A Message Routing System

A Manuscript

Submitted to

the Department of Computer Science

and the Faculty of the

University of Wisconsin-La Crosse

La Crosse, Wisconsin

by

Andrew Backes

in Partial Fulfillment of the

Requirements for the Degree of

Master of Software Engineering

October, 2015

A Message Routing System

By Andrew Backes

We recommend acceptance of this manuscript in partial fulfillment of this candidate's requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

Dr. Kasi Periyasamy Examination Committee Chairperson	Date
Dr. Marti Allen Examination Committee Member	Date
Dr. Josh Hursey Examination Committee Member	Date

Abstract

Backes, Andrew, D., "A Message Routing System", Master of Software Engineering, June 2015, Kasilingam Periyasamy, Ph.D.

This manuscript describes the design and development of a message routing system that analyzes and routes error messages to the appropriate party. The system has been developed as a desktop application that provides the ability to manage messages and create rules that define how particular error messages are handled. It includes a class library that developers will use within applications to handle exceptions, and a console application that consumes the error messages and determines which rule(s) to apply. The manuscript briefly describes future work that will enhance the message routing system.

Acknowledgements

I would like to express my sincerest thanks to my project advisor Dr. Kasi Periyasamy for his valuable guidance. I would also like to thank the project sponsor, Authenticom Inc., who was ultimately the motivation for this project. Finally, I would like to thank my wife and my parents for their patience and encouragement that kept me motivated to finish what I started.

Table of Contents

List of Figures

Glossary

Entity Framework

Object-relational mapper that allows .NET developers to work with relational data using domain-specific objects.

RabbitMQ

Open source message broker software that implements Advanced Message Queuing Protoccol (AMQP).

Relational Database Management System (RDBMS)

An organized collection of data based on a relational model.

Repository Pattern

A software design pattern that separates data access layers and business layers of an application by mapping data from a source to a business entity.

Structured Query Language (SQL)

A programming language designed for managing data in a RDBMS.

1. Introduction

1.1 Background

Authenticom is a company that provides DMS integration and automotive data services for third party service providers, CRM systems, and marketing agencies that serve the automotive industry [1]. Authenticom has consistently been recognized as one of America's fastest growing private companies since 2010, and had a growth of 326% from 2010 to 2013 [2]. During this time period, revenue increased from $3.7 million dollars to $15.6 million dollars, solidifying the company as a multi-million dollar enterprise [2]. As the company continued to grow, so did its technological needs.

The author of this manuscript was employed by Authenticom from January of 2014, until April of 2015. During this time, he was able to work with many different applications that Authenticom developed, and got a real taste of what rapid growth within a company means for software developers, and how one must adapt. With dozens of applications constantly running, being able to handle any failures is crucial, and making sure the appropriate individuals are notified if such an error occurs. A common method of error handling was shared between applications that would essentially email software developers whenever an exception within an application was thrown. This was extremely useful, as developers were notified immediately of errors.

It was quickly realized that this method of error handling was becoming much too verbose. Applications would often retry to process data in the case of an error, which could end with more than 1000 emails being sent out to developers. Email filtering became second nature at this point. But as easy as it was to create rules within Microsoft Outlook to help filter emails, there was still no control over the amount of emails being sent out, which started to add a huge load on mail servers.

1.2 Motivation

In 2014, the author reached out to the Director of Development, Tom Thatcher, regarding the possibility of improving error handling. Rather than filtering thousands of emails (which also puts a load on the email server), why not filter them prior to even being sent out? If it is possible to pool similar error messages together as they are received, it should be easier to notify developers only once about the error that occurred, and they won’t have to rely on email filtering.

As a result, the author proposed a new error handling solution that could be completed as a capstone project for the Master of Software Engineering degree which the author pursued at that time. After a few discussions with other developers, the Message Routing System (MRS) was officially approved for development, and the work began within a few weeks.

2. Requirements and Assumptions

2.1 Requirements

Most of the requirements for the error handling tool were derived from a preexisting error handling software system known as the Message Center. However, the developer of the message routing system (the author of this manuscript) vastly improved upon system architecture, reusability, and scalability. The following requirements were determined for this project:

- MRS must be able to create rules that group up similar message together which are being received within a certain time limit. The grouping up of similar messages is known as a “message pool”. For example, a rule could be defined such that when an error of a certain type occurs, only send out a notification to developers if the error was received 100 times within an hour. If only 50 of the same type of errors occurred within that hour, a single notification should be sent.
- The user must be able to create rules based off the following conditions:
    - Application name
        - The application name is the name of the executable that is using the Message Routing System as its error handling solution.
    - Machine name
        - The machine name is the name of the computer/server that the application is running on.
    - Help text (summary of an application)
        - Help text is any additional information regarding a program that a user can provide to help debug issues with the application.

    - Command line arguments
        - The command line arguments of the application.
    - Exception type thrown
        - The exception type is any exception thrown by an application such as StackOverflowException or DivideByZeroException.
    - Exception message
        - An exception message is a descriptive sentence associated with the exception. For example, in the case of the DivideByZero exception, the exception message might be, “Invalid operation – unable to divide by 0.”
    - Stack trace
        - The stack trace contains information that is helpful while debugging an application, and is a report of the stack at a particular time during a programs execution.
- The error handling library that gets integrated within dozens of applications currently running in production must be decoupled from any database dependencies or specific frameworks such as ADO.NET or Entity Framework.
- MRS should allow application and machine groups to be created, such that one can apply a rule to many applications or machines at the same time (e.g., catching an error thrown by application “A” on servers “B”, “C”, and “D”).
- MRS must be easy to use and rather self-explanatory.

2.2 Assumptions

The development of MRS was designed with the following assumptions:

- The GUI portion of MRS will be developed with Microsoft WinForms, using C#, with SQL Server as the back end.
- An error message can be matched up with many rules, yet a rule can only be in one active pool at a time. An active pool is a pool of messages such that neither the pool maximum time or the pool maximum number of messages has been reached.
- When a new rule is created, the creator of the new rule will be notified if it is overwriting or interfering with any preexisting rules.
    - For example, Bob creates a rule called “Handle Invalid Files” that will send an email to Sarah when an exception of type FileNotFound is encountered. He attempts to save his new rule, but a message pops up telling Bob that a rule called “File not found” already exists that handles the exact same type of exception, but sends an email to Greg. If Bob continues with saving his rule, and puts his rule at a higher priority than the preexisting one, only his rule will be used, and the other rule will never be encountered. Sarah will now receive an email when the FileNotFound exception occurred, and Greg will no longer be receiving the emails regarding this type of exception, because Bob’s rule is interfering with the old rule.

3. Design

3.1 Development Model

The iterative development model was chosen for this project due to its size, and the fact that the system requirements were clearly defined. By using the iterative model, the product was developed and improved step by step, which allowed the developer to track defects at early stages [5]. By repeating the cycle of analyzing, designing, implementing, and testing, the product became more complete after each iteration. Figure 1 demonstrates the iterative development model that was used for this project.

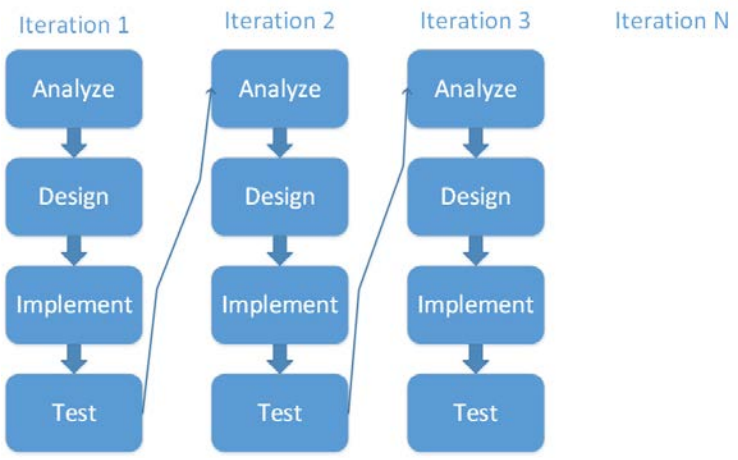


Figure 1: Iterative Development Model

3.2 Database Design

Microsoft SQL Server was used as the relational database management system (RDBMS) for MRS because this server is currently used by Authenticom, and there was no logistical reason to change at this point. The entity relationship (ER) diagram for MRS is shown in Figure 2. There are 11 tables within this database design.

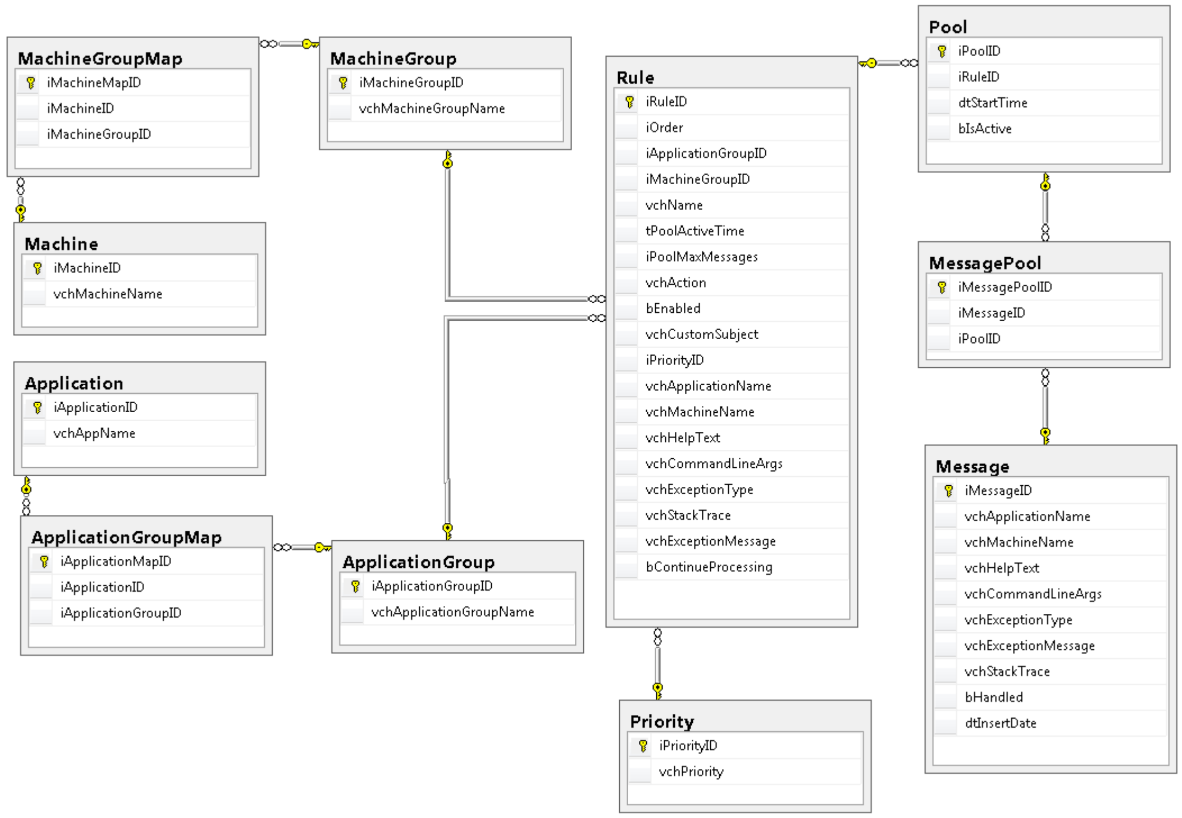


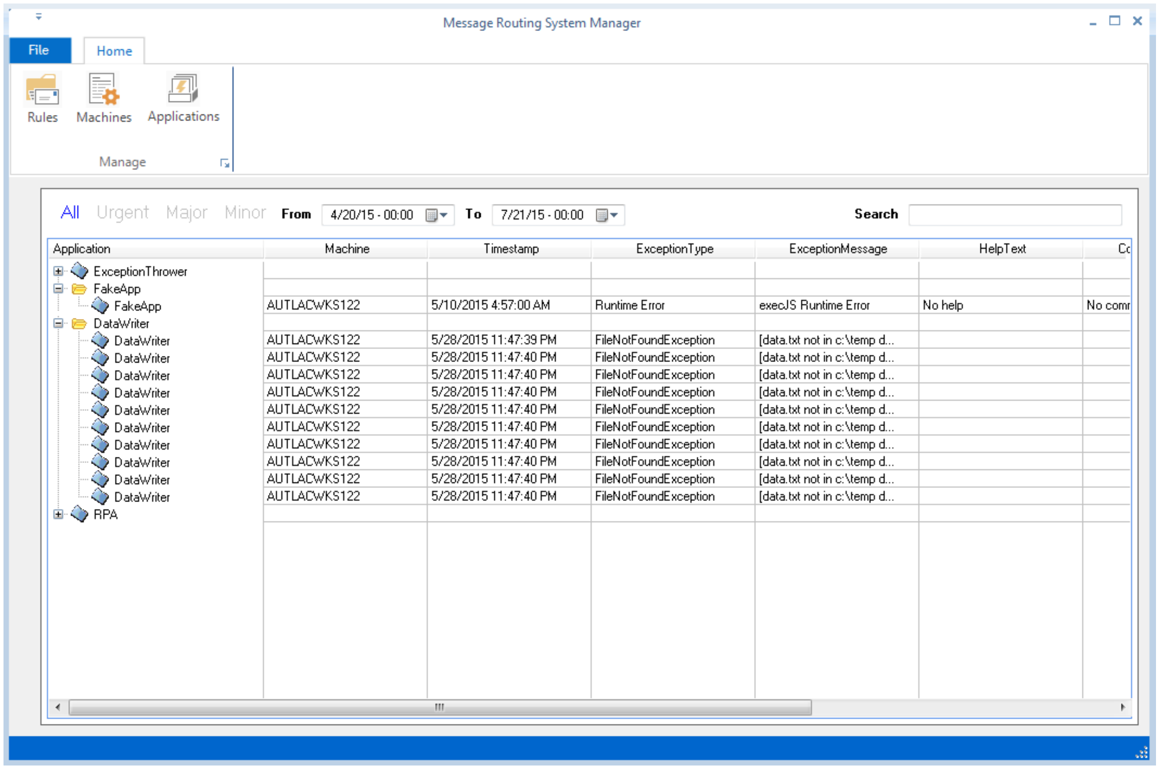
Figure 2: ER Diagram for the Message Routing System

Although there are quite a few tables in the schema, the database design itself is actually rather simple. A user is responsible for creating machines, applications, and rules. When an application is created, it may belong to zero or more application groups. When a machine is created, it may belong to zero or more machine groups. A rule can then utilize applications, machines, application groups, and machine groups. This way, a user can create specific rules to match whole groups, or individual applications and machines.

Messages are created automatically by applications, and not by users. A rule is not directly associated with any message, but is associated with a pool. Using the intermediary table MessagePool, messages can be associated with a pool. By associating messages with a pool, one is able to monitor when a pool has reached its max size (if it has one). This also allows developers to trace back messages to see how they were handled, and when they were stored in a pool (if applicable).

3.3 Interface Design

The interface for MRS is designed to be simple and easy to understand. There are four main screens for the graphical user interface (GUI) of the MRS Manager. The design and colors of the GUI are meant to mimic Microsoft Office 2013 so that MRS looks familiar to users at Authenticom. In Figure 3, the main screen shows the messages, and has a ribbon bar for accessing rules, applications and machines.



From 4/20/15 - 00:00 To 7/21/15 - 00:00

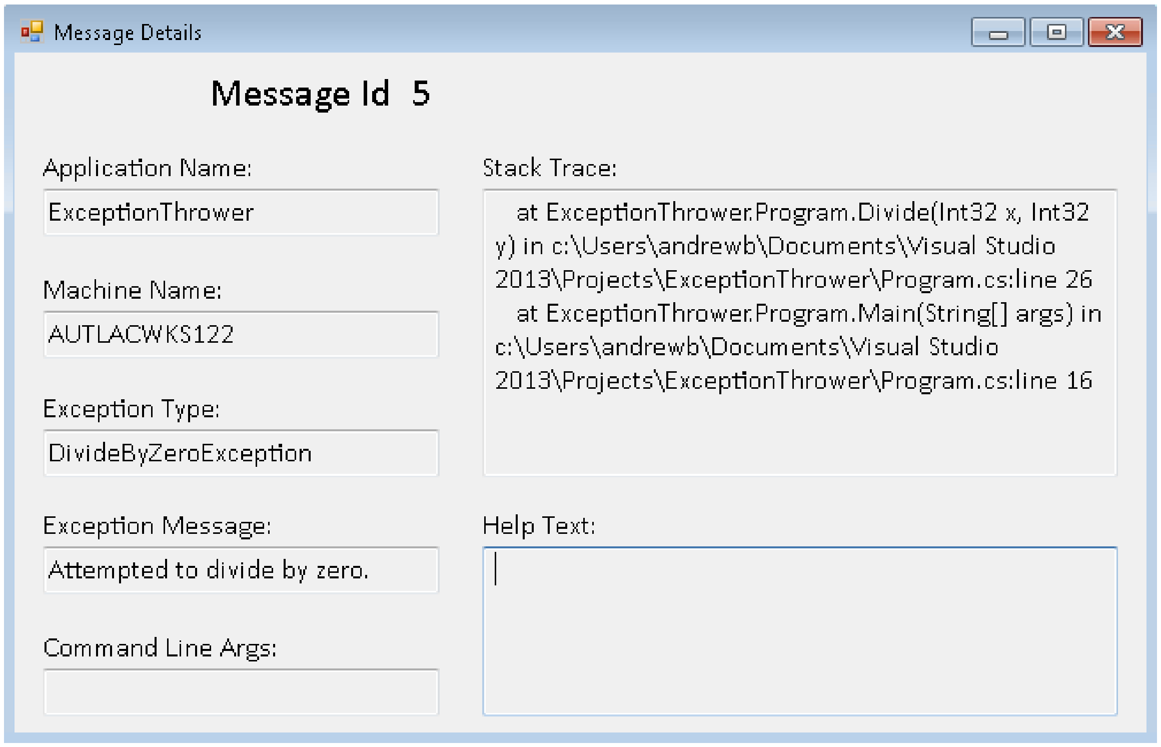
Application	Machine	Timestamp	ExceptionType	ExceptionMessage	HelpText	Cc
ExceptionThrower						
FakeApp						
FakeApp	AUTLACWKS122	5/10/2015 4:57:00 AM	Runtime Error	execJS Runtime Error	No help	No comr
DataWriter						
DataWriter	AUTLACWKS122	5/28/2015 11:47:39 PM	FileNotFoundException	[data.txt not in c:\temp d...		
DataWriter	AUTLACWKS122	5/28/2015 11:47:40 PM	FileNotFoundException	[data.txt not in c:\temp d...		
DataWriter	AUTLACWKS122	5/28/2015 11:47:40 PM	FileNotFoundException	[data.txt not in c:\temp d...		
DataWriter	AUTLACWKS122	5/28/2015 11:47:40 PM	FileNotFoundException	[data.txt not in c:\temp d...		
DataWriter	AUTLACWKS122	5/28/2015 11:47:40 PM	FileNotFoundException	[data.txt not in c:\temp d...		
DataWriter	AUTLACWKS122	5/28/2015 11:47:40 PM	FileNotFoundException	[data.txt not in c:\temp d...		
DataWriter	AUTLACWKS122	5/28/2015 11:47:40 PM	FileNotFoundException	[data.txt not in c:\temp d...		
DataWriter	AUTLACWKS122	5/28/2015 11:47:40 PM	FileNotFoundException	[data.txt not in c:\temp d...		
DataWriter	AUTLACWKS122	5/28/2015 11:47:40 PM	FileNotFoundException	[data.txt not in c:\temp d...		
DataWriter	AUTLACWKS122	5/28/2015 11:47:40 PM	FileNotFoundException	[data.txt not in c:\temp d...		
RPA						

Figure 3 : Main screen of the MRS Manager

The ribbon bar at the top of the application contains three buttons. The first button, “Rules”, is used to access the rules screen to create, delete, and update rules. The “Machines” button is used to access the screen where the user can create and delete machines and machine groups. The final button is the “Applications” button which is used to create and delete applications and application groups. The benefit of using the ribbon bar is that it is easily extendable, and new actions can be added onto it with little effort.

One can also group buttons: For example, the “Rules” button, “Machines” button and the “Application” button belong to the group “Manage”, which are all on the “Home” tab.

By default, the messages for the current day will appear in the main table. The dates and times are adjustable, so one can view older messages as well. The search field allows the user to type in any text to filter results. The columns that the filter gets applied to are Application, Machine, ExceptionType, and ExceptionMessage. All messages get grouped into folders by Application name, and the folder will be collapsed by default. This will make it much easier to view messages, because otherwise having them all appear could be overwhelming in cases of thousands of messages. Expanding the folder reveals the messages inside, and its details can be viewed by double clicking the message. Figure 4 shows an example of message details.



Message Id 5

Application Name: ExceptionThrower

Machine Name: AUTLACWKS122

Exception Type: DivideByZeroException

Exception Message: Attempted to divide by zero.

Command Line Args:

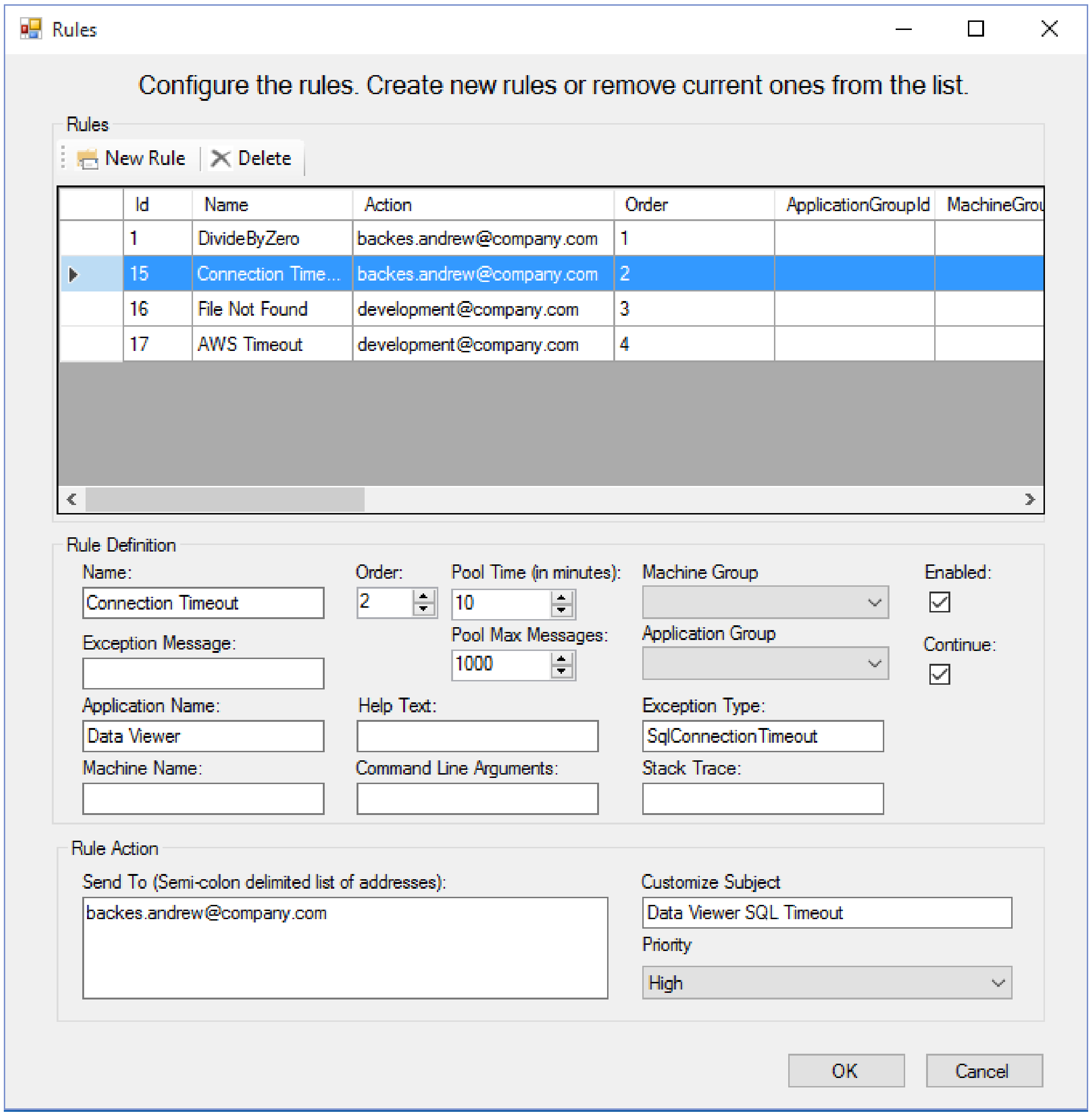
Stack Trace:

```
   at ExceptionThrower.Program.Divide(Int32 x, Int32 y) in c:\Users\andrewb\Documents\Visual Studio 2013\Projects\ExceptionThrower\Program.cs:line 26
   at ExceptionThrower.Program.Main(String[] args) in c:\Users\andrewb\Documents\Visual Studio 2013\Projects\ExceptionThrower\Program.cs:line 16
```

Help Text:

Figure 4: Message Details

The rules screen is where the user will spend most of their time. The user will be able to create new rules, edit existing ones, or delete rules using the rules screen. Figure 5 shows the complete details of the rules screen.



Configure the rules. Create new rules or remove current ones from the list.

Rules

	Id	Name	Action	Order	ApplicationGroupId	MachineGrou
	1	DivideByZero	backes.andrew@company.com	1		
▶	15	Connection Time...	backes.andrew@company.com	2		
	16	File Not Found	development@company.com	3		
	17	AWS Timeout	development@company.com	4		

Rule Definition

Name: Connection Timeout

Order: 2

Pool Time (in minutes): 10

Machine Group:

Enabled: ✓

Exception Message:

Pool Max Messages: 1000

Application Group:

Continue: ✓

Application Name: Data Viewer

Help Text:

Exception Type: SqlConnectionTimeout

Machine Name:

Command Line Arguments:

Stack Trace:

Rule Action

Send To (Semi-colon delimited list of addresses): backes.andrew@company.com

Customize Subject: Data Viewer SQL Timeout

Priority: High

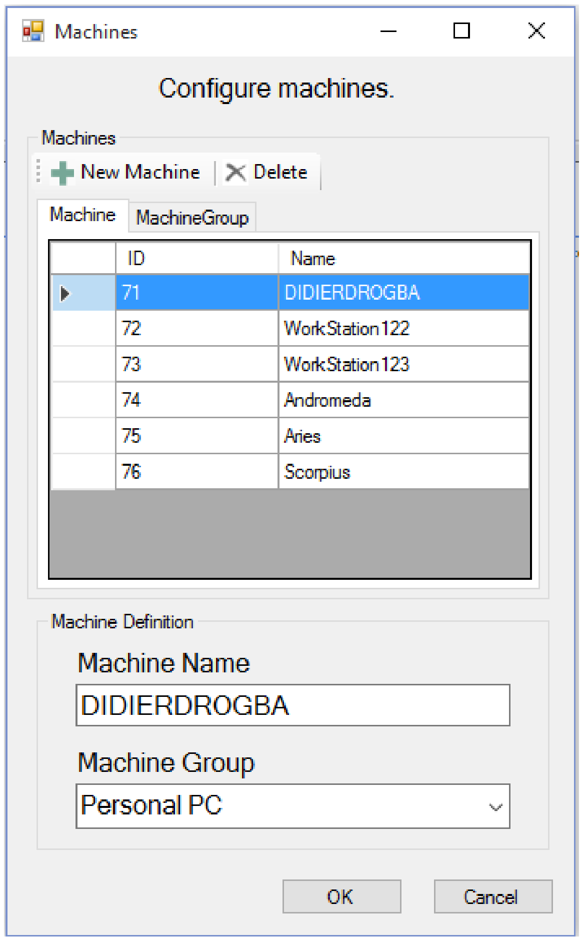
Figure 5: Rules Screen

This screen is divided into three sections – “Rules”, “Rule Definition”, and “Rule Action”. The “Rules” section displays all the created rules, and clicking on any row will populate the “Rule Definition” and “Rule Action” sections. Clicking on “New Rule” will blank out every field on the screen, and automatically set the Order number to be the next number in rule order. If the user then fills out the rule definition and rule action sections, clicking “OK” will create a new rule. If the user wants to edit an existing rule, they must simply select a rule from the table, update whichever fields they want, and hit “OK.” Clicking “Delete” on a selected rule will remove it from the system completely.

The “Rule Definition” section is where the user determines what kind of messages to apply the rule to, and whether or not a message pool should be created. There are multiple fields in this section, which include information about the exception that was thrown. This information may include the exception type, the stack trace, or the message attached to the exception. Other information in the rule definition may include the name of the application that threw the exception, which machine the exception came from, the command line arguments specified from the application that threw the application, and any help text associated with the application. In order to make a rule create a message pool for a specific message, the “Pool Time” or “Pool Max Messages” numeric up downs must be set. The pool time refers to how long the pool should be active before performing the rule action, and the pool max messages refers to how many messages a pool can contain before performing the rule action. Finally, a rule can be enabled or disabled via the “Enabled” checkbox, and a user can also optionally check a “Continue” checkbox to keep processing rules, even if the current rule gets applied to an incoming message.

The “Rule Action” section is primarily responsible for specifying the email address that the rule should be delivered to. The “Send To” section accepts a semi-colon delimited list of email addresses. The “Customize Subject” section is for specifying what the subject of email will be. Finally, the “Priority” dropdown allows a user to mark an emails priority, so that messages appearing in inboxes can be distinguished via a low, medium, or high priority.

The “Machines” screen helps the user create new machines and add them to machine groups. As seen in Figure 6, this screen is small and simple to use.


Machines

Configure machines.

Machines

New Machine | Delete

Machine | MachineGroup

ID	Name
71	DIDIERDROGBA
72	WorkStation122
73	WorkStation123
74	Andromeda
75	Aries
76	Scorpius

Machine Definition

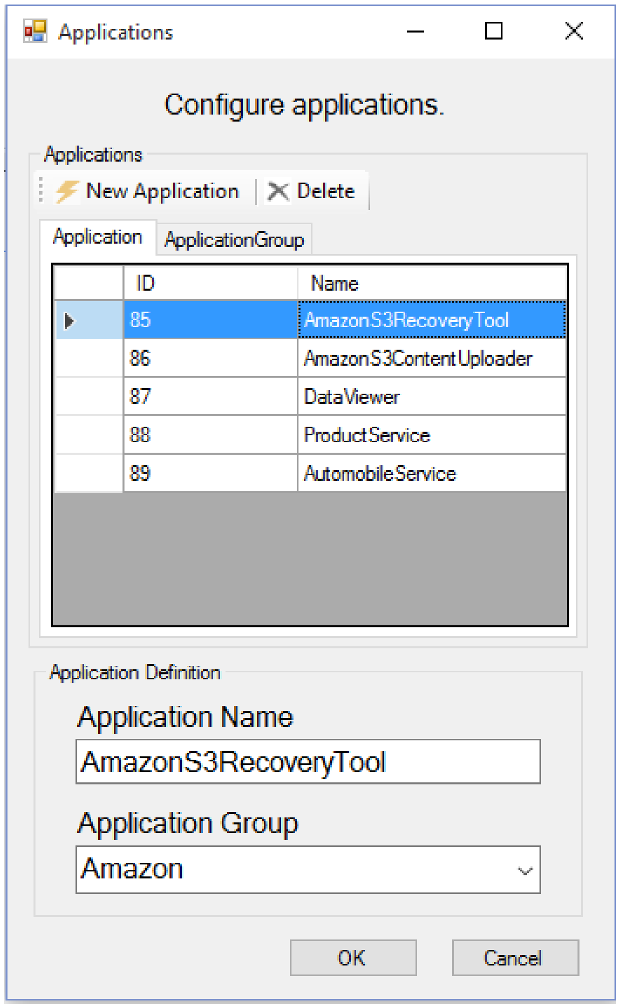
Machine Name: DIDIERDROGBA

Machine Group: Personal PC

Figure 6: Machines Screen

The Machines screen is divided into two sections, the “Machines” section and the “Machine Definition” section. There are two tabs within the “Machines” section that the user can view, the “Machine” tab or the “MachineGroup” tab. The “Machine” tab just shows a table of all the machines already saved in the database, and their respective IDs. The “MachineGroup” tab also shows a table of the machine groups that are saved in the database. The actions a user can perform are either deleting a machine, or adding a new machine. The “Machine Definition” section gets populated with information when the user clicks on a machine in the table. If the user hits “New Machine”, the text fields in this section get blanked out, and the user can create a new machine and select or create a machine group. The machine group field is a combo box so that the user can still type in a new machine group name if they wish, or just select a preexisting group. The “OK” button is used for both creating new machines/machine groups and updating them. If "New Machine" is clicked and the "Machine" and "Machine Group" fields are filled out, hitting "OK" will create a new one. If a row from the machines table is clicked, and then edits are made to the "Machine" and/or "Machine Group" fields, the selected machine will be updated.

The "Applications" screen is very similar to the machines screen and can be seen in Figure 7.



ID	Name
85	AmazonS3RecoveryTool
86	AmazonS3ContentUploader
87	DataViewer
88	ProductService
89	AutomobileService

Application Name: AmazonS3RecoveryTool

Application Group: Amazon

Figure 7: Applications Screen

Just like the "Machines" screen, this screen is divided into two sections as well – "Applications" and "Application Definition". The functionality associated with this screen is exactly the same as the “Machines” screen, except that it is for creating/editing applications and application groups, rather than machines and machine groups. The “OK” button also works the same as the “OK” button on the machines screen. It is used to create new applications/groups, but also update them.

3.4 Message Routing System Manager

The Message Routing System (MRS) Manager is the desktop application that users interact with to create and manage rules. Prior to development, multiple design patterns were considered for this project. These patterns included the Data Access Object Pattern, the Data Mapper Pattern, and the Repository Pattern with a Unit of Work. When determining which pattern would be the most useful and efficient pattern for this project, the developer was primarily concerned with maximizing the amount of code that can be tested with automation and allowing the isolation of the data access layer and the service layer. The Repository Pattern stood out the most, as it is incredibly easy to maintain and read which is important when developing software for any company. It also encompasses the primary functionality offered by the Data Access Object Pattern and the Data Mapper Pattern – that is, the repository is simply a layer of abstraction over the data mapping layer [4]. While the Data Access Object Pattern and Data Mapper Pattern deal with single objects, the Repository Pattern deals with a collection of objects, which expands upon the functionality provided by the other two patterns. By using the Repository pattern, a layer of abstraction is created between the data access layer (Entity Framework / Repositories) and service layer of the application. This not only protects the presentation layer (in this case, Windows Forms) from any changes to the data store, but also facilitates the automated testing process. Figure 8 below demonstrates the layers involved in this pattern.

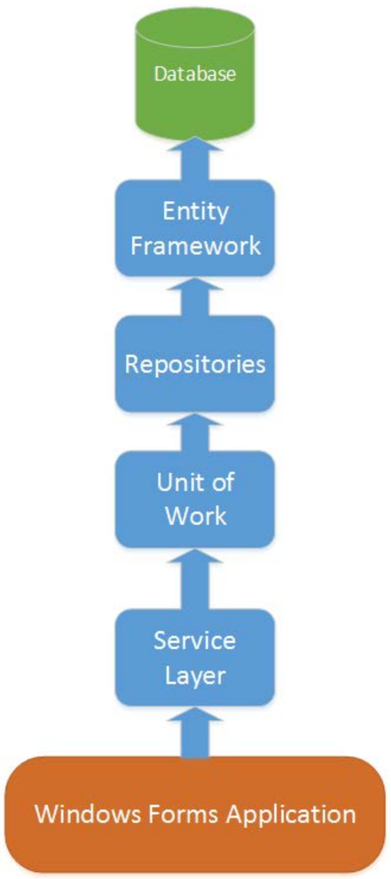


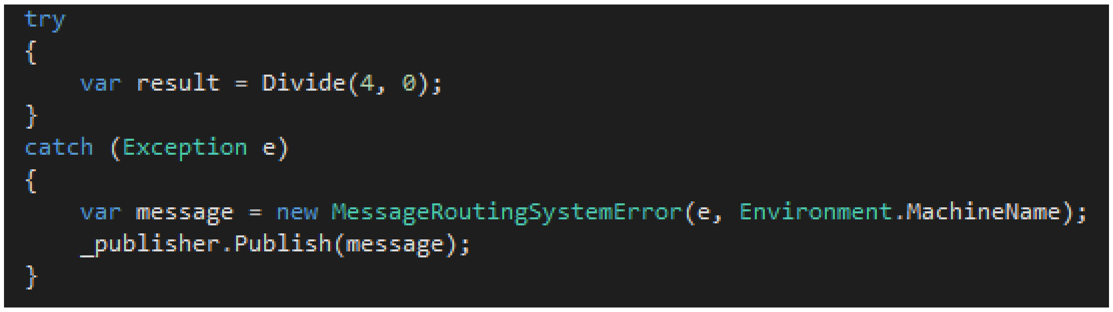
Figure 8: Repository and Unit of Work Design Pattern

The first layer is the Windows Forms Application, or, the MRS Manager. It interacts directly with the service layer, which is where the business logic for the manager resides, and also bridges the gap between the database, and the presentation layer. The service layer uses the unit of work to interact with all the database repositories. These repositories coincide with auto generated entity data models from the Entity Framework. Finally, the Entity Framework is the database object-relational mapper (ORM) that is the direct link to the database.

The MRS Manager was created with Visual Studio 2013 using Microsoft's Windows Forms and was written in C#. This decision was made primarily because many developers at Authenticom are familiar with Windows Forms, and this makes the application easily maintainable. MRS could have been developed as a web-based application but then it would become necessary to support all major browsers, which involves a lot of work during development.

3.5 RabbitMQ Utility

The RabbitMQ Utility tool is the C# class library that applications will reference to perform their error handling, and publish messages to RabbitMQ. RabbitMQ is open source software that implements the Advanced Message Queuing Protocol (AMQP) [7]. It is simply a message broker. So rather than an application inserting its errors directly into a database, it simply publishes them to a message queue via RabbitMQ. Figure 9 below demonstrates how a client application would utilize the error handling found within the RabbitMQ Utility tool.



```
try
{
    var result = Divide(4, 0);
}
catch (Exception e)
{
    var message = new MessageRoutingSystemError(e, Environment.MachineName);
    _publisher.Publish(message);
}
```

Figure 9: Publishing an error message to RabbitMQ

By using RabbitMQ, software applications are able to connect and scale [6]. RabbitMQ only needs to ensure that a message is published and processed, and doesn't need to lock tables in a database or write transaction logs. This greatly improves performance, versus reading and writing to an SQL table potentially thousands of times in a matter of minutes. RabbitMQ queues also reside in memory rather than the disk [6]. An application utilizing RabbitMQ for error handling is completely decoupled from any specific logic as to how the messages are handled. As far as the user is concerned, once the message is published to RabbitMQ, their job of handling the error is done. The message routing and delivery logic is handled by the message broker, so the client isn't responsible for making any routing decisions [7].

3.6 Message Routing System Listener

The Message Routing System Listener is responsible for consuming messages within RabbitMQ. Figure 10 below describes the process of an application publishing a message to RabbitMQ, and a listener that consumes the message from the queue.

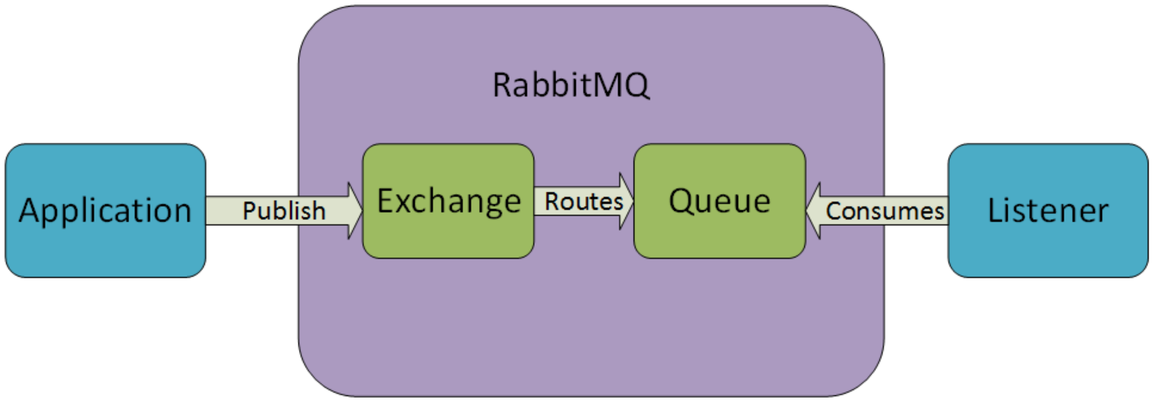


Figure 10: RaabitMQ Workflow

RabbitMQ introduces the concept of producers, queues, and consumers. The producer is the application that publishes the message, a queue is a buffer that stores the messages, and a consumer is the application that receives the messages. Rather than an application publishing a message directly to a queue, it publishes it to what is called an exchange. The exchange is responsible for routing the messages to the appropriate message queues. Once a message ends up in a queue that a listener is subscribed to, it will consume the message. As the message is consumed by the MRS Listener, it will try to find a rule to apply to the message. Once a rule is found, it will either add the message to a message pool, or perform the rule action (e.g, email an error message to one or more recipients).

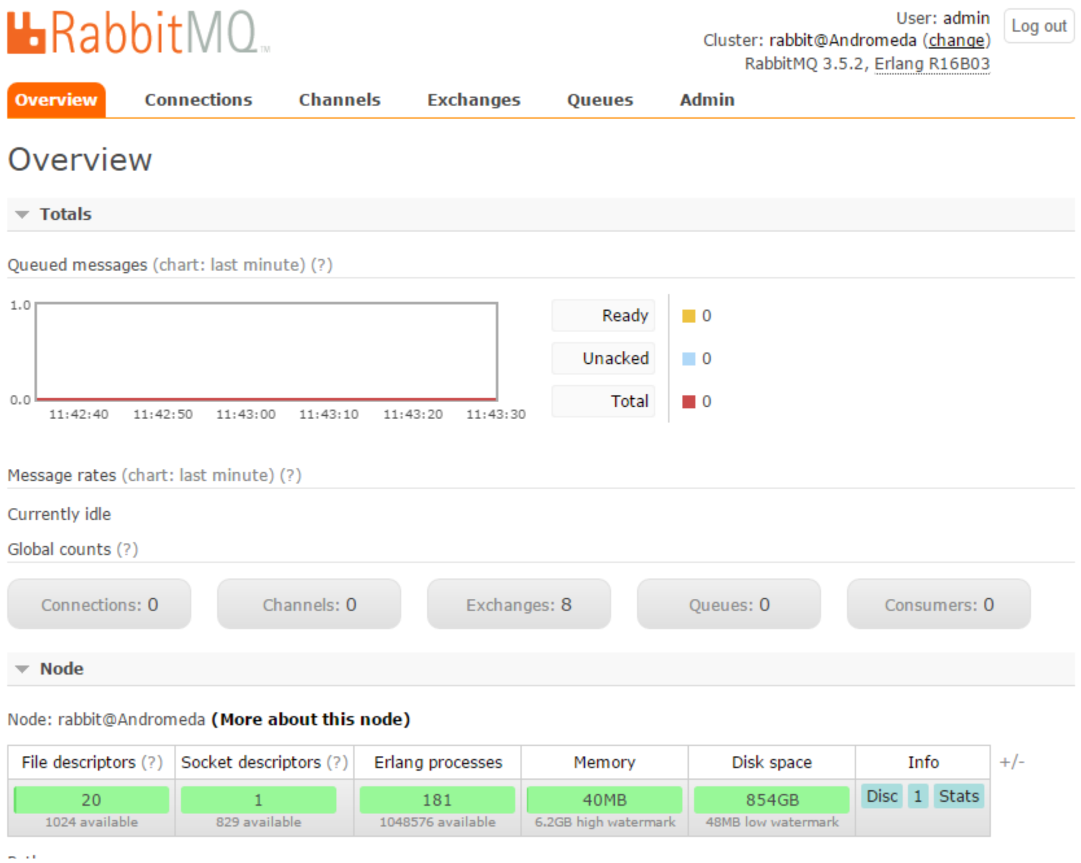
4. Implementation

4.1 Technology

All components of the Message Routing System were developed within Visual Studio 2013 using the programming language C# and .NET 4.5. Most applications at Authenticom are either developed with C# or VB.NET, so C# was a logical choice as a programming language. Entity Framework was chosen as the object relational mapper because of its simplicity, and how easily it works with the repository design pattern. Very few applications at Authenticom use Entity Framework, as most stick with ADO.NET. Entity Framework isn't difficult to understand, but it will require some learning by the developers.

The next decision to make was whether the MRS Manager should be a desktop application or a web application. The decision was made to have the application be developed as a desktop application because of facilitating deployment and maintainability. With a desktop application, any user can run the application executable from their machine, and there is little setup required. But with a web application, it would need to be hosted on an IIS Server, and it would need to support major web browsers (IE, Chrome, Firefox). This application also doesn't need to be accessed externally, so by making it a desktop application, this improves security around who has access to it. The GUI design of the MRS Manager also follows suite with Microsoft Office 2013 applications like Word and Powerpoint. It has the familiar Ribbon bar on top, and gives users a familiar environment to work within.

RabbitMQ was used as the message broker for the MRS Listener because of its simplicity, yet the many useful features associated with it. The server itself is incredibly easy to install, and provides an administration panel in which users can monitor message queues. Figure 11 shows what the administration panel looks like. Users can see messages come into the queues in real time and view which consumers are picking the messages of the queues.



User: admin
Cluster: rabbit@Andromeda (change)
RabbitMQ 3.5.2, Erlang R16B03

Overview

Queued messages (chart: last minute) (?)

Ready	🟨 0
Unacked	🟦 0
Total	🟥 0

Message rates (chart: last minute) (?)

Currently idle

Global counts (?)

Connections: 0 Channels: 0 Exchanges: 8 Queues: 0 Consumers: 0

Node: rabbit@Andromeda **(More about this node)**

File descriptors (?)	Socket descriptors (?)	Erlang processes	Memory	Disk space	Info
20 1024 available	1 829 available	181 1048576 available	40MB 6.2GB high watermark	854GB 48MB low watermark	Disc 1 Stats

Figure 11: RabbitMQ Admin Dashboard

The use of a message broker like RabbitMQ is incredibly beneficial to developers, as one can make any number of listeners to consume the error messages from the queue. Because of this, other applications can be created to handle the error messages as well, while not affecting the MRS Listener in any way.

4.2 Difficulties

Initially, the use of RabbitMQ was not even considered. The original design was that the applications would insert error messages directly into the SQL Server database. This was working out fine until it was realized that by making the applications insert to a database directly, it would be forcing them to support newer versions of .NET and also bind them to specific frameworks like Entity Framework. This isn’t problematic for new applications, but for older ones, forcing them to support multiple versions of .NET frameworks could cause additional headaches for software developers. By using RabbitMQ as a message broker, this allows applications to simply include references to libraries required by RabbitMQ, and not force the use of newer .NET frameworks, Entity Framework, and other ORM systems.

5. Testing

By following the iterative development model, test cases were created as the project was being developed. These test cases revolved around testing methods within MRS as they were created. Unit tests and integration tests were the primary focus for the developer because the developer was using an iterative model and was using shorter cycles of integration. In general, there was typically a minimum of one unit test per method. The number of input parameters and various outcomes of a method dictated the number of unit tests for a single method. Integration tests focused on end-to-end testing, versus testing a single method at a time (e.g., testing the entire workflow of creating a rule and making sure that rule exists in the database). Overall, hundreds of automated units tests and integration tests were created. The execution of automated test suites delivered feedback to the developer within seconds, and any unexpected side effects from changes made are noticed instantly. The downside of relying on automated tests is that it has to be guaranteed to have as many scenarios as possible so that not many cases are missed. Missing tests can be avoided by using Test Driven Development (TDD) approach, but this approach was not used consistently during the development of this tool.

To test the complete functionality, a test application was created that throws exceptions. This application can throw any number of exceptions specified, and will use the RabbitMQ.Utility library to publish the error messages to the RabbitMQ server. The developer would then run the MRS Listener, and validate that it was consuming the messages. A rule was created to catch a specific exception (like DivideByZeroException), and the action that would be sent to the developer’s email address.

6. Deployment

Deploying the Message Routing System relies on a few key components that need to be setup prior to use. First, an SQL Server database needs to be created for storing the rules and messages. This is minimal effort for Authenticom, as they already have multiple servers that can support this database. Next, RabbitMQ needs to be installed. For development purposes, this server was running on the local machine. But for live production purposes, RabbitMQ should be installed on a server that is accessible by any machine that will be running the Message Routing System. Setup is as easy as running an executable file, but hardware requirements must be met. Once these two key components are up and running, the Message Routing System can now be configured to use the appropriate server settings via the App.config files in the project solution.

7. Future Work

Although the Message Routing System was designed to be scalable and robust, it could be even more reusable by bundling some of the classes into NuGet packages. NuGet is a package management system for .NET. By bundling software into NuGet packages, software developers can easily add third party libraries to their projects. The ErrorMessasge class could be made into a NuGet package so that all consuming applications don't need a project reference to it, just add the NuGet package. The same thing could be done with the RabbitMQ.Utility project. By turning the most reusable portions into Nuget packages, applications can't be broken by changes being made to its dependencies. Only when the client explicitly wants to upgrade the package will they receive the latest code. Applications can even target older packages, in case a newer package has something the application cannot support. The reason this wasn't implemented is because Authenticom would most likely want to have a private Nuget repository, versus making these packages available to the public. The idea of breaking apart some of the code into separate deployable packages follows the idea of microservices architecture (http://microservices.io/patterns/microservices.html). This makes an application more maintainable, and helps remove any tightly coupled portions of the application.

8. Bibliography

[1] (2015, May 26). *Authenticom* [Online]. Available: http://www.authenticom.com/faq.html

[2] (2015, May 26). *Authenticom: Press Release* [Online]. Available: http://www.authenticom.com/pdf-downloads/Inc5000_2014_PressRelease.pdf

[3] Cheng Zhang; Budgen, D., "What Do We Know about the Effectiveness of Software Design Patterns?" *Software Engineering, IEEE Transactions*, Vol.38, No.5, pp.1213,1231, Sept.-Oct. 2012.

[4] (2015, June 3). *Microsoft Developer Network: Repository Pattern* [Online]. Available: https://msdn.microsoft.com/en-us/library/ff649690.aspx

[5] Osorio, J.A.; Chaudron, M.R.V.; Heijstek, W., "Moving from Waterfall to Iterative Development: An Empirical Evaluation of Advantages, Disadvantages and Risks of RUP," *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference*, Vol., No., pp.453,460, Aug. 30 2011-Sept. 2 2011

[6] (2015, September 20). *What can RabbitMQ do for you?* [Online]. Available: https://www.rabbitmq.com/features.html

[7] Vinoski, S., "Advanced Message Queuing Protocol," *Internet Computing, IEEE*, Vol.10, No.6, pp.87,89, Nov.-Dec. 2006.

Appendix

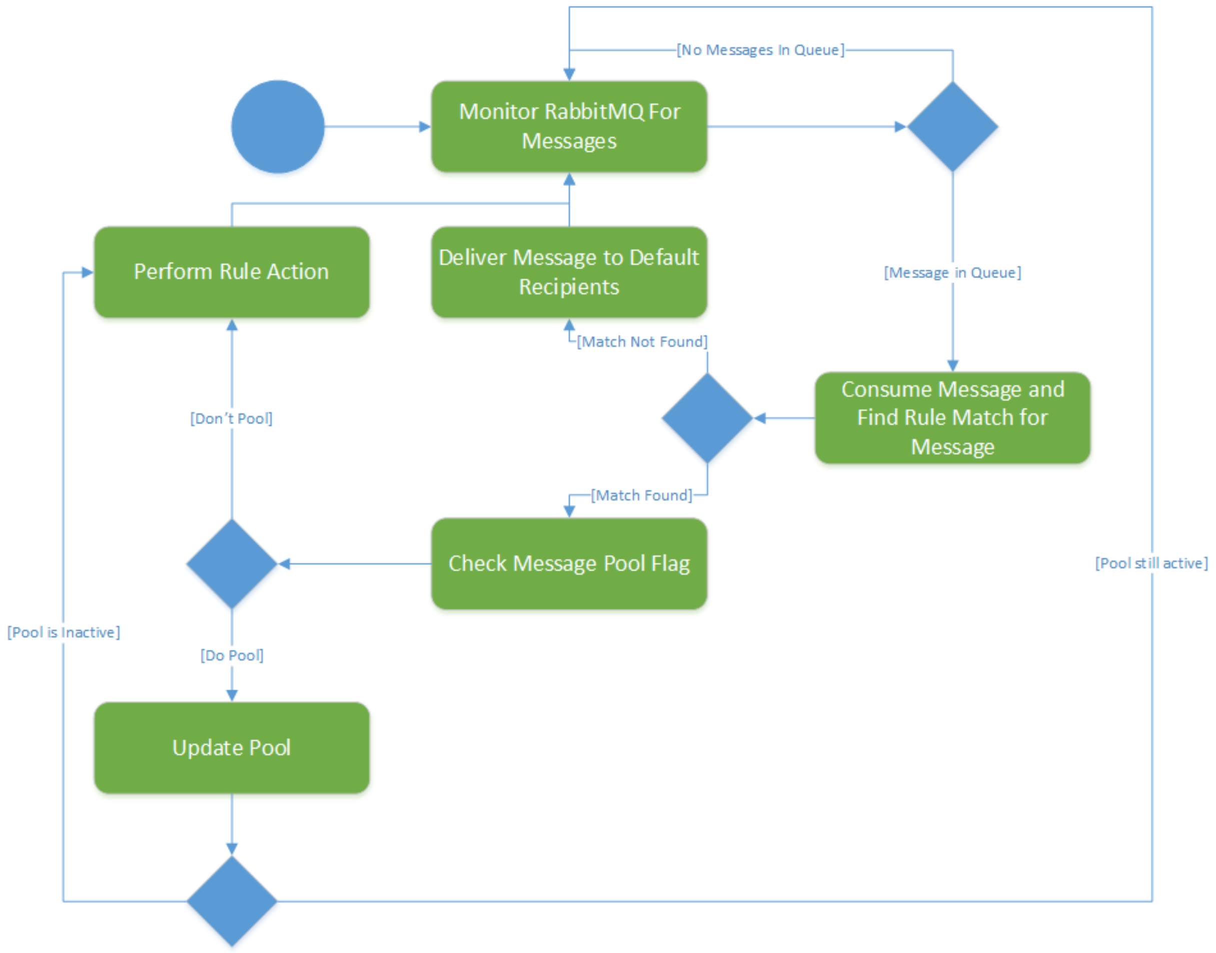


Figure 12: MRS Listener Activity Diagram

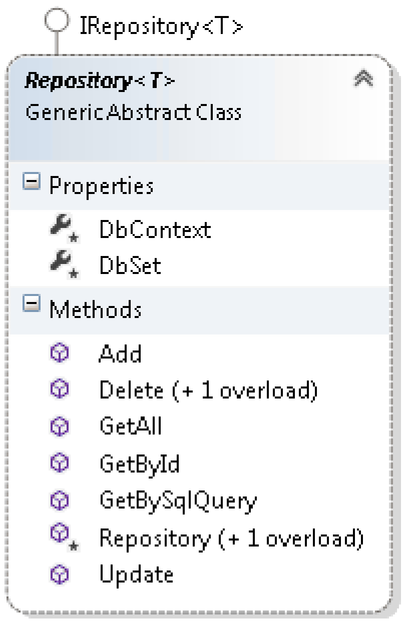


Figure 13: Generic Repository Class Diagram

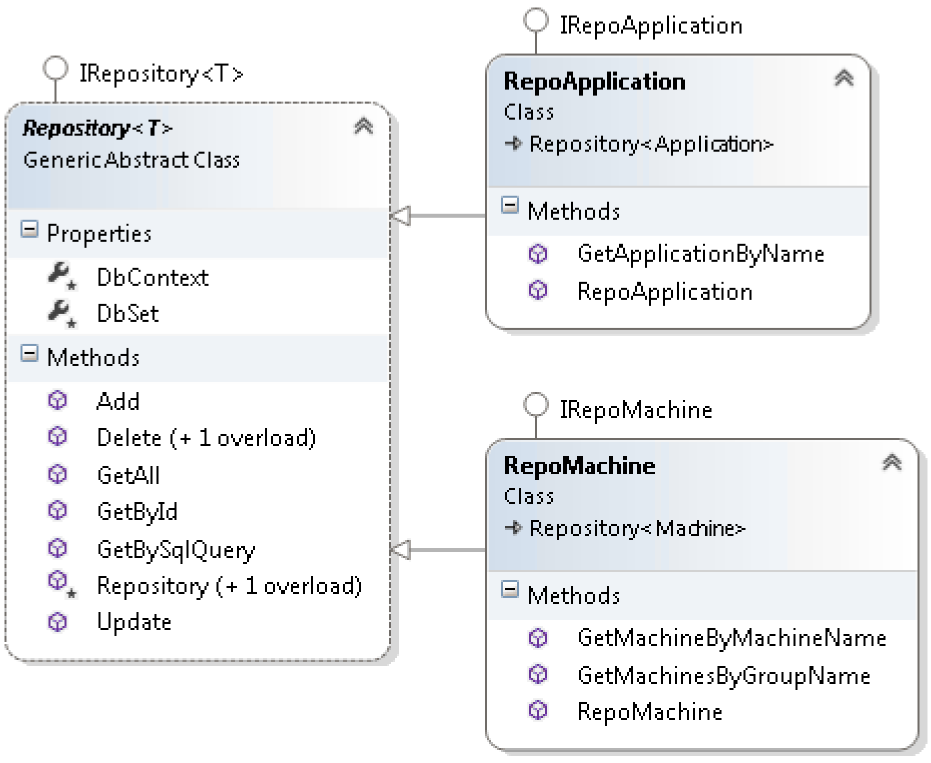


Figure 14: Example Usage of Generic Repository

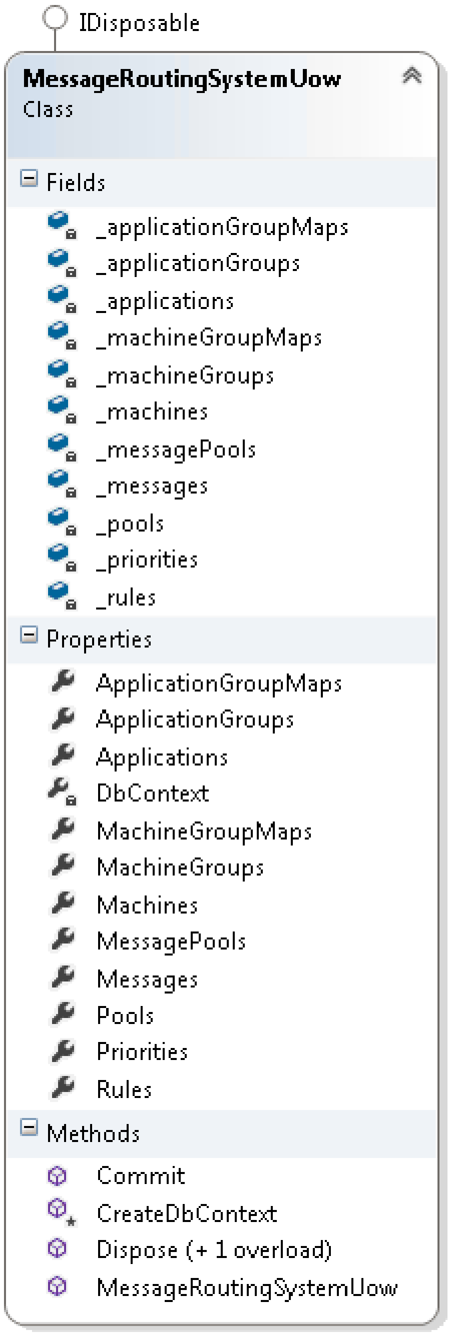


Figure 15: MessageRoutingSystemUow Class Diagram (Unit of Work)