January 2015

The Gateway Interface

Krystel Sophia Walker
Worcester Polytechnic Institute

Yuchen Guo
Worcester Polytechnic Institute

Zhuohao Ling
Worcester Polytechnic Institute

THE GATEWAY INTERFACE

Sponsored by:

Yuchen Guo
Zhuohao Ling
Krystel Walker

THE GATEWAY INTERFACE

A Major Qualifying Project Report
submitted to the faculty of Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
Submitted on January 29, 2015

Submitted By:

Yuchen Guo
Zhouhao Ling
Krystel Walker

Submited To:

On-site Liaison:

Terrance Snyder

Project Advisors:

Jon Abraham
Micha Hofri
Xinming Huang

Abstract

The gateway interface acts as a communication framework, which connects any potential internal client system to the Barclays' reporting database. This gateway interface processes a client's pre-confirmation request and triggers a message to the reporting platform. A pre-confirmation report, of futures contracts, documents all the daily transactions for each client before it is archived in the books and records system. Since Barclays has clients in Tokyo, who frown on amendments in the books and records system, this gateway interface provides them the ability to confirm their stance on the offsetting trade transactions. In addition, it automates the process and significantly reduces the fifteen minutes wait-time for one report to two minutes for all pre-confirmation reports.

Executive Summary

Barclays offers pre-confirmation report as a product for ten of its Tokyo clients, because these clients frown on amendments in the books and records system. A pre-confirmation report, of futures contracts, documents the daily transactions for each client before it is archived in the books and records system. The purpose of a pre-confirmation report is to show the clients what their end of day data will look like. This allows clients to verify their end of day transactions before the confirmation is sent, which eliminates the Firm having to make possible amendments. Barclays generates approximately ten pre-confirmation reports daily, this is done via a manual process. Similar to many other manual processes it is error prone and takes approximately 15 minutes per report.

We were invited to Barclays to automate the process of generating the pre-confirmation report. Two of the major requirements that the firm needed the students to achieve were to:

Barclay's Major Requirements:

- Reduce the wait time of approximately fifteen minutes (per report) between the time the client request the pre-confirmation report and the time the report is delivered to two minutes (for all reports).
- Create an application that would automate the process that allows the Futures Contracts transaction data to be inserted into the reporting database.

Based on the objectives outlined by the firm, the goal of the project is to create an application that serves as a gateway interface between any potential upstream system and the reporting database. Upstream systems are often referred to as client systems within the Firm, for example a trade blotter system. These client systems are the Firm's entities. The reporting database is

an oracle database within the Firm, which is used to store data coming from upstream systems. Based on this goal, we outlined the following objectives:

1. Understand pre-confirmation reports
2. Understand different technologies associated with the development of the gateway interface
3. Design and develop the gateway interface
4. Develop a system that is isolated and easy to maintain

To ensure that we were able to accomplish these objectives, we enforced an agile form of project management, Scrum. Scrum is an iterative and incremental agile software development framework for managing product development. It defines a work strategy where a development team works as a unit to reach a common goal. Applying this form of project management allow us to work efficiently in designing a gateway interface that automates the process of generating pre-confirmation reports. The gateway interface is designed to be flexible and scalable. It can accept and process data from multiple sources that are intended to send data to the reporting database. The Figure A below illustrates the manual process and the automated process.

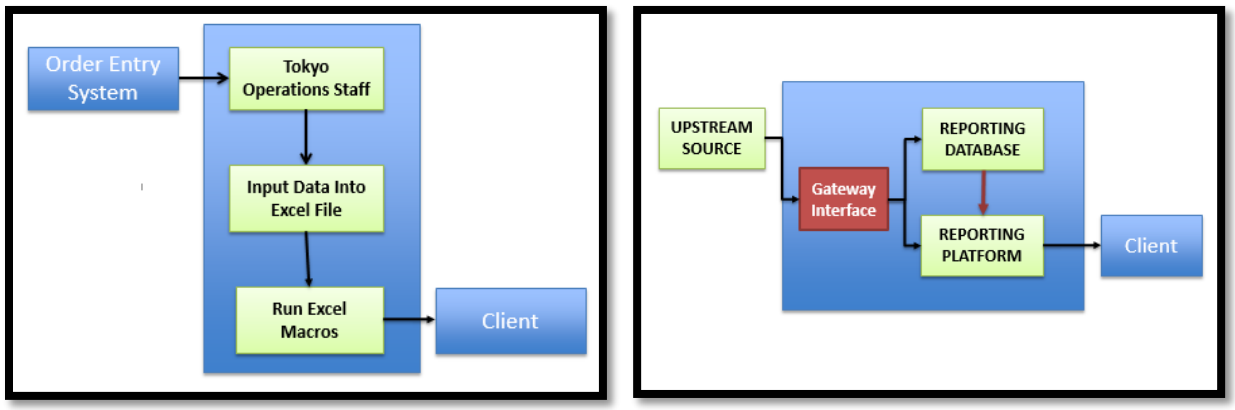

Figure A: Manual Process (left) & Automated Process (right) of Generating a Pre-confirmation Report

The image on the left illustrates the manual process, while the image on the left illustrates the automated process. The manual process is as follows: an employee, from Tokyo operations, will query transaction data from the Order

Entry System and manually import the data into a Microsoft Excel file. Within the excel file he will run Excel Macros, which will generate a pre-confirmation report. The automated process starts with the transfer of the transaction data, from the upstream system. It is transmitted into the gateway interface via a JMS queue. The gateway interface is responsible for, but not limited to: identifying the data type, injecting data into the reporting database, and building a trigger message. The data leaves via a database inserter and a different JMS queue. The database inserter inputs the transaction data into the reporting database and sends a trigger message to the reporting platform. This automated process of generating a pre-confirmation report also makes a connection between the reporting database and reporting platform. Once the reporting platform receives the trigger message, it will extract the necessary transaction data from the reporting database and create a pre-confirmation report.

This new process that we designed and implemented reaps many benefits to Barclays.

Benefits of the Gateway Interface:

- Automates the process of generating pre-confirmation report
- Reduces the time of generating the pre-confirmation report from about fifteen minutes for each report down to two minutes for all reports
- Give clients the ability to approve their stance on the offsetting trading transactions before the data is submitted to the books and records system
- Allows multiple upstream systems, rather than one, to send data to downstream reporting database.
- Strengthens the relationship between Barclays and their clients in Tokyo, who frown on revisions in books and records system

As we reflect on the project, we found that the project was conceptually fascinating and yet intellectually challenging. We are from different academic background, this allowed us to approach the problems from our own perspective and contribute through our own special way. During this intense

period, we realized that we were able to practice the ideas and theories we learned throughout our academic career at WPI and came up with creative solutions to solve real world problems at Barclays. We also feel very fortunate to have the opportunity to make an extra impact on the firm by providing solution to connecting to the reporting database.

Acknowledgement

We would like to thank our advisors, Professor Jon Abraham, Professor Micha Hofri, and Professor Xinming Huang, for their continued support and guidance throughout this project. Furthermore, we would like to thank onsite liaisons, Terrance Snyder, and Gregory Friel, for their support, willingness to contribute to this project, and for giving us the opportunity to work on this project.

Our team was comprised of three students with different academic backgrounds from Worcester Polytechnic Institute (WPI), and they were pursuing Bachelor degrees in Actuarial Mathematics, Computer Science, Electrical and Computer engineering. Working as cross-functional team provided a unique and educational experience.

Table of Contents

Table of Figures

Chapter 1: Introduction

Barclays is a major global financial services provider founded in Lombard Street, London during the 1690's by two goldsmiths, John Fream, and Thomas Gould. Barclays is currently located in 50 different countries around the world, spanning from the Americas, to Asia, and employs over 140,000 people worldwide.

This project is sponsored by the Barclays IT - Futures Clearing department within the Investment Bank in New York. The department is currently working on the decommission Futures trade processing project. The Middle-Office system, that is responsible for the trade processing, performs a variety of tasks. A few are: position maintenance, position reporting, trade closeouts, and client specific pre-confirmation reports. This project focuses on client specific pre-confirmation reports; a client in this case can be thought of as a trader.

A pre-confirmation report, of futures contracts, documents all the daily transactions for each client before it is archived in the books and records system. This is sent to the client pending their confirmation of their daily trading activities. These daily trading activities refers to futures contracts. Futures contracts can be thought of as a financial agreement, which requires a buyer or seller to purchase or sell at a prearranged date or price. In a futures contract a client opens or closes a position by buying or selling futures. A client can have a long position or a short position. A long position, can be thought of as, buying futures with the intention to sell it at a higher price on an upcoming date. A short position is selling futures that a client does not yet own, with the intention to buy it at a lower price on a upcoming date. A client can either open a new position or close an existing position depending on the risk they are willing to take. Closing an existing position is similar to canceling a risk; the client's potential of a loss occurring has been repealed. While opening a new position would be creating a new risk, the client has now made himself vulnerable to a potential loss.

The pre-confirmation report gives client the ability to verify their end of day transactions before it is submitted and archived in the books and record system. Since the Firm has “Platinum” clients who frown on amendments to the books and records system, the pre-conformation report is needed to eliminate the possibility of the Firm having to make changes. “Platinum” clients refer to the Firm’s largest clients, clients who generate a great deal of business. At this time, pre-confirmation reports are only offer to the “Platinum” clients in Tokyo. Currently there is a manual process in place that generates a pre-confirmation report upon request; as such it is error prone. We focused on automating the pre-confirmation process through a new workflow, in order to generate reports in real time.

The goal of the project is to create an application that serves as a gateway interface between any potential upstream system and the reporting database. Upstream systems are often referred to as client systems within the Firm, for example a trade blotter system. These client systems are the Firm’s entities. The reporting database is an oracle database within the Firm, which is used to store data coming from upstream systems. This gateway interface stores, processes, and creates a trigger message to the reporting platform for the pre-confirmation report. The gateway interface will act as a communication framework between the client systems and the reporting database. The objectives that are outlined in *Chapter Three: Methodology* of this paper allow the team to accomplish this goal.

Before the project commenced, the team developed an understating of a pre-confirmation report and its core factors. Additionally, the team was required to understand different technologies that are essential to the design and the development of the interface. The team also ensured that the interface they developed is easy to maintain and provided documentation as well as recommended enhancements.

Chapter 2: Background

This Chapter gives more details on the Firm, an overview of the project and takes a closer look at the concepts and technology that we used to complete the project.

2.1 Barclays

The British bank offers a variety of products, such as: retail banking, credit cards, corporate and investment banking and wealth management.[1] Barclays, the Firm, continues to be a pioneer for banks across the world. In 1819, when others thought the train was a futuristic idea, Barclays funded the world's first industrial steam railway. Barclays introduced the world's first ATM in 1967. In addition, Barclays was the first to offer online banking services and the first bank in the UK to offer credit cards. During the 2008 financial crisis the firm acquired the bulk Lehman brothers investment banking operations. More recently the Firm introduced Barclays Biometric Reader in the UK. This allows customers to register a finger with a backup PIN. Giving the customers access to their accounts by simply scanning their fingers, more specifically the biometric reader scans the vein patterns for login credentials. This new technology will be available to corporate clients in 2015. Barclays continues to be a pioneer to offer banking services with innovative technology.

2.2 Project Overview

The Order Entry System (OES) is a legacy Japan middle office system used for futures trade processing. The Firm's Japanese business flows through the OES. The initial implementation of the OES was for a Futures and Options/Cash platform. Since the decommission process of the OES has started, its trading functions has been move to other strategic platforms and it

[1] https://www.banking.barclaysus.com/about-us.html

is most commonly used for its middle office function. Below are the divisions for the *OES Decommission Project.*

***OES Decommission* divisions:**

- IT Risk
- Operational Risk
- Client Capacity
- Cost Avoidance
- Client Specific Pre-Confirmation Reports

This project focuses on Client Specific Pre-Confirmation Reports, a small section of the *'OES Decommission Project'*. As alluded to earlier in this report, a pre-confirmation report documents the daily transactions for each client before the data is submitted and archived into the books and records system. The Firm offers pre-confirmation report as a product for its Tokyo clients, because these clients frown on amendments in the books and records system. Currently pre-confirmation reports are offered to ten Tokyo clients. The purpose of a pre-confirmation report is to show the clients what their end of day data will look like. This allows clients to verify their end of day transactions before the confirmation is sent, which eliminates the Firm having to make possible amendments. The report gives the client the opportunity to confirm their stance on the offsetting transactions. Ultimately giving them the chance to agree with the opening of a new trade position or the closing of an existing trade position; a position can either be long or short. Currently the pre-confirmation is generated manually, and like many other manual processes, it is error prone. Below is a flow chart that outlines the current manual process.

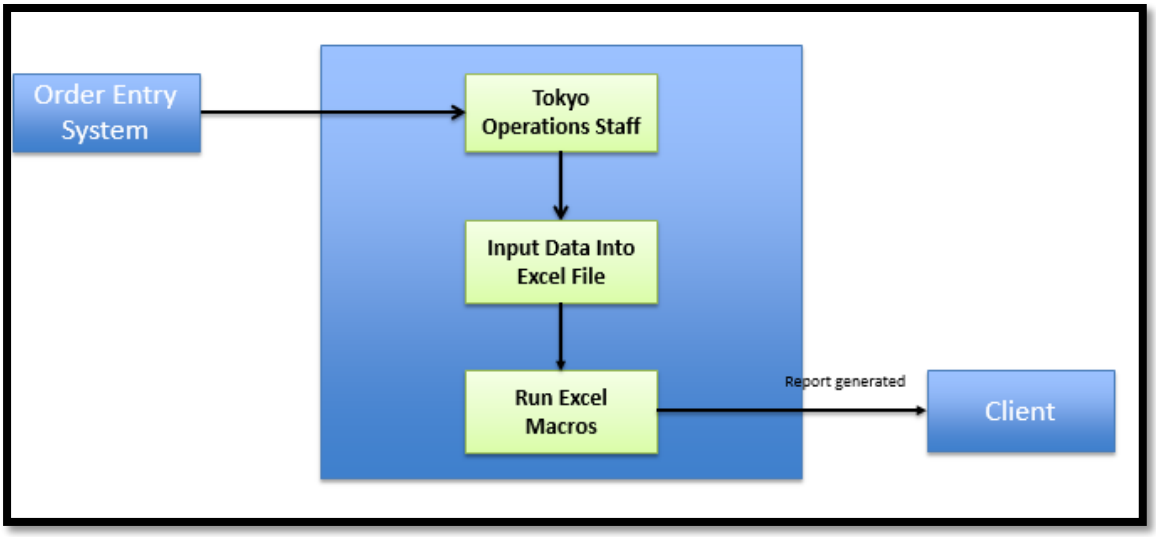

Figure 1: Current Process of Generating a Pre-Confirmation Report

The process is as follows: an employee, from Tokyo operations, will query transaction data from the OES and manually import the data into a Microsoft Excel file. Within the excel file he will run Excel Macros, which will generate a pre-confirmation report. It takes approximately fifteen minutes to generate each pre-confirmation report. As you can see in figure 1, the OES is an essential member of the current process. Since the Firm intends to no longer use the OES, our project designed a gateway interface that creates an alternate way to generate pre-confirmation reports. This will be discussed in more detail in *Chapter Four: Design and Development of the Gateway Interface*.

2.3 Technology

This section describes the different technologies used in this project.

2.3.1 Java

Java is a programming language created by Sun Microsystem.[2] It is a language designed for objected-oriented programming concepts. Implementing this concept in Java, developers can represent the objects with data fields (attributes) and methods (desired procedures). Java is also designed to allow programs to run on any platform without having to be rewritten or recompiled.

[2] https://www.java.com/en/download/faq/whatis_java.xml

This feature is achieved with Java Virtual Machine (JVM), which interprets compiled Java bytecode and executes it as actions of the operating system.[3]

2.3.2 Java Message Service

Java Message Service (JMS) is an Application Programming Interface (API) that enables applications to create, send, receive, and process messages.[4] For any project that requires message transmission from one system to another, developers can set up a messaging queue, which connects the upstream and downstream system. Within the upstream system, a message sender is responsible for constructing the payload, or message, by adding headers to the data and sending it downstream. On the other end, the message receiver listens on the arrival of the desired payload. Once the payload is received, the receiver can process the data and perform a series of commands to complete the request.[5]

2.3.3 Enterprise Integration Patterns and Messaging

Enterprise Integration Patterns (EIP), proposed by Gregor Hohpe and Bobby Woolf, is a collection of design patterns that aims to provide consistent and reliable solutions for connecting large-scale systems.[6] For systems to communicate, data need to be exchanged in a consistent form. Messaging is one of the most commonly used forms of EIP, which enables high-speed, asynchronous communication with reliable delivery. A channel, or queue, is the gateway that connects two systems to facilitate message delivery. A sender is the one that produces and delivers the message to a channel, while a receiver is the one that accepts and processes the message. On the other hand, a message normally consists of a metadata and a message body. The metadata contains the information about the message, which is usually irrelevant to the

[3] http://searchsoa.techtarget.com/definition/Java-virtual-machine
[4] http://docs.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/overview.html#1027335
[5] http://docs.oracle.com/javaee/1.3/jms/tutorial/1_3_1-fcs/doc/client.html#1056518
[6] http://www.enterpriseintegrationpatterns.com/

application. The body contains the data to be delivered, and it is processed by the receiver.

2.3.4 Spring Framework

Spring framework is an open source Java based application framework. It contains a set of powerful extensions that help speed up the development process. For example, the Spring Web Service extension within the framework provides libraries that help developer quickly launch a website by writing minimal amount of code.[7] Spring framework, which supports Enterprise Integration Patterns (EIP), can be used to serve as the foundation of a message-based system.[8, 9] With the framework, the development process will be immensely simplified, due to that fact that many components of the system are already created and available for use. Once new implementation is added, the project can be configured and built seamlessly.

2.3.5 Apache Maven

Apache Maven, often referred to as Maven, is a build framework that ensures that the application is built successfully. Maven is designed specifically for Java-based projects.[10] Build framework refers to automating the various tasks of the developers within the project. Using Maven to aid a project not only allows developers to build the application uniformly, but also allows them to manage the project dependencies on external components and modules.

2.3.6 Testing

Developers often use Test Driven Development (TDD) as a systematic way to ensure code accuracy. It combines test-first development where one writes a

[7] http://projects.spring.io/spring-ws/
[8] http://projects.spring.io/spring-framework/
[9] http://www.enterpriseintegrationpatterns.com/
[10] https://maven.apache.org/

test code before the production code. A unit test is one or a collection of test codes that executes a specific functionality in the application. JUnit is a framework for developing Java unit tests, which help implement the concept of TDD.[11,12] It provides a base class called "Test Case", which create a series of tests for the production code. In addition, the "Assertion" library is used for evaluating the results of individual test cases.[13]

2.3.7 Google Protocol Buffers

Google Protocol Buffers is a widely used, robust, and efficient data interchange format provided and maintained by Google. At present, the main Protocol Buffers implementation supports three target programming languages: C++, Java and Python. To build a protocol buffer, developers define structured data using a *.proto* specification file. The file normally contains one message type that has one or more uniquely numbered fields. In addition, fields can be specified as optional, required, or repeated based on the need of project. This file is then consumed by the Protocol Buffers compiler (protoc), which will generate supporting methods so that it can write and read objects to and from a variety of streams. Protocol Buffers plays an important role for systems that need to communicate between each other through messages.[14] Since Protocol Buffers is extensible, the defined message type can be updated at a later time without breaking programs that used the earlier message. The main benefit of using Protocol Buffers is that data can be encoded using Protocol Buffer API and sent over a specific network.[15]

[11] http://users.csc.calpoly.edu/~djanzen/research/TDD08/cdesai/IntroducingJUnit/IntroducingJUnit.html
[12] http://www.vogella.com/tutorials/JUnit/article.html
[13] http://www.mathcs.richmond.edu/~lbarnett/mcs_dept/junit/junit_intro.html
[14] http://www4.in.tum.de/~schwitze/TUM-I1120.pdf
[15] http://www.javacodegeeks.com/2012/06/google-protocol-buffers-in-java.html

2.4 Development Technique

Scrum, required by the Firm, is an iterative and incremental agile software development framework for managing product development. It defines a work strategy where a development team works as a unit to reach a common goal.[16] In this technique, there are three roles: Product Owner, Scrum Master and Development Team. Product Owner is responsible for communicating the vision of the product to the development team. He determines the various tasks and the time frame for completing them. The Scrum Master works to remove any impediments that are obstructing the team from achieving its goal. The Development Team is responsible for completing work and demonstrating what was built.

The core of Scrum is called sprint, a regular repeatable work cycle. Every new sprint starts immediately after the conclusion of the previous one. The sprint can be cancelled by the Product Owner before the work cycle is over. Sprint planning is a meeting where the team schedules the different task they intend to complete during the sprint. The Scrum Master ensures that the event takes place and the team understands its purpose. During the Sprint Planning, the team needs to decide what to accomplish and how to deliver at the end of the upcoming sprint.

In addition, the team hosts Daily Scrum, usually a fifteen-minute meeting, for them to synchronize activities and plan for the time between now and the next Daily Scrum. During this meeting each team members explains what he did, is currently doing, will do, and what might prevent him from meeting the sprint goal. The Daily Scrum helps the team track progress that trends toward completing the sprint. Moreover, at the end of each sprint, the team needs to review what was done. This review process is called Sprint Review.[17] After which the team should have a Sprint Retrospect to reflect on obstacles and try to make improvements for upcoming sprints.

[16] http://www.scrumguides.org/scrum-guide.html
[17] http://searchsoftwarequality.techtarget.com/definition/Scrum

Chapter 3: Methodology

This chapter goes more in depth, outlining the various steps we took in order to achieve the goal of creating an application that serves as a gateway interface. The gateway interface will act as a communication framework between any internal upstream system and the Firm's reporting database. To achieve the goal, we took the time to outline the following objectives:

1. Understand the pre-confirmation report
2. Understand different technologies associated with the development of the gateway interface
3. Design and develop the gateway interface
4. Develop a system that is isolated and easy to maintain

Below you will find the different methods used to achieve these objectives incorporated in the sprints.

3.1 Pre-confirmation report

To understand the pre-confirmation report, we acquired knowledge of the different client systems involved in generating the report. We gained this knowledge from conducting meetings with numerous employees at the Firm. In addition, we learned the necessary data that should be included in the report. As a result of the Firm's well detailed documentation, we were able to study the flow chart of the existing report structure. Thus, we gained more insight into the changes we would be making to the structure and the benefits of the new gateway interface. Meetings were the most effective way to learn this information, because it enabled us to interact with the Firm's employees frequently.

3.2 Understand the different technologies associated with the development of the gateway interface

After developing a comprehension of the pre-confirmation report, we researched the different technologies that would be used to build the gateway

interface. From initial meetings with the sponsor, we received a list of recommended software and development tools that should be used to build the gateway interface. Since each member had a different level of expertise, we did the necessary research and practiced with tutorials, in order to become adept at using the development tools. In addition, we arranged meetings with a few of the employees at the Firm to learn how to use the existing infrastructure, as this was essential to developing the gateway interface. The gateway interface was built *"by convention not by configuration."* This means that the team needed to include the required dependencies and use modules from the existing application framework rather than write new source code for each component of the gateway interface. This allowed us to work at a more efficient pace and to write code that was clean and easy to read by the Firm's employees.

3.3 Design and development of the gateway interface

First, we examined the existing infrastructure at the Firm to assess which type of design would be the most beneficial to the Firm. Second, we had numerous meetings with the project sponsor to agree on a design that would be acceptable to the Firm. Finally, we met with the different groups from the upstream, downstream systems and the report platform to determine the most effective way to communicate and pass to and from the gateway interface.

After numerous meetings and reaching an agreement on the design, the team began the development process. The development of the components of the gateway interface followed both of the methods listed below:

- Utilizing the existing framework developed by the Firm’s employees to configure various components.
- Writing new Java source code for the components that cannot be configured using the existing framework.

The existing framework, implementing the concepts from EIP, is optimized for message exchange systems, such as our gateway interface. This allows the team to reuse many components from the framework in the gateway interface

implementation. Moreover, it is reliable to reuse the components from the framework, since all of them have been well tested, deployed, and used in development.

Most of the components in the gateway needed to be developed by us. Enforcing test driven development, we developed each gateway interface component in the following procedure. The team first wrote an automated test case to define the desired feature and then write the source code to pass that test. Once the test case was passed, we repeated the process until the gateway interface components were fully implemented.

In addition to using existing framework and developing new components, we also wrote integration test to ensure that all components of the gateway interface were properly connected. The integration test simulated the process by populating mock data into a payload and sending the payload to a JMS queue, the entry point to the gateway interface. Once the payload arrives, the gateway interface would then process the payload with the necessary components. The integration test would pass if the data processing succeeded without causing any error.

3.4 Develop a system that is isolated and easy to maintain

The gateway interface will be placed into production after the completion of the *'OES Decommission Project'*. As such, it was fundamental that the team build a system that was isolated and easy to maintain. The gateway interface we developed does not depend or belong to any other system within the Firm. It was built to facilitate multiple upstream systems that would like to import data into the downstream reporting database.

During the course of the project, we learned that there are two distinct sides within the engineering department of the Firm: *"Build the Firm"* and *"Run the Firm."* The former refers to the side that develops the different products that will eventually go into production; while the latter refers to the team that maintain the products in production. Currently the gateway interface is

developed by the "*Build the Firm*" side. In order to make the interface maintainable for the "*Run the Firm*" side, the team ensured that the system code is readable, well documented, and fully tested with JUnit test cases.

As mentioned earlier, the team built the gateway interface by convention. This allows members from both sides of the Firm to understand the system code, since it used a framework that they were extremely familiar with. This allows them to spend less time understanding the different parts of the system code easily. In addition to building by convention, we had a well-documented code. We did this by creating Javadoc to accompany the system code. Javadoc is a tool that analyzes the documentation comments within the system code and produces HTML files that explain the methods used.[18] This gives the members of the "*Run the Firm*" side an opportunity to read the documentation and understand exactly what the system code does.

3.5 Project Management

As mentioned in the background of this report, the team used Scrum development technique to complete this project. The team ran one-week sprints, and the tasks for each sprint were assigned during the sprint planning meetings held once a week. During the sprint, the team also held fifteen-minute daily scrum meetings. These allowed each member of the development team to synchronize their progress, discuss obstacles, and decide how they will move forward. At the end of each sprint, the team would participate in a sprint retrospect, where they would reflect on the assigned tasks, the challenges they faced during the sprint, the solutions to the obstacles, and the expectations for next sprint

Below is a burndown chart of a specific sprint. The team had a list of tasks which weighted a total of twenty points in difficulty. Ideally, the team should complete four points of tasks at the end of each day. In reality, the team was able to complete five points of the tasks on the first day. On the second

[18] https://docs.oracle.com/javase/7/docs/technotes/guides/javadoc/

day, the team finished four points; on the third day they cut down the total remaining difficulty points to eight, and so on. As shown in the "Burndown Chart-Sprint 2", the blue line shows the ideal burndown rate, and the red line shows the actual burndown rate based on the work that the team completed each day.

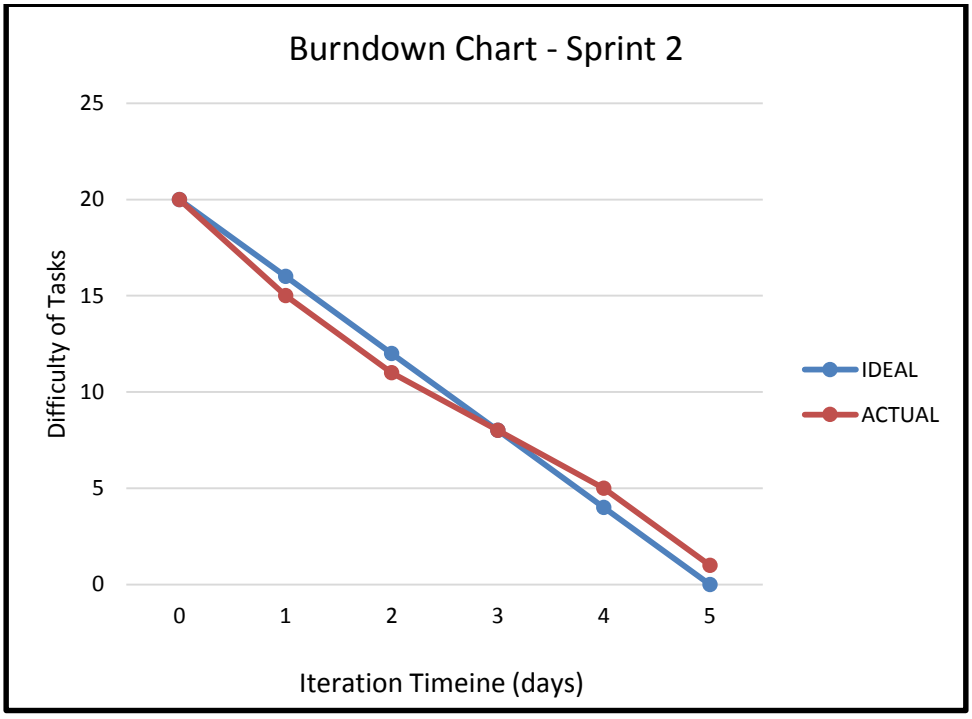

Figure 2: Burndown Chart - Sprint 2

A burndown chart was created at the end of each sprint. Examining the burndown rate every week allowed the team to plan the upcoming sprints more effectively.

Chapter 4: Gateway Interface Design and Process

This upcoming chapter takes a closer look at the current and upcoming process of generating pre-confirmation reports. It illustrates both processes with flow charts. In addition, it explains the role of the different components of each process and examines the design considerations of the gateway interface.

4.1 Current process of generating pre-confirmation reports

As alluded to earlier in the report, pre-confirmation reports are generated manually, and as such they are error prone. In addition, there is also a wait time of approximately fifteen minutes between the time the client request the pre-confirmation report and the time the report is delivered. To date there are approximately ten pre-confirmation reports generated daily. Below is a flow chart that illustrates the current process.

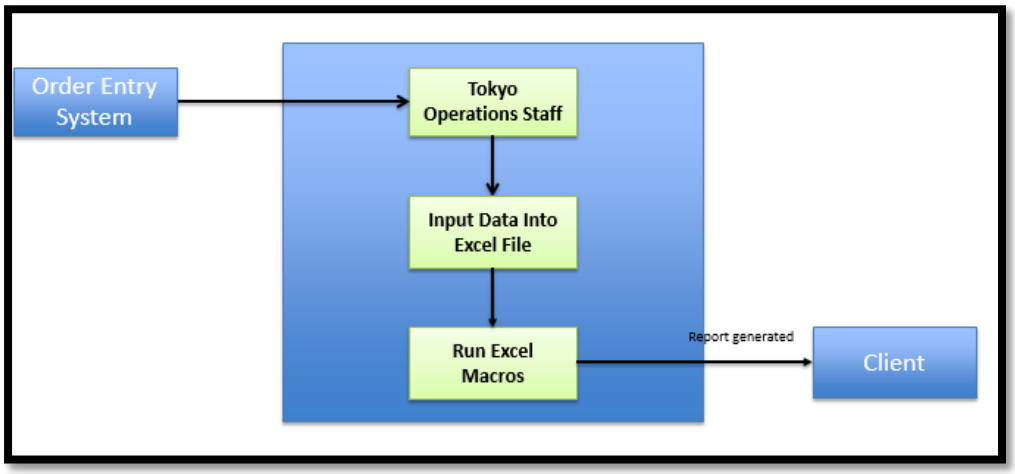

Figure 3: Current Process of Generating a Pre-confirmation Report

When a client requests a pre-confirmation report, an employee from the operation team in Tokyo has to retrieve the trade transaction data from the Order Entry System, as seen in Figure 3. Once the data is ready, the employee then inputs the data into an Excel file and runs the Excel Macros. These Macros are programmed to generate the pre-confirmation report. The excel file containing the pre-confirmation report will then be attached to an email and sent to the client.

4.2 Generating a pre-confirmation report with the gateway interface

The new process, that utilizes the gateway interface, is automated. This process eliminates the possibility of errors. In addition, the wait time for generating reports is reduced significantly, and the client will only have to wait two minutes. This process also utilizes a message-based event driven system. Figure 4 illustrates a flow chart that documents the process of generating a report using the gateway interface.

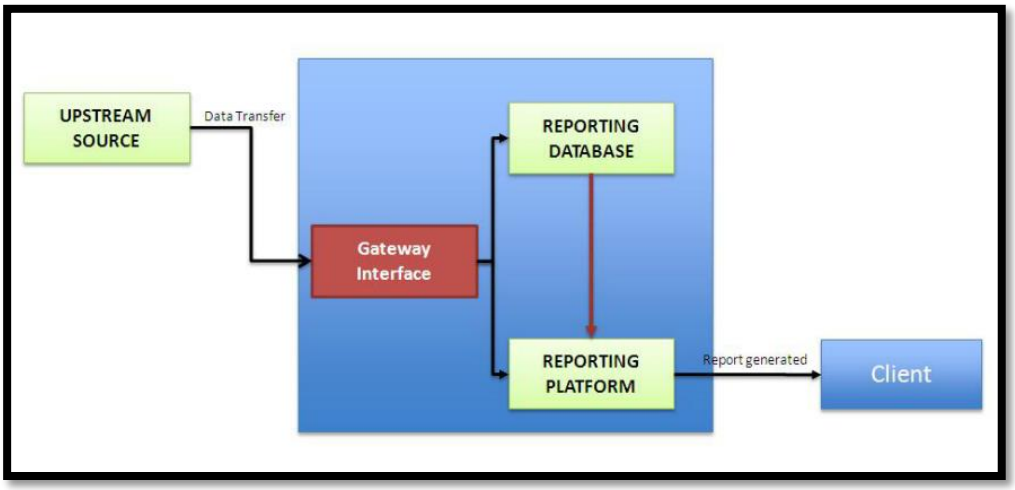

Figure 4: New Process of Generating a Pre-confirmation Report

The work starts with the transfer of the transaction data, from the upstream system. It is transmitted into the gateway interface via a JMS queue. As the data arrives in the JMS queue, it will be processed by the gateway interface. The gateway interface is responsible for, but not limited to: identifying the data type, injecting data into the reporting database, and building a trigger message. The data leaves via a database inserter and a different JMS queue. The database inserter inputs the transaction data into the reporting database and sends a trigger message to the reporting platform as shown in figure 4. The reporting platform refers to the entity within the Firm that is responsible for generating pre-confirmation reports. The new process of generating a pre-confirmation report also makes a connection between the reporting database and reporting platform. Once the reporting platform

receives the trigger message, it will extract the necessary transaction data from the reporting database and create a pre-confirmation report. The flow chart below outlines the different processes within the gateway interface.

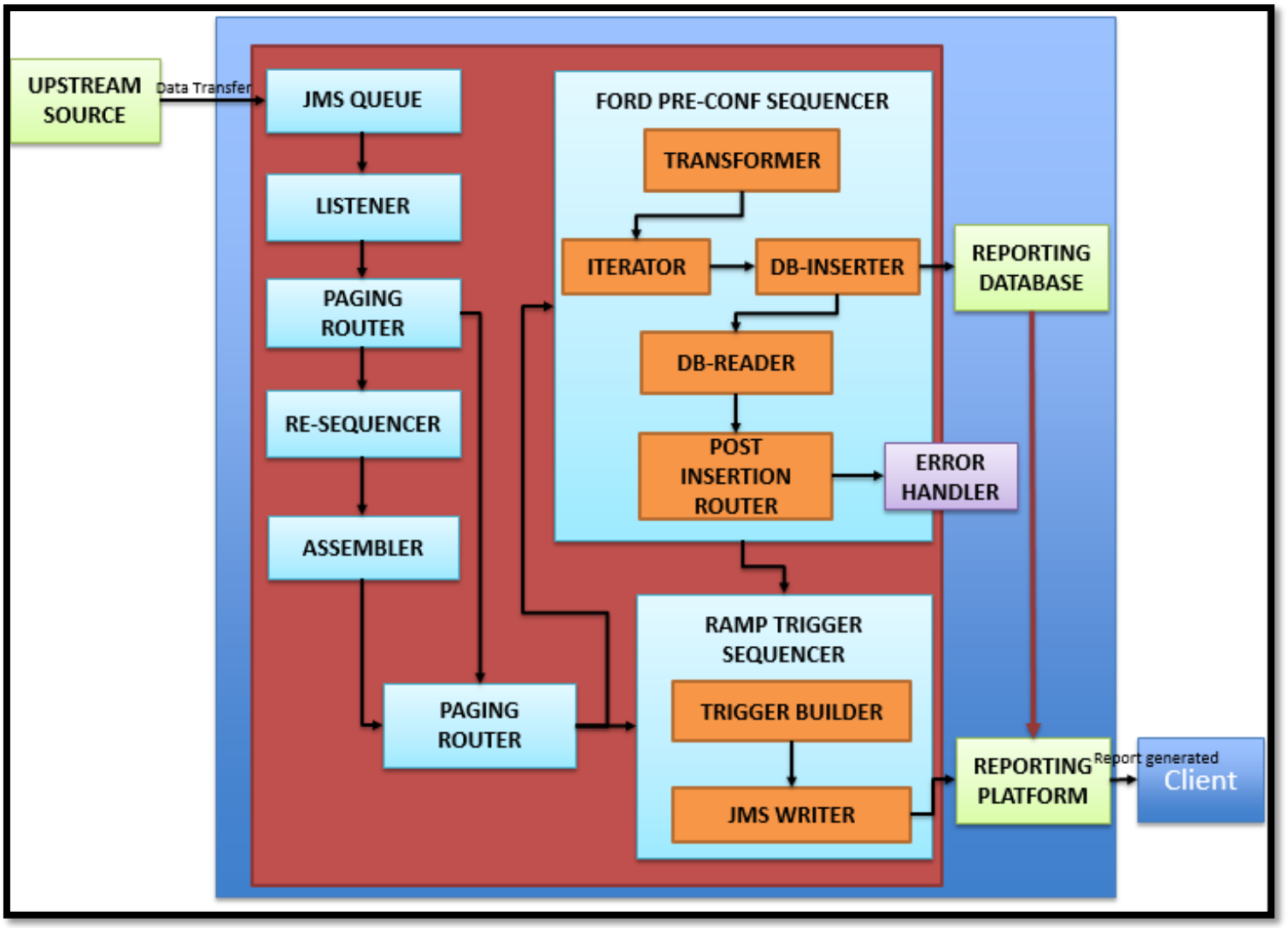

Figure 5: Gateway Interface

To efficiently build a gateway interface that allows communication between upstream and downstream systems, the team designed an Application Program Interface (API). Designed with Google Protocol Buffers, the API sets the standards for the messages sent from the upstream systems in a payload. The payload contains the metadata and the message body. The API requires the

upstream systems to populate the payload with transaction data before sending it to the gateway interface through the JMS queue.

JMS queue is the entry point into the gateway interface. The queue is an instance of a message channel that allows payloads from multiple upstream systems to be delivered to the gateway interface. With the help from Barclays' software developers, we established a JMS queue in the server (PDQ Solace Instance). The queue is set up with a maximum size of five megabytes. This requires upstream systems to divide their data into multiple payloads, when the size of the data being transmitted exceeds the size limit.

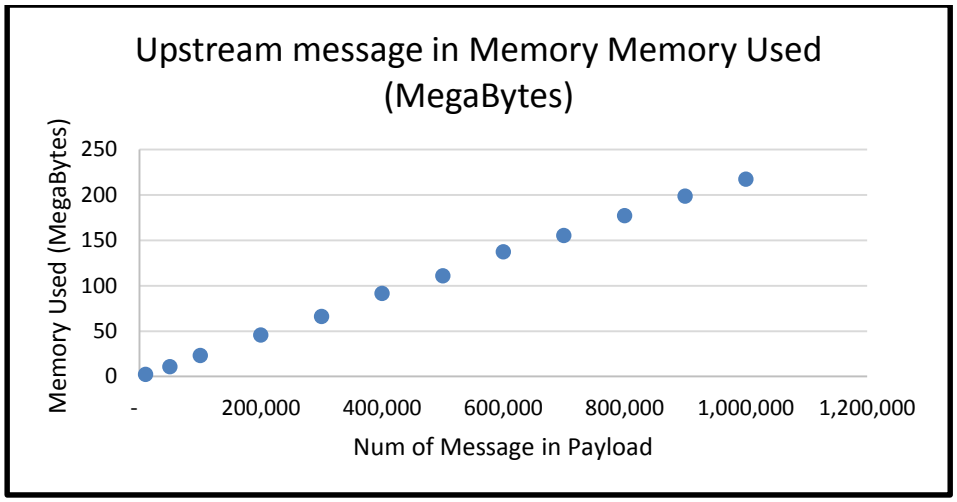

Figure 6: Upstream Message in Memory Used

In addition to the size limit of the JMS queue, we also took into account the memory used in JVM. The scatterplot in Figure 6 shows the relationship between the memory of the JVM and the size of the payload. The data were collected by running a test that populated a payload with various numbers of data, or messages, and observing the program with JConsole, a JVM monitoring tool. The purpose of this test was to ensure that the payload was within the size limit of the JMS queue, does not exceed the memory used in the JVM.

The payload defines the format of the data that the gateway interface expects to receive from upstream systems. It contains the metadata and the message body. The information in the metadata contains logical dataset identifiers that help the different components of the gateway interface determine how to process the transaction data. For example, it requires the upstream systems to list the message type, a sender reference, business date, etc., as seen in Figure 7. The metadata is also designed to contain the essential information for triggering a report, which will be discussed later in this chapter. The message body is essentially a collection of transaction data. This transaction data will not be processed by the gateway interface. The gateway interface is responsible for sending the transaction data to the different handlers that will process it. Figure 7 is a copy of the payload as a *.proto* file.

*core.proto

```
1  package odh.model;
2
3  import "descriptor.proto";
4  option java_package = "com.barcap.dds.model.protobuf";
5  option java_outer_classname = "DDSGATEWAY";
6  message Metadata {
7      //For the Payload Router
8      optional string classname = 1;
9      optional bool is_replay_message = 2;
10     optional int32 version = 3;
11     //For the Focus Message Collector (paging router)
12     optional int32 total_num_pages = 4;
13     optional int32 page_num = 5;
14     //For the RAMP Trigger Builder
15     optional string party_id = 6;
16     optional paty_role_enum party_id_role = 7;
17     optional string sender_reference = 8;
18     optional int64 business_date = 9;
19     optional int64 event_timestamp = 10;
20     optional string feed_name = 11;
21     optional string region_code = 12;
22     optional string source_system_code = 13;
23     enum paty_role_enum {
24         efam_acct = 0;
25         efam_acct_group = 1;
26         efam_client = 2;
27         sender_defined = 3;
28     }
29
30 }
31
32 message Payload {
33     repeated bytes msg = 1; // collection of messages
34     optional Metadata metadata = 2;
35 }
```

Figure 7: Required Format from Upstream Systems

The illustration shows that the payload is divided into four sections. The first three sections define the metadata and the last section comprises of both

the metadata and the message body. The first highlighted section (lines 6 – 10) will be used by the payload router, the second highlighted section (lines 11 – 13) will be used by the paging router, and the third highlighted section (lines 14 – 31) will be used by the trigger builder. Once the payload is fully populated, it is sent through the JMS queue and captured by the listener.

The listener is configured by utilizing the Firm's existing framework. The listener's purpose is to wait on the incoming payload as it enters the JMS queue. Once a payload arrives, the listener notifies and forwards the payload to the paging router, as shown in the figure 5.

The role of the paging router is to examine the total number of pages represented by the field, *total_num_pages*, in the metadata and decide whether the message should be sent to the re-sequencer or to the payload router. The *total_num_pages* field is populated by the upstream system. If the field contains an integer greater than one, this means that the gateway will receive multiple payloads for the same logical dataset. A logical dataset is a collection of transaction data that belongs to one client. Then the payloads are passed down to the re-sequencer. On the other hand, if the field is not populated or is equal to one, then the gateway will only expect to receive one payload for the logical dataset. In this case, the payload is sent to the payload router.

In the event that the paging router expects to receive multiple payloads, each payload is sent to the re-sequencer. The purpose of this re-sequencer is to allow the upstream system's data to be passed down when the size of the data exceeds the size limit of the JMS queue. The re-sequencer is designed to cache the payloads until it receives all that belong to the same logical dataset. The re-sequencer then reorganizes the payloads from the upstream system, based on the *page_num* field in the metadata. The re-sequencer utilizes four strategies: correlation, sequencing, identity, and release. The correlation strategy defines a function that identifies all the payloads that belong to the same logical dataset. It correlates the payloads based on the *party_id* in the metadata. Since the payloads may not come in order, the sequencing strategy is designed to reorder

them by looking at the *page_num* field in the metadata. The identity strategy defines how the re-sequencer can uniquely identify a payload by examining the universally unique identifier of the metadata, which is the *party_id*. The release strategy checks whether or not all the payloads have arrived. If so, the re-sequencer will hand over the upstream collection of payloads associated with one logical dataset to the assembler.

The assembler then receives the collection of payloads that belongs to the same logical dataset and concatenates them into one new payload. It will first create a message builder designed to construct a payload. Then the assembler iterates through the collection of payloads and extracts the metadata from the first payload, as well as the message body from each payload. It will use the convenient function in the payload protocol buffer builder to create the new payload. This new payload contains the necessary information that will be used by the remaining components of the gateway interface, and it will be sent to the payload router.

The payload router will receive one payload from either the paging router or the assembler. The payload router consists of an advisor and the processing rules. This router uses an advisor to process the payload and stores the metadata in the context. The processing rules are stored in a map. It maintains Java string constants as the keys and chain event handlers as values. The advisor informs the router what to do with the message body in the payload. The advisor is programmed to examine the metadata and check for the payload's originating upstream system. Once the advisor identifies the upstream system, it returns a string as the key to the router. The router then retrieves the corresponding chain event handler with the given key to continue the procedure. Additionally, the advisor is able to check whether a specific payload has been sent multiple times, known as a "*replay message*". In the event that a replay message occurs, the payload router will direct the payload to a different chain event handler.

Currently the gateway interface contains two chain event handlers (CEH), Reporting Database Pre-Conf Sequencer and Reporting Platform Trigger

Sequencer. A CEH is made up of numerous handlers that perform different tasks, in sequence, on the payload. Each handler has a generic design, with the exception of the Transformer. This was done so that they can be used in multiple CEHs. The Reporting Database Pre-Conf Sequencer includes five handlers: Transformer, Iterator, Database (DB) Inserter, DB Reader, and Post Insertion Router. The Reporting Platform Trigger Sequencer contains Trigger Builder and JMS Writer. The chain event handlers are configured in an xml file, using handlers built by the team and existing framework built by the Firm's employees. Each handler logs the events at a different interest level to the server, so that the processed events can be accessed at a later time.

The Transformer is responsible for converting the collection of transaction data from the upstream system to an acceptable data format for the reporting database. It does this by preforming a mapping from upstream to downstream transaction data. The mapping uses the convenient functions from two *.java* files, which are generated from compiling the *.proto* system files. The first java file written by the team is a simple transformer that only handles one transaction message. Once the team successfully built the simple version of the Transformer, the second Transformer was built. It utilizes the simple transformer to convert a collection of transaction messages. The second transformer uses the system code from java files that are generated by compiling the Google Protocol Buffer *.proto* files. The transaction data from the upstream system contains data fields that are different from those required by the reporting database. The downstream transaction data, or converted structures from the upstream transaction data, are convenient for inserting the transaction data into the downstream reporting database. The input of the transformer is a single payload, and the output is a collection of transaction data, which will be passed to the iterator.

The Iterator is designed to filter through the given collection of transaction data and delegate the task of handling each message to the DB Inserter. As a result of this the Iterator and the DB Inserter function synchronously. The DB Inserter picks up the current message that the Iterator

is processing and inserts the message directly into the client reporting database, the downstream system. Both the Iterator and the DB Inserter are configured from the existing framework developed by the Firm's staff. Once the DB Inserter has inserted the entire collection of transaction data, the DB Reader will begin processing the payload.

The DB Reader performs the data validation of the insertion in the reporting database. It executes a stored procedure in the reporting database environment. A stored procedure is a function, stored in the reporting database. This function will return a value indicating the status of the data insertion. More specifically, it will reveal whether or not the transaction data was successfully inserted into the reporting database. This stored procedure will return an integer value of zero, indicating data insertion has succeeded without any error. Otherwise, it will return an error code signaling the exact error that has occurred during the data insertion. The error code is an integer of any value except zero. This value will be sent to the Post Insertion Router.

The Post Insertion Router determines whether or not to build the trigger message for the reporting platform. Similar to the other routers mentioned above, it has an advisor and processing rules. The advisor examines the return value from the DB Reader and returns a String key to router. If the value zero is received, the process will move on to the CEH that is responsible for creating the trigger message. In the event that the router receives an error code, it will invoke the error handler with the error code. The errors are currently dealt with by a system developed by the Firm.

The Trigger Builder handler references a trigger template and compiles a trigger message, based on the subscriptions outlined by the reporting platform. The roles of the Trigger Builder include: examining the metadata in the context that is set available by the Router, extracting the essential data in the metadata, and composing the trigger message by loading the data into the template. The output of the builder is a String representing the XML trigger message that will trigger the reporting platform to generate the desired pre-

confirmation report. After the trigger message is composed, it is sent to the reporting platform, via a JMS Writer.

The JMS Writer sends the trigger message to a different JMS queue connected to the reporting platform and eventually delivered to the other JMS queue for processing. This queue is created by the reporting platform team. Based on the contents in the given trigger message, the reporting platform will extract the data from the reporting database and generate the pre-confirmation report with the data.

Once the reporting platform receives the trigger message in their JMS queue, it will extract the transaction data from the reporting database based on the information specified in the trigger message. This transaction data will be used to generate the pre-confirmation report, after which the pre-confirmation report will be emailed to the client. This process takes approximately 2 minutes.

4.2.1 Design Considerations

We had to take many considerations into account when designing the gateway interface. This section will explain the different considerations the team took.

File-Based versus Message-Based

The team first reviewed the approach of using a file-based polling system, before deciding to go with a message based system. Filed based polling sends a batch of files based on a **timer**, while a message-based system is event driven. The approach of using a file-based polling system was proposed by the sponsor, because there had been such a system developed by the Firm previously. The developers chose to make the file-based system polling timer based, rather than event driven, in order to fulfill other requirements within the Firm. If we adopted this approach, it would allow us to shorten the development process by simply building infrastructure to use the polling system. Although both systems would automate the process of generating a report, the team decided to use a message-based event driven system. We chose this because it is

important to reduce the wait time of generating pre-confirmation reports, which the polling based system would fail to do. The event driven system reduces the wait time by utilizing a **message listener**, which processes the message upon arrival in the JMS queue. Reducing wait time in the automation process is a part of the Service Level Agreement.

Bridge between multiple upstream systems and the reporting database

The team was tasked with automating the process of generating pre-confirmation reports. We decided to build a gateway that will automate the process, and also serve as a bridge between any potential upstream system and the reporting database. That is, it is not only designed to accept data from the upstream system that is responsible for pre-confirmation reports, but it is also designed to accept data from any upstream system that wants to input data into the reporting database in the future.

Multiple downstream systems

As part of the task to automate the process of generating pre-confirmation reports, the transaction data will be saved to the reporting database. This is the only downstream system that the gateway interface currently interacts with. However, the team designed a highly customizable gateway interface. This aims to allow future developers to direct data coming from upstream to any downstream system, by simply adding new component to the gateway interface without heavily modifying the existing code.

Separation of Data Model and Gateway Interface Instance

The team used Google Protocol Buffers to define the data model used in the gateway interface. One benefit of using Google Protocol Buffers is that the data format can be compiled into source code in multiple languages, including Java, C++, and Python. To take advantage of this feature, the team created two projects in Eclipse; one contains the data model, while the other consists of all components of the gateway interface. The data model project is added to the gateway interface project as a Maven dependency. This separation gives the gateway interface more flexibility. First, the data model can be compiled into source codes in other languages, which allows for systems that are developed

in languages other than Java to interact with the gateway interface. Futhermore, future developers will be able to add new data formats that they desire the gateway interface to be processed in the data model project, without having to modify the code in the gateway interface project.

4.2.2 Benefits of the gateway interface

As a product that is delivered to the real world, the gateway interface has several benefits to the firm and its clients.

Benefits of the Gateway Interface:

- Automates the process of generating pre-confirmation report
- Reduces the time of generating the pre-confirmation report from about fifteen minutes for each report down to two minutes for all reports
- Give clients the ability to approve their stance on the offsetting trading transactions before the data is submitted to the books and records system
- Allows multiple upstream systems, rather than one, to send data to downstream reporting database.
- Strengthens the relationship between Barclays and their clients in Tokyo, who frown on revisions in books and records system

Chapter 5: Future Recommendations

In this chapter, we will discuss our recommendations for improving the existing gateway interface. After reviewing our design, we developed the following list of recommendations for future developers that will continue to develop the gateway interface.

1. Implement a feedback system that sends acknowledgement and non-acknowledgement message based on the sender of the payload. The acknowledgement message will state that the gateway interface was able to process the data, while a non-acknowledgement message will state that we receive the data but was unable to process it. Moreover, non-acknowledgement messages can be implemented with error code, so that the upstream system will be able to identify what error has occurred in the gateway interface and adopt certain actions, such as republishing the data.

2. Implement replay message table that stores a subset of data fields from the metadata of the payload, which can identify the logical dataset the payload belongs to, when previous data has been processed. This is useful for processing the “replay message”. Since this payload contains data that have been already stored in the reporting database, the table can be used to verify if the previous appearance of the payload, so that the gateway interface can create the trigger message without saving the data in the payload into the reporting database.

3. Create new transformers specific to other upstream systems. Currently the gateway interface only facilities converting data from one upstream system.

4. Create multiple trigger builders and templates that can be used with a client id, and account group rather than party_id. This will allow the

gateway interface to send multiple types of trigger messages to the reporting platform.

Chapter 6: Conclusion

In seven weeks, we were able to complete a conceptually fascinating and yet intellectually challenging project of building a gateway interface. At the beginning of the project, there was a steep learning curve of understanding the task and the expectation of the project. Also, we familiarized ourselves with the existing framework and the technology associated with the project. The customized Scrum method that we utilized assisted us to create a deliverable product within the time frame.

The task we were given upon arriving at the Firm was to design a gateway interface, which functioned as a communication tool between **one** potential client system and the reporting database. As the project expands, the team collects more requests from different teams. The team set the goal as to create this gateway interface that allows **multiple** upstream system to send data to the reporting database. We were able to achieve this very technical goal. This gateway interface not only eliminates the Firm making changes to the books and records system, but also reduces the time for generating the pre-confirmation report.

At the end of the project, we also created an existing framework training tool for the Barclays Futures Clearing Department and a handover document for the developer who continues to advance this gateway interface. A sample from the handover document can be seen in Appendix B. The training tool explains the operation of the gateway interface in technical detail, so that future developers could understand this project that uses EIP and build on to the interface that we designed.

As we reflect on this project, we realize that were able to put the theories and ideas we learned at WPI into practice at Barclays. When we arrived at the Firm, we were required to work with teams of different disciplines. We were able to work efficiently with the teams, since we have trained to work in a project based environment at WPI. This gateway interface plays an important

role in strengthen the relationship between Barclays and its clients in Tokyo. We feel honored to be given the opportunity to have this global impact.

Appendices A: Glossary of Terms

Term	Explanation
Client	Trader
Upstream System	The Firm's entities
Client System	The Firm's entities
Reporting Database	An oracle database within the Firm, which is used to store data coming from upstream systems
Reporting Platform	An entity within the Firm that responsible for generating pre-confirmation reports
Gateway	Communication Channel between multiple upstream systems and the reporting database
Logical Dataset	A collection of transaction data that belongs to one client
By convention not by configuration	The team needed to include the required dependencies and use modules from the existing application framework rather than write new source code for each component of the gateway interface
Build the Firm	The side of the Firm that develops the different products that will eventually go into production
Run the Firm	The side of the Firm that maintain the products in production.

Appendices B: Sample from Handover Documentation

The metadata contains the data fields hat help the different elements of the gateway interface determine how to process the payload. In addition. It also consists of the data that is necessary for populating a trigger message. All fields in the metadata should be populated by upstream source. The following are the fields in the metadata:

1)	*classname*	String	Class name of the proto buffer message
2)	*is_replay_message*	Boolean	Flag indicating if the payload is a replay message
3)	*version*	Integer	Key determining if this dataset is related to another. (eg. If pre-conf report (version1) is sent to the client and he/she disagrees with the information. The client will request another report (version 2) with accurate information)
4)	*party_id*	String	Identification of a logical dataset. This is used by the gateway interface, it associates the message body with an account ID
5)	*party_id_role*	Enumeration	Type of the *party_id*
6)	*sender_reference*	String	Reference to the sender of the payload. This is for the reporting platform to track back to the sender in case of an error. This is a string that uniquely identifies the message that was sent. This can be thought of as a sender system ID.

7)	*business_date*	Integer	Business date for the dataset. This is used to create the trigger message for the reporting platform.
8)	*event_timestamp*	Integer	Timestamp populated by sender. This is used to create the trigger message for the reporting platform.
9)	*feed_name*	String	Name of the feed/trigger. This is used to create the trigger message for the reporting platform.
10)	*region_code*	String	Where the subscription comes from. This is used to create the trigger message for the reporting platform.
11)	*source_system_code*	String	Code of the upstream system. This is used to create the trigger message for the reporting platform. This is a string that uniquely identifies the upstream system that sent the payload. This can be thought of as sender system.
12)	*total_num_pages*	Integer	Number of pages the dataset is divided into. This is used for the re-sequencer.
13)	*page_num*	Integer	Current page the data in the payload message represents. This is used for the re-sequencer.

The message body is essentially a collection of transaction data from the upstream system. With the current contract, all transaction data in the payload message will be saved into a table in the reporting database. The message body is represented as byte array in the message.