the frequency of changes to the underlying database structures and to provide flexibility for the applications and object layers. As presented above, meta-data is data about data. Therefore, the meta-data layer contains information (data) about the data contained in the RDBMS.

At a high level, the types of data within the repository are Content Data and Context Data. Content Data can best be described as the values of attributes, for example, 100 Main Street; John Smith; US$. Context Data are descriptions of the attributes. In the preceding sentence, 100 Main Street is the Content Data and Billing Address might be the Context Data; John Smith is Content Data and Account Name is Context Data; US$ is Content Data and Invoice Currency is the Context Data. Content Data is maintained by the Customer Management System. The Administration Management Console (AMC) provides a complete user interface and application for maintaining Priority CS' Context Data. The following paragraphs will attempt to describe Context Data in further detail.

Meta-data objects define the basic context for a set of content data. The meta-data repository defines dynamic business objects. Examples of business objects might include a business customer or a product. The meta-data repository defines the attributes (or pieces of information) to capture about each object, the organization of those attributes, and the relationship(s) of those attributes to other business objects. For example, the account name for a customer and the address for a customer may be attributes of business objects. Business objects are dynamic because Priority CS provides for the ability to create, change and remove attributes from existing objects as well as the ability to create new business objects to reflect changes is business needs, markets, customers and practices.

In addition, the meta-data also defines the business logic associated with each object and when necessary, each attribute. Business rules such as if the information is required, how the information should be formatted and edits required on each attribute and between attributes. Examples of business rules implemented in Priority CS can include any of the following:

- **Interface elements**
  - Field masks
  - Pick lists
  - Drop down calendars

- **Data Access**
  - Visibility at a system level
  - Visibility at a user-class level
  - Entry at a system level
  - Entry at a user-class level
- **Optional and required attributes**
- **JavaScript**
- **VBScript**

In addition, any or all of the attributes and business rules can be created, modified or removed during the lifetime of Priority CS' implementation.

In short, configuring the meta-data defines how the object layer and interface layer should interpret, process and present the business objects and its attributes described by the meta-data. This configuration is accomplished by using Priority CS' AMC.

Effective Date(s)	Data Type
Data Sequence	Related Image(s) (icon)
Name	Related Edits
Data Source(s) • User input • Table or Database • Externally sourced	User interface properties: • Visibility • Modifiable

Figure 5 – Examples of meta-data defined by Priority CS' AMC

4.3 Object Layer

Priority CS' Object Layer interprets and organizes the information stored in the Meta-data Repository. Priority CS' Object Layer contains System Defined Objects and Dynamic Business Objects. System Defined Objects are those entities that are required for Priority CS to function. Examples of system defined objects include the Table Management Object and the Meta-data Base Object. Dynamic Business Objects are instances of the Base Object that are created by processing the Meta-data Repository (Context Data). Application development is not necessary to create Dynamic Business Objects. The Base Object's interpreting the Context Data contained in the Meta-data Repository creates Dynamic Business Objects. Priority CS' Object Layer is in essence an extremely powerful and flexible data mediation layer between the meta-data layer and the user interface layer. This concept is further is illustrated in the diagram.

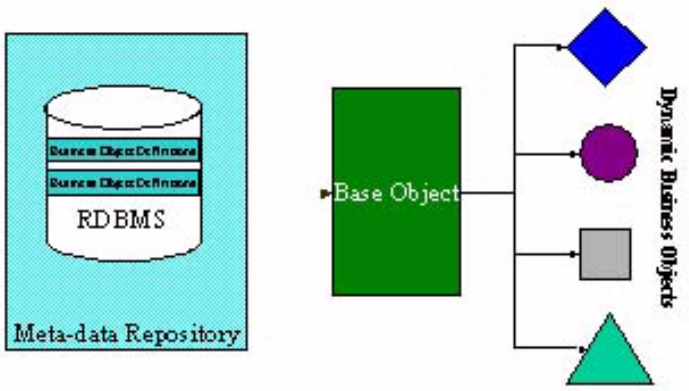

Figure 6 – Priority CS Object Layer

4.4 User Interface Components

The User Interface (UI) does not contain business logic; it merely determines the appropriate mechanism to display the data provided to it by the Object Layer. This layer consists of SMG developed components and controls to work with Priority CS' Object Layer. Examples of the components include a Web Browser, tree control and grid controls. These components are self-contained modules to provide a user interface for the object layer.

Priority CS' UI can be extended by the use of Extended Interface Components (EICs). EICs are used to provide additional functionality to display and capture attributes and information that are not easily accommodated by Priority CS' existing UI. One example of Priority CS' use of EICs is the display of rate plan information.

4.5 User Application Layer

The User Application Layer consists of two applications. These applications are Priority CS Customer Management System (CMS) and Priority CS Administrative Management Console. The AMC is used for configuring Priority CS Meta-Data as it relates to Dynamic Business Objects discussed earlier. In addition, the AMC is used to manage Users and User Classes, as well as WorkFlow Management. Priority CS CMS is an end-user application designed for Account Management, Order Entry, Customer Care, Remittance Processing and Task Management.

Priority CS' User Interface (UI) is a multiple document interface (MDI) application that creates instances of Dynamic Business Objects from the Object Layer and presents the information to the user. Priority CS UI adheres to Windows' standards for usability, appearance and commands. In addition, the user interface contains an embedded Telnet client and terminal emulators for 5250 and 3270 for access to existing legacy applications.

4.6 Process Flow System

The goal of Priority CS Process Flow modules is to increase operational efficiency, effectiveness and excellence through mechanization or automation of business activities involved in fulfilling a customer's request for service. While automation allows significant reduction of operations expenses, it also requires more up-front investment for both operations engineering efforts and inter-system interface specifications when compared to the more simplistic mechanization methodology. The benefit of automation is derived

from analysis across the system's life cycle and estimated operational savings. Typically, automated solutions can cost-justify (self-fund) within 6 months to one or two years of operational implementation.

Priority CS' Process Flow diagrams detail a logical set of tasks that are required to fulfill a service request for a customer. Using Priority CS' AMC, Process Flows are created using the point and click interface. The tasks are organized to support the desired business process. Dependency between tasks is identified using a graphical tool to link predecessors and successor tasks. Tasks can be executed in parallel or serially. Each task is then assigned properties such as duration, user or user class assignment, escalation levels and escalation classes. Tasks may be automated or manual and support is provided for logical branching. Information captured during order entry and order processing is available to the individual task that requires it, providing a self contained and informed action for the process to perform.

4.7 Messaging Queues

Priority CS allows for custom-developed or user supplied applications to be accessed in real-time, near real-time or in a batch-processing mode. Monitoring a specific port for network events can accommodate real-time processing. Near real-time processing is supported with messaging technologies such as IBM MQ Series, MS Message Queue Server and MS Transaction Server. Applications can be developed to access external data stores for processing within Priority CS, extend application functionality such as event processing, rating and discounting or to update external applications with Priority CS information. Applications can be developed in the language of choice and communication between Priority CS Messaging Layer can occur using COM or CORBA Objects.

Put another way, Priority CS' Messaging Layer provides a container into which additional applications can be inserted. For example, an application could be deployed into the environment that performs the business logic based on the specific event type.

4.8 Operations Management System

Priority CS' Operations Management System is a tool through which Operations Staff can execute and manage Priority CS application execution. The Operations Console monitors the throughput of the active Messaging Queues and provides the necessary tools to manage system processes (such as server groups) and business applications (such as Rating, Bill Processing, etc.). The Operations Management System contains distinct components, which are the Billing Flow Control System and the Billing Support Flow Control System. The Billing Flow Control System manages billing related applications and the Billing Support Flow Control System manages non-billing applications such as processes necessary for fulfillment of customer requests. Examples of these processes include welcome e-mail, letters, etc.

5. Priority CS Integration, Release 1

Key to any enterprise-class or carrier-class application is its ability to interface with existing processes, people, technology and applications. Priority CS components were designed to support those integration requirements. This section will discuss integration possibilities into existing environments.

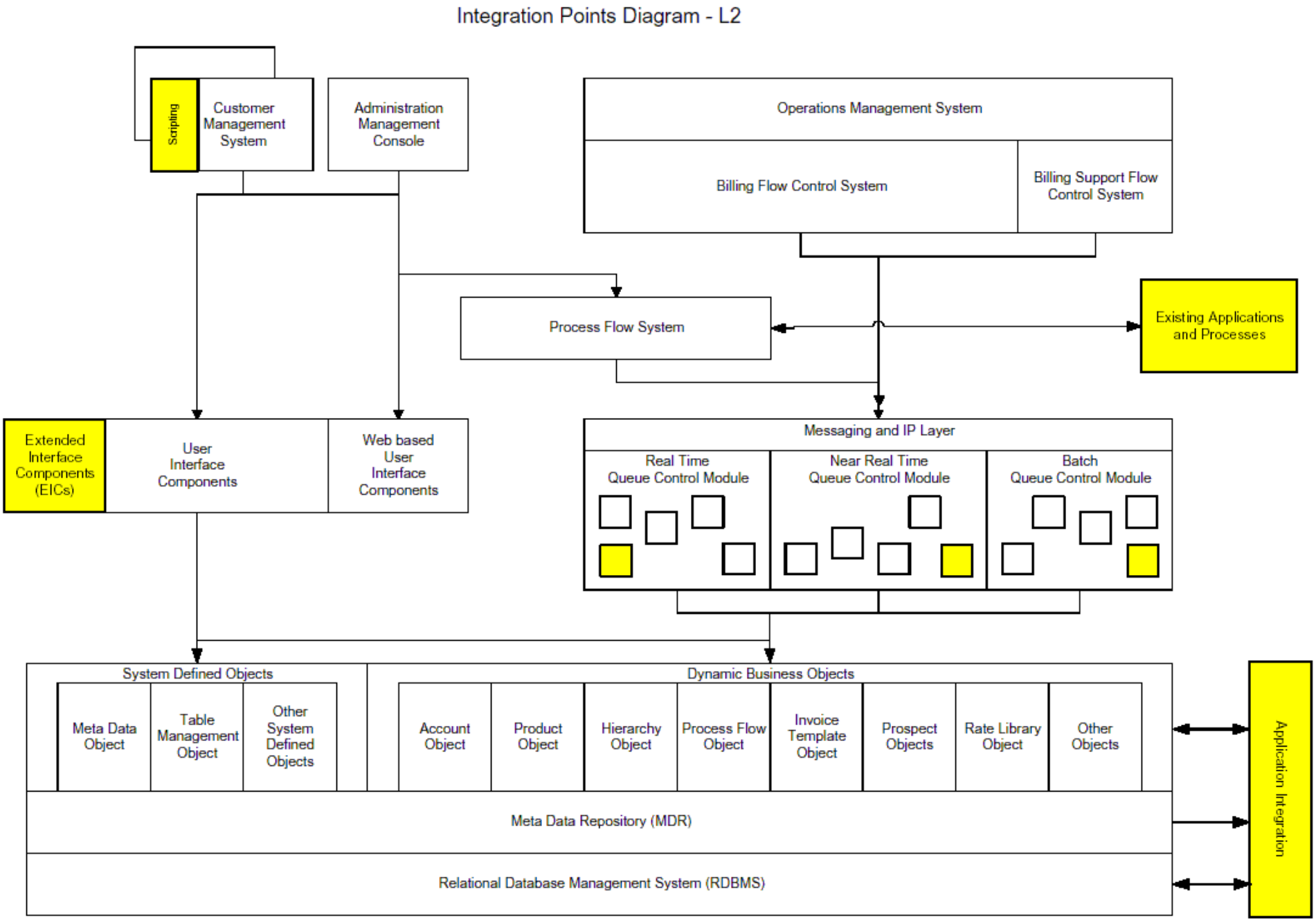

Figure 7 – Priority CS Integration Points

Figure 7 – Priority CS Integration Points

Priority CS provides integration capabilities at various levels.

5.1. Direct access to Priority CS' Relational Data Base
5.2. Access to Priority CS' Meta-data Repository
5.3. Access to Priority CS' COM Objects
5.4. Priority CS' Extended Interface Components
5.5. Processes triggered by Priority CS' Process Flow
5.6. Scripting capabilities via JavaScript and VBScript
5.7. Processes that can be plugged in to the messaging queues that can execute in batch, near real time or real time.

Each of these capabilities will be expanded upon. Additional information is available on request from Bluespring.

5.1 Direct Data Base Access:

While not the preferred or recommended integration method, processes can be developed that will access Priority CS' Relational tables directly. Due to Priority CS' meta-data architecture, extreme care must be used when accessing the tables directly. Values that are set incorrectly may yield unpredictable results when used by the Object, User Interface and User Application Layers.

5.2 Access to the Meta-data Repository:

Priority CS Meta-data can be configured to support a URL that can be passed parameters such as account number. When accessing Priority CS' RDBMS, the Meta-data Repository should be referenced to provide the correct context to the content data. Risk of incorrectly updating Priority CS data is also reduced by accessing the RDBMS with the context provided by the meta-data repository. In addition, the meta-data can be accessed to provide information for reporting systems and other downstream applications.

5.3 Access to Priority CS' COM Objects:

The preferred method for accessing information contained in Priority CS is via the COM Objects. Priority CS' Object Model is provided to all registered users of the product. By providing access to Priority CS' COM Objects, the capabilities of the system can be enhanced while maintaining the integrity of the data in Priority CS. Accessing data in Priority CS via the COM Objects is the recommended approach.

5.4 Extended Interface Components:

As described above, Priority CS' UI can be extended by the use of Extended Interface Components (EICs). EICs are used to provide additional functionality to display and capture attributes and information that are not easily accommodated by Priority CS' existing UI. EICs are SMG custom-developed or user-provided DLL's and accessed by Priority CS' User Interface. These DLL's can perform processing or provide an alternative user interface to support unique business requirements. Priority CS' provides a documented API to which EICs must conform. In addition, EICs have access

to all of Priority CS' COM Objects and can also make use of Priority CS' other User Interface Components (Ocx). One example of Priority CS' use of EICs is the display of rate plan information.

5.5 Processes triggered by the Process Flow:

As described above, Priority CS' Process Flow can be used to extend and enhance the capabilities of Priority CS'. Any number of processes can be triggered by Priority CS' Process Flow System. Priority CS is configurable to provide the external process with the necessary information to be able to complete the task. In addition, values can be returned from external processes and stored in Priority CS, reducing redundant data and the costs associated with inaccurate or out-of-synch information. Examples of processes that can be triggered from Priority CS' Process Flow include:

- Execution of SQL call to an external data base, such as an Oracle PL SQL statement
- Execution of a CGI script, perl script, JavaScript, VBScript, FTP, HTTP, etc.
- Execution of an external program, such as a C, C++ or Visual Basic DLL, Unix shell, etc.
- Execution of a script that emulates a screen on an external system, such as a mainframe or AS-400

5.6 Scripting capabilities via JavaScript and VBScript:

As discussed earlier in the document, one method that Priority CS' provides for the ability to enforce business rules is through the use of scripting technologies such as JavaScript and VBScript. Any capabilities that are available with those technologies can be incorporated into Priority CS. The scripting provides access to Priority CS' Objects and enables edits at multiple levels in the Dynamic Business Objects. In addition, edits are time sensitive. This time sensitivity ensures that edits occur when necessary to enforce the necessary business rules. Some of the options available for edits include field entry, section exit and confirmation of the order.

5.7 Messaging Layer Plug-ins:

Priority CS allows for custom-developed or user supplied applications to be 'plugged in' to Priority CS' processing. The applications can be available for real-time processing, near real-time processing or in the traditional batch-processing environment. These applications must meet Priority CS' documented API and register with Priority CS. Registering with Priority CS informs the application environment and Operations Staff of the availability of the module as well as providing operations with information about the types of events that the application can process. As is common throughout Priority CS, the applications can be supplied in the user preferred development language as long as the resulting application is either a COM or CORBA based object.

Notes

A. A Batch Environment is typically a reference to a mainframe system where a predetermined set of programs is run on a regular basis with little human intervention. This is typical of most Telecommunications Bill Generation processes.

B. Usage data is received in the form of Usage Records

C. A mediation program is used to take data that is received in one format and transform it into a format usable by another system.

D. A Usage Record is a text record that describes a single event, such as a long distance phone call.

References

1. Dhawan, Priya. “Performance Comparison: Exposing Existing Code as a Web Service”. *Microsoft Developer Network*. October 2001.

2. Esposito, Dino. “Design and Develop Seamless Distributed Applications for the Common Language Runtime”. *MSDN Magazine*. October 2002.

3. “Optimizing Message Queuing Performance – White Paper”. Microsoft Corporation. 2000.

4. Treier, Karl. Chief Technology Officer, Bluespring Software, Inc. Personal Interview. February 2003.

5. Zigman, Erik. Former Chief Technology Officer, Bluespring Software, Inc. Personal Interview. November 2002.