Project Number: AG-WS07

Bank of America: Using Technology to Mine and Analyze Data from TRACE

A Major Qualifying Project

submitted to the faculty

of the

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Brent Gilmore

Andrei Paduroiu

Date: December 14, 2006

Approved by:

Professor Arthur Gerstenfeld, Co-Advisor

Professor Michael J. Ciaraldi, Co-Advisor

Abstract

Sponsored by Bank of America, this project mined raw data regarding corporate bond trades; these statistics help traders better understand the bond market and receive more trading ideas, faster. An application was designed to create reports composed of this data that serve as analytical tools for the traders. Upon its completion, the project was adopted by several traders and should result in significant financial savings for the company.

Executive Summary

The credit and equities traders at Banc of America Securities require many applications to filter and interpret information relevant to the securities they are interested in trading. Specifically, the bond traders are concerned with the activity in the corporate bond market and are looking for statistics that can provide them with relevant knowledge, which can then be used to interpret market activity. The National Association of Securities Dealers implemented an automated system, the Trade Reporting and Compliance Engine (TRACE), which records bond transaction details for all corporate bond trades and then disseminates it to the subscribers of its Bond Trade Dissemination Service. As it is reported to traders, this data is in a very raw form and is not especially useful in the analysis of the market. Therefore, the need for an application that can present this information in a human-readable form arose. Our project aimed to satisfy this need through condensing and organizing trade data into easy-to-read reports.

To best develop an application to suit these needs, we needed to gain a better understanding of the data set and the technology with which we would be working. We read articles pertaining to investing in bonds and TRACE, and we also met with knowledgeable Bank of America employees that helped us design our application.

Consequently, we created *TraceMon*: a small, but very useful application which, based on user preferences, searches and interprets information from TRACE and presents it in an easy-to-read form. It is able to generate three types of reports that pertain to trading activity for particular credits: Daily Trade Summary, Liquidity Report and Trade Outliers. *TraceMon*'s users can personalize each of these reports through a friendly graphical user interface, as well as schedule reports to run periodically or on-command.

TraceMon is able to synthesize vast amounts of information into short and clear reports. It provides the user with statistics that would otherwise be too time-consuming and unrealistic to determine. The Daily Trade Summary provides the user with a basic trading overview of the previous day's trading activity. It is valuable to traders since, by using it, they can quickly assess the prior day's trading activity and pricing action for select issuers. The Liquidity Report allows traders to get a quick perspective on the average monthly trading volume for select issuer's bonds in each of the last twelve months. Lastly, the Trade Outliers report acts as a comparison tool, where the user can plainly see how recent trading activity compares to historic values. This serves as an effective way to monitor the market and quickly notice trading volume outliers.

The creation of *TraceMon* provided bond traders at Bank of America with a very useful tool. Previously, the data reported by our application was either tediously searched for or simply not used. By processing a vast amount of information and condensing it into straightforward reports, this program quickly provides statistics to the user that would otherwise be unrealistic to determine. With the advent of *TraceMon*, this valuable data is now easily accessible and can provide the user a great deal of insight into the corporate bond market. Because of this insight, traders are able to make better investment decisions and therefore increase overall profitability of the credit trading department.

TraceMon comes with a thorough documentation, containing both a detailed user manual and a software development guide. Together with the comprehensive code comments and scalable architecture, these documents make TraceMon an application that is very easy to understand from both development and user points of view. Extending the existing functionality will not imply re-architecting or significantly modifying the code; instead, with the help of the

documentation we created it should only take a short amount of time for a developer to be able to fully support and modify *TraceMon*.

Despite the value it presently adds to bond trading operations at Bank of America, future measures could be taken to further enhance the effectiveness of *TraceMon*. The three reports currently generated by it are only the beginning of what could be a much larger TRACE analytics package. Also, this larger collection of reports could be classified by frequency so that reports whose data changes daily would be generated more often than those whose output changes less regularly. Since some reports can be fairly large and thus harder to read, changes can be made in future versions to enhance the way the user accesses the desired information. Additionally, if *TraceMon* is adopted by multiple users, further support could be added in order to reduce report redundancy.

Acknowledgements:

We would like to thank Scott Burton, Martin Gonzalez, Alex Gregory and Kurt Vile at Bank of America for their assistance, support and guidance. We would also like to thank Jian Huang, Davran Muzafarov, Brad Menoche, Jing Shih, Tom J. Spak, Ellen Tsai, Charles Waddington, Jason Wang and Igor Zitser. Additionally, we would like to recognize the Professors Michael J Ciaraldi and Arthur Gerstenfeld from Worcester Polytechnic Institute for their involvement and counsel regarding our project.

Table of Contents

Abstract i
Executive Summary ii
Table of Contents vi
Table of Figures viii
Table of Tables ix
1. Introduction 1
2. Background 3
2.1 Bank of America Overview 3
2.1.1 Banc of America Securities LLC 3
2.2 Corporate Bonds 4
2.2.1 Static Bond Characteristics 4
2.2.2 Dynamic Bond Characteristics 5
2.2.3 Ratings 6
2.3 Financial Regulatory Agencies 9
2.3.1 Securities and Exchange Commission 10
2.3.2 National Association of Securities Dealers 10
2.3.3 Trade Reporting and Compliance Engine 10
3. Methodology 14
3.1 Research 14
3.1.1 Literature Study 14
3.1.2 Meetings 15
3.2 Analysis 15
3.2.1 Requirements 15
3.2.2 Data Mining 16
3.3 Architecture and Design 16
3.4 Development 17
3.5 Testing 18
3.6 Release 18
4. Results 20
4.1 Information Gathering 20
4.1.1 Stakeholder Meetings 20
4.1.2 Informational Meetings 20
4.2 Requirements 21
4.3 Data Mining 24
4.4 High Level Architecture 25
4.5 Development 29
4.5.1 Sketching the flow of events 29
4.5.2 Actual Development 31
4.6 Testing 36
4.6.1 Quality Assurance 36
4.6.2 User Acceptance Testing 37

4.7 Final Product – *TraceMon* 38
4.7.1 Configuration Mode 38
4.7.2 Execution Mode 42
4.7.3 Advanced Options 46
4.7.4 Identifying and fixing problems 48
4.7.5 Documentation 50

5. Conclusions and Recommendations 52
5.1 Improve Estimated Value Accuracy 52
5.2 Diversify Reports and Organize Them 53
5.3 Expand Report Format 54
5.4 Other Suggestions 55

Appendix A: Meeting Minutes 57
Initial Project Meeting 57
NASD Call 58
ALICE Database Conference Call 59
TraceMon Demonstration 60
TraceMon Release for User Acceptance Testing 61
The Future of *TraceMon* 62

Appendix B: Diagrams 63

Bibliography 69

Table of Figures

Figure 4.1: Configuration Mode Screenshot 1 39
Figure 4.2: Configuration Mode Screenshot 2 40
Figure 4.3: Configuration Mode Screenshot 3 41
Figure 4.4: Daily Trade Summary Sample Output 43
Figure 4.5: Liquidity Report Sample Output 44
Figure 4.6: Trade Outliers Sample Output 45
Figure 4.7: EventLogger Screenshot 1 49
Figure 4.8: EventLogger Screenshot 2 49
Figure B.1: Package Diagram 63
Figure B.2: Class Diagram 64
Figure B.3: Sequence Diagram for Execution Mode 65
Figure B.4: Sequence Diagram for Configuration Mode 66
Figure B.5: Use Case Diagram for Configuration Mode 67
Figure B.6: Flow of Execution Diagram 68

Table of Tables

Table 2.1: S&P Ratings 7
Table 2.2: Moody's Ratings 8
Table 2.3: Fitch Ratings 9

1. Introduction

An important aspect of any financial investment is information. This information can pertain to the specific investment, the type of investment, and the market for that particular investment. Investing in corporate bonds is no exception, although market transparency has only developed in the past five years. Previously, information about corporate bond trades in the secondary, or over-the-counter (OTC), market was not disseminated, making it difficult for investors to gauge market conditions. With the development of the Trade Reporting and Compliance Engine (TRACE) by the National Association of Securities Dealers (NASD) in 2001, trade information has become widely available. However, the mere existence of this data is not necessarily enough to fully benefit investors.

At Bank of America, credit and equities traders are looking for any information that will help them be more successful. With multiple sources of many types of data available to them, it is sometimes difficult to sift through all of the statistics to find those which are most useful to these traders. The Credit Technology division of Bank of America develops applications to help traders filter and analyze this information. This project involves the development of one such program that organizes TRACE data and presents it in a useful form to bond traders.

The main focus of this project was to generate reports that would present TRACE data to bond traders in its most useful form. The application that was developed as a result of this project collects data from multiple sources and presents it in useful reports that are generated on a nightly basis. Through their use of this program, bond traders are able to better analyze trading activity from the previous day and use that information to make investment decisions.

In order to develop this application, a series of steps were undertaken. We first did research to better understand the TRACE data set and the implications of our project. Next, we

carefully examined several Bank of America databases to determine which data was most accurate and where to obtain it from. We then created the application and revised it repeatedly based on user feedback. Through this process, we were able to give the bond traders at Bank of America a useful tool for studying the corporate bond market. While this is only the initial tool developed at Bank of America for analyzing TRACE data, it could end up being part of a larger TRACE analytics package in the future, which contains multiple tools that process raw data and present it in a useful form.

2. Background

This section contains a brief history of the Bank of America Corporation, and because our project heavily involved several different aspects of credit and equities trading, they are also detailed in the following sections. Furthermore, a history of the NASD is included, as it is the regulatory agency for the corporate bond market.

2.1 Bank of America Overview

Bank of America is the resultant firm of multiple mergers and acquisitions of large banks. The two principal organizations involved in the largest merger were NationsBank and the California-based Bank of America, forming the current Bank of America in 1998. NationsBank dates back to 1874 when it was known as the Commercial National Bank of Charlotte (CNBC). Through a large number of mergers and acquisitions, CNBC evolved into North Carolina National Bank, and then NationsBank. The California-based Bank of America began as Bank of Italy in 1904 in San Francisco established by A.P. Giannini. This bank grew rapidly for most of the 20th century and financed many companies in the agriculture, wine and motion picture industries, including many Walt Disney projects. Currently, Bank of America has refocused its growth strategy from one of acquisition to one looking for an organic growth through deeper customer relationships [Bank of America].

2.1.1 Banc of America Securities LLC

A subsidiary of Bank of America, the securities division represents the investment banking part of the bank's overall operations. Banc of America Securities (BAS) offers services including trading, brokerage, debt and securities underwriting and research, as well as advice on large financial transactions. BAS works mostly with corporations, institutional investors, and government entities [Yahoo Finance]. The Credit Technology division (CT) develops and

maintains applications for credit traders as part of the Global Credit and Equities Technology division of BAS. We worked closely with CT during our project.

2.2 Corporate Bonds

Corporate bonds are issued by both public and private corporations to raise money to fund projects and company expansion. This is done when an investor lends money to the issuer in exchange for a bond promising to return the funds on a specified maturity date plus interest payments. Most bonds are assigned ratings by agencies based upon the probability of the issuer defaulting on payments. From these ratings, the bonds are then segregated into two categories: investment grade bonds and high yield bonds. Investment grade bonds consist of those within the four highest rating categories, and the remaining bonds are classified as high yield. After their initial issuance, corporate bonds are traded mainly in the secondary, or Over-the-Counter (OTC) market. This market is made up of many locations across the United States and around the world, and the bonds are usually traded electronically or over the phone.

2.2.1 Static Bond Characteristics

Corporate bonds are differentiated through a few key statistics and identifiers. The most basic of these is the ticker, which corresponds to the issuer of the bond. The symbol of the bond is more descriptive as it contains two parts: the first part is the ticker and the second part is a unique two-letter identifier for each bond separated by a period. Therefore, a standard symbol will look like ‘ABC.DE’ where ‘ABC’ is the ticker symbol for the issue and ‘DE’ corresponds to the particular bond. Another identifier that is unique to each bond is the Committee of Uniform Securities Identification Procedures (CUSIP) number. This is a nine character number where the first six identify the issuer and the last three identify the issue, similar to the symbol. A standard CUSIP number may look like ‘123456AB1’.

Aside from these classifications, each bond has certain characteristics that are determined at issuance and will not change during the life of the bond. One such attribute is the maturity date, which is the date when the principal, or face value, of the bond must be repaid. The coupon rate is another important classification, as it tells the percentage rate of interest, which is usually paid out semi-annually. This number can change if the bond is a floating rate security, as bonds of this type have coupon rates that are periodically adjusted according to a predetermined formula. Other possible bonds characteristics include a call or put feature. A callable bond is one that may be redeemed prior to its maturity by the issuer. A bond with a put option enables the investor to demand repayment of principal prior to the bond's maturity.

2.2.2 Dynamic Bond Characteristics

There are also several features of corporate bonds that are constantly changing. It is these types of dynamic attributes that make corporate bonds appealing or unappealing for investment. The two main features of a corporate bond are price and yield. What is called the yield is most often the yield to maturity (YTM), as it is essentially the entire return the investor will receive for holding the bond to maturation. The price is largely determined by the bond's coupon rate and its relation to prevailing current rates. A bond sells at a premium if it is priced higher than its par value or at a discount if it is priced lower than its par value. The former situation arises when the bond's coupon rate is higher than prevailing interest rates, and the latter situation occurs when the opposite is true. The price is then used to calculate the yield, creating an important relationship between these two attributes.

Yield spreads are another element of corporate bonds; they give investors an indication of the relative risk of their investment. For this project, we were dealing predominantly with spreads comparing corporate bonds to a U.S. Treasury security with a similar maturity date. This

Treasury spread number is calculated by subtracting the yield for the Treasury security from the yield of the bond. The result is usually displayed in basis points (bps), where one basis point is equal to 0.01%. For example, if the 2-Year Treasury bond yields 5.0% annually, and some bond that matures in 2 years yields 5.25%, the spread is 0.25% or 25 basis points. This number is very dynamic however, because the Treasury securities' yields are constantly changing based on corresponding yield curves. Therefore, to calculate a Treasury spread for a certain trade, the Treasury yield at the time of trade execution must be used.

2.2.3 Ratings

As mentioned previously, bonds are assigned ratings based on their creditworthiness. There are two principal ratings companies: Moody's Investors Service and Standard & Poor's (S&P). Fitch Ratings is a third ratings agency but it is not nearly as influential as its counterparts. All ratings have similar properties and most bonds tend to receive comparable ratings from all three agencies. The following section contains a brief description of each firm's ratings.

Standard & Poor's

Standard & Poor's rating system values companies from AAA for the best quality to D for those in default, plus other ratings for companies in certain situations. S&P also has an intermediate rating system composed of the standard ratings with a '+' or '-' to further describe the company's current situation. These intermediate ratings are only used for ratings between AA and B.

Rating		Description
Investment Grade	AAA	Prime bonds. Maximum safety for investors.
	AA	High grade bonds. A high quality investment.
	A	Upper medium grade bonds.
	BBB	Lower medium grade bonds.
High Yield	BB	More prone to changes in the economy. Considered slightly speculative.
	B	Highly speculative bonds. Financial situation varies noticeably with economy.
	CCC	Substantial risk. Issuer usually in poor standing.
	CC	Extremely speculative and vulnerable bonds.
	C	Highly vulnerable bonds. Issuer may be in default.
	CI	Issuer is past due on interest
	R	Under regulatory supervision due to its financial situation
	SD	Selectively defaulted on some obligations
	D	Issuer has defaulted on obligations and S&P believes that it will generally default on most or all obligations
	NR	Not rated

Table 2.1: S&P Ratings
Source: www.bondsonline.com

Moody's

The Moody's ratings follow a similar structure as the S&P ratings, but the symbols differ slightly. The ratings assigned by Moody's range from Aaa to C, with Aaa denoting the highest quality issuers and bonds. For bonds rated between Aa and Caa, there are intermediate modifiers (1, 2 and 3, with 1 being the best) to give more information regarding the creditworthiness of the issuer.

Rating		Description
Investment Grade	Aaa	Highest quality with minimal credit risk
	Aa	High quality. Subject to very low credit risk
	A	Upper-medium grade. Subject to low credit risk.
	Baa	Medium grade: may possess certain speculative characteristics. Moderate credit risk.
High Yield	Ba	Possess speculative elements. Subject to substantial credit risk
	B	Considered speculative. Subject to high credit risk.
	Caa	Considered of poor standing. Subject to very high credit risk
	Ca	Highly speculative. Likely in or near default with some prospect of recovery of principal or interest
	C	Lowest rated class of bonds. Typically in default with little prospect for recovery of principal or interest.
	WR	Withdrawn Rating
	NR	Not rated
	P	Provisional

Table 2.2: Moody's Ratings
Source: www.wikipedia.com

Fitch Ratings

The Fitch rating system is very similar to Standard & Poor's. These ratings also use plus and minus symbols to denote intermediate ratings for each category between AA and CCC.

Rating		Description
Investment Grade	AAA	The best quality companies. Considered reliable and stable.
	AA	Quality companies. Slightly higher risk than AAA bonds
	A	Economic Situation can affect finance
	BBB	Medium class companies, which are satisfactory at the moment
High Yield	BB	More prone to changes in the economy
	B	Financial situation varies noticeably
	CCC	Currently Vulnerable and dependent on favorable economic conditions to meet its commitments
	CC	Highly vulnerable. Very speculative bonds.
	C	Highly vulnerable. Perhaps in bankruptcy or in arrears but still continuing to pay out on obligations
	D	Issuer has defaulted on obligations and Fitch believes that it will generally default on most or all obligations
	NR	Not publicly rated

Table 2.3: Fitch Ratings
Source: www.wikipedia.com

2.3 Financial Regulatory Agencies

Regulatory agencies play a critical role in the continued livelihood of all of the world's financial markets. The United States Treasury lists five of the largest of these agencies on its website, the most relevant of which being the *Securities and Exchange Commission* (SEC). The primary purpose of regulatory agencies is to ensure ethical behavior in all financial transactions through establishing and enforcing trading rules. The SEC endeavors to oversee all activity in the financial markets of the United States. Through this, financial markets in the United States are regulated to ensure the safest and most effective environment for investors.

2.3.1 Securities and Exchange Commission

A product of the stock market crash of 1929, the SEC was established in 1934 to "to enforce the newly-passed securities laws, to promote stability in the markets and, most importantly, to protect investors" [SEC 2006]. The laws established in the Securities Exchange Act of 1934 were designed to restore investor confidence in the market through disclosure of important information and honesty of those involved in securities transactions. The SEC works closely with the United States Treasury, self-regulatory agencies, state securities regulators and other private organizations such as the *National Association of Securities Dealers* (NASD).

2.3.2 National Association of Securities Dealers

Since its inception in 1939, the NASD has been providing investors with confidence and the markets with integrity. Through regulatory systems, this organization has monitored and policed the securities industry as to provide the most benefit to investors. Their motto, "*Investor Protection. Market Integrity*" is upheld through "examination, rule writing, professional training, licensing and registration, dispute resolution, and investor education" [Shulman].

2.3.3 Trade Reporting and Compliance Engine

As recently as 2001, there was almost no transparency in the corporate debt market and trade decisions were rather intuitive, depending much on the trader's own perception of the market. However, within the past five years the NASD has helped shape a significant change. Through the development of the *Trade Reporting and Compliance Engine* (TRACE), data regarding approximately 22,000 transactions involving $18 billion par value daily is reported and disseminated every day according to NASD. There are many types of market data reported to TRACE, most of which is immediately disclosed to investors. This data provides investors information about market activity, overall pricing and execution quality, and enhances the

integrity and transparency of the market. Because of the type of data it disseminates, TRACE also has the potential to be a powerful tool in analyzing the corporate debt market.

The concept of TRACE arose in 1998 when the Securities and Exchange Commission (SEC) was placing an emphasis on price transparency and requested NASD to take three steps in order to improve the corporate debt market. SEC wanted a system that would report all corporate bond transactions to NASD and then distribute the prices of those transactions immediately. Also, a compliance surveillance program was requested along with a database for these transactions in order to supervise the market. From these specifications NASD developed TRACE and implemented a new set of trading rules on July 1, 2002. While only about 500 bonds were included in the program at first, more have been added to TRACE over time. This has resulted in an increasing transparency in the corporate debt market.

All NASD members are required to report their trade transactions to TRACE. Any corporate bond traded on the secondary market is a considered a TRACE-eligible security, excluding those that are publicly disclosed on other national securities exchanges. One other exception of note is corporate debt transactions where the buyer and the seller have agreed to trade at a price substantially unrelated to the current market for the TRACE-eligible security. This seems somewhat illogical, as these oddly priced bonds are exactly the type of data that TRACE should record and disseminate.

Upon its launch in 2002, TRACE publicly disseminated all transaction data in investment grade bonds greater than $1 billion in original issuance and 50 representative high yield bonds. This represented only 31% of all transactions and 38% of investment grade trades [Shulman]. In April 2003, TRACE expanded its range of publicly disseminated data. This expansion included investment grade bonds that were rated A or better by a nationally recognized statistical rating

organization and at least $100 million in original issuance, as well as data on 120 representative BBB rated bonds and 50 high-yield bonds. At this point, it covered 43% of all transactions and 61% of investment grade trades. Beginning 2004, all transactions reported to TRACE were publicly disseminated. In this case, 99% of all corporate bonds were available immediately and the remaining 1% was delayed[1]. As of January 2006, 100 percent of public corporate bond transactions data is being disseminated in real time through the *Bond Trade Dissemination Service* (BTDS). This service broadcasts last sale price and other relevant trade data to authorized vendors.

TRACE reports exact trade volumes for all investment grade trades of $5 million or less and all high yield trades of $1 million or less. For trades of these types larger than their respective limits, values of '$5MM+' and '$1MM+' are displayed in TRACE. These are referred to as capped trades. This is done so that a particular trader's intentions are not obvious to other investors.

Every time a transaction is executed, the reporting members have to submit a trade report containing information about the trade. This report must include the CUSIP number or NASD symbol, the number of bonds traded, the price of the entire transaction, whether the transaction is a buy or a sell, the date and time of the trade execution, and other descriptive information[2].

Before the existence of such publicly available data, decisions on the bond market were rather opportunistic and were based more on the traders' intuition than on factual knowledge. With the introduction of TRACE, all the essential information regarding bond trades was available, thus giving traders and investors all the knowledge they needed in order to make a

[1] This 1% included certain transactions in lower rated securities executed during a short period after issuance and infrequently traded non-investment grade securities

[2] Other information reported to TRACE: the contra-party identifier, the capacity (Principal or Agent), stated commission, the lower of yield to call or yield to maturity, and, if applicable, the reporting side executing broker as "give-up" and contra-side introducing broker.

decision. Therefore, TRACE is a very good source of information for anyone who wants to know what the state of the corporate bond market is and also wants to use this knowledge to make an intelligent investing decision.

3. Methodology

For this project, we used multiple methods to better understand how to most accurately and efficiently produce a useful application for corporate bond traders. We used mostly TRACE data in our project, so we began by researching TRACE and the data it collects and disseminates. We then gained a broad understanding of the implications of the TRACE data through meetings with the project stakeholder, Martin Gonzalez (Principal Trader). With a good concept of the general purpose of the project, we met with two staff members (Jason Wang and Igor Zitser) who were familiar with previously developed applications and databases that could be of use to us. After gaining access to these databases, we explored them in great detail to discover which data was useful and which was not.

3.1 Research

We used two approaches to obtaining information: literary research and interviewing. By studying articles on NASD's website, we were able to comprehend the purpose, structure and functionality of the TRACE system. Also, by meeting with knowledgeable Bank of America employees, we gained a better understanding of the uses of TRACE data and technological resources available to us.

3.1.1 Literature Study

To successfully build a useful application, we needed to have a very firm grasp on the principles of the corporate bond market as well as the specific data collected and disseminated by TRACE. We read multiple documents issued by the NASD regarding corporate bonds, enabling us to begin our research on TRACE. As a secondary part of our research, we studied how the TRACE data was being reported to individuals within BAS. We examined a proprietary application of Bank of America, called *RealTic*, which captures data from the *Bond Trade*

Dissemination Service (BTDS) and reports it in a table that can be sorted by any of the thirty columns.

3.1.2 Meetings

After researching, we began to meet with several parties that would impact our project in varying ways. First, we met with Martin Gonzalez (Principal Trader), who is the primary stakeholder, and our support team to determine what was expected of this project. Next, we had meetings with several BAS employees, including Alex Gregory (VP) and Ellen Tsai (Associate VP), who could act as resources for the technology with which we would be dealing. After this step, we had all the necessary information to begin doing our analysis and draft requirements for the project.

3.2 Analysis

At first, we received a basic set of requirements from the primary stakeholder, who also mentioned that there would be future additions to them. These initial requirements set the foundation on which we could create a project plan that was adaptable to further changes.

3.2.1 Requirements

Initially, Martin asked for an output in the form of a table with certain data on each bond traded during the previous day (the Daily Trade Summary). He provided us with a mockup of what the report would best look like to him, in the form of a spreadsheet document. From this, we learned what particular aspects of the TRACE data would be most relevant to him and consequently to our project. We also had several discussions with him on clarifying the format and contents of the report. Through these meetings we were able to change the original layout of the report, with several fields added and some aesthetic changes made. Discussing such changes

before the actual creation of the software was beneficial to our development process in the sense that we were able to avoid making many costly modifications (such as redesign and recoding).

After the initial set of requirements was implemented, Martin came back to us with another set of specifications relating to two additional reports. We pursued the same process as for the first report; we discussed the requirements with the stakeholder and came up with final mockups, and only after that did we begin developing them.

3.2.2 Data Mining

The next, and most time-consuming, phase of our project required us to explore multiple databases in search of applicable data and ways it could be applied to our application. This represented the data mining process. We began by looking at a *RealTic* database and searched through multiple tables for pertinent data. Because *RealTic* only contains trade history information, we needed another source for static bond data. After further discussions with our support team (Kurt Vile, Principal), we learned of another source of information, named *ALICE*, that provided the static data for which we were looking. We then began to mine data from *ALICE*. We worked on accessing it through a human-readable web interface and then looked for ways to retrieve this data. Through these two sources, we were able to find all the information we needed to produce the reports.

3.3 Architecture and Design

Based on the requirements and on the results of the data mining process, there were two approaches to the design of this system. One would be a client-server architecture with a “core” application running on a fast and reliable 24-hour server and a client application that would be used by the trader whenever he/she wants to change the running parameters of the system. The advantages of this approach are that the application would be available day and night, and it

would not affect the trader's machine with respect to performance. The main disadvantage is that it would be more difficult to design, test and especially maintain the entire system after its release because there would be two applications involved.

The other approach, that of a single-tier architecture, would imply having only one application that would run on the trader's machine at a time he/she would specify. This application would have different subsystems that accomplish different tasks, as well as the means for the user to access a Graphical User Interface (GUI) in order to configure it. Different than the client-server approach, this one has the advantage of being in one piece, and is thus easier to implement and maintain. The drawback, however, is that support would need to be added in order for it to successfully run on the client's computer. That would include, but not be limited to, the Java Virtual Machine and other libraries required for the different tasks it accomplishes.

After discussing the options with the stakeholders and our support team, we decided to proceed with the single-tier architecture and deploy any additional software to the client's machine, if required. In this case, we would not only make our application more secure (since the data the trader inputs into it is confidential, and any outside disclosure of it would be considered illegal), but also easier to use and access. Given the above constraints, we decided to structure our application into eight different packages, each performing a different function.

3.4 Development

The actual development (or coding) part of the project could not be started until we had a good idea of how the program should run based on user preferences and how it was supposed to respond to external factors. We knew from the requirements that this program should have two main flows of events: one that allows the user to configure his/her preferences (named "Configuration Mode") and another one that generates the actual reports and sends them to the

designated recipients (“Execution Mode”). Therefore, we drafted a sequence diagram for each flow in part [Figures B.3 and B.4] and then created a state diagram that would fully describe the behavior of our application [Figure B.5]. We then discussed this flow of events with our stakeholder Martin, as well as with our support team, and agreed that it was viable and would produce the desired results.

3.5 Testing

Every software application has to undergo a comprehensive series of tests before it can be released. These include unit tests, functional tests and user acceptance tests. The unit tests refer to the internal functioning of the software and should be performed by the developer to ensure that all the pieces of software work individually. Functional testing ensures that the program behaves as it is supposed to, and that it meets all the requirements. Such testing is usually performed by a quality assurance engineer, but in our case, we both participated in making sure our application met the requirements. Finally, we released the application to the primary user and received valuable feedback from his experience with it. This allowed for further bug-fixing and improvements (as the stakeholder’s opinion is the most important one).

3.6 Release

After our application passed all tests and we received approval from the stakeholders, *TraceMon* was ready for its final release. This step involved packaging the software into a single JAR (Java Executable Archive) file which could be deployed on any machine that had Java JRE 1.5 installed. As it is designed, it can be run either on a local machine or on a remote server.

Other steps needed at this moment were to create a user manual (to familiarize new users to the program), developer information (to educate other software developers on how to modify

our program after our departure), and a short troubleshooting guide (to get users moving in case something unexpected happens).

4. Results

The main objective of this project was to create an application to analyze TRACE data. Through literary research and meetings, we compiled a list of requirements. Based on those requirements, we began looking for sources of data and efficient ways of extracting it. At the same time, we began the conceptualization and development of our application. After discussing our findings with the project stakeholders and implementing suggestions from their feedback, we managed to create a useful program that saves traders a significant amount of time and presents important information in an organized and easy-to-read format.

4.1 Information Gathering

4.1.1 Stakeholder Meetings

We first met with our direct supervisor, Kurt Vile, and the debt trader for whom we were developing the application, Martin Gonzalez. At this meeting, Martin explained what makes TRACE data appealing to him and how he intends to use it. We then discussed our options for the basic structure of the application. Our second meeting with Martin also included another of our sponsor liaisons, Alex Gregory. It was at this meeting that we gained further knowledge of the output type Martin was looking for as well as the feasible options for meeting his requirements. From these meetings we were able to learn the basic objectives and requirements for the application we were to develop.

4.1.2 Informational Meetings

Throughout the course of the project, we met with five BAS employees who were able to provide us with important information regarding the applications and databases with which we were working. Through meeting with Jason Wang and Igor Zitser, we learned about the basic capabilities of the *RealTic* program, and following our own exploration of it we were able to

document its capabilities. To obtain bond information not contained in TRACE, we accessed another database system called *ALICE*. We contacted Charles Waddington in the Chicago office to discuss the use of *ALICE* and how to best access its resources. Additionally, we spoke with Jian Huang to determine the best method for sending our application's output to Martin. Once we had determined that emailing the results would be best, we met with Davran Muzafarov to learn how to be able to send emails from our program using existing Bank of America systems.

4.2 Requirements

Both at the beginning and as the project progressed, we received many requirements of the program set forth mainly by Martin. The first report, the *Daily Trade Summary*, was requested to give traders a daily overview of activity for specific bond tickers. By summarizing the daily data, this report allows the user to see traded volumes and other analytical measures (such as prices, spreads and investment ratings). The initial report mockup required the following information to be shown:

- Ticker symbol
- Coupon rate
- Maturity date
- CUSIP
- Outstanding notional (in millions)
- Traded notional for the previous day (in millions)
- Number of trades greater than $1 million par value
- Number of trades greater than $5 million par value
- High, low and average spread to U.S. Treasury benchmark
- Description of U.S. Treasury benchmark
- Bond Ratings, gathered from Moody's, S&P and Fitch

After presenting a sample report based on these initial requirements, Martin requested the following statistics to be included:

- Optionality (callable, non-callable and put-option bonds)
- High price
- Low price
- Average price

Additional requirements included a label of ‘floating’ in the coupon field and the removal of spreads for floating rate bonds, better descriptions of the U.S. Treasury benchmark (such as coupon and maturity), the floating rate bonds, and to list the bonds in order of maturity starting with those maturing soonest. Furthermore, we were asked to use an estimator for investment grade bonds traded over $5 million and one for high yield bonds traded over $1 million. This was done to provide a more accurate traded notional value, as trades over those values are capped in the TRACE data.

On top of these data requirements, we received format and user interface (UI) specifications as well. Martin requested to be able to receive this information through email, mostly so it could be read while he is away from the office. Also, a spreadsheet output (in Comma-Separated Values, or CSV, format) was requested as it is a preferable form in which to have data. This last concept never materialized because the email format turned out to be very helpful and easy to import into a spreadsheet program, such as Microsoft Excel, without too much action required from the user to format the table columns.

The first report was the longest and most difficult to design and implement since we had to do research and develop a great part of our application (as described in the following sections). After the first report was done, the other two were relatively simple to design and implement, and we only had to deal with performance and other, minor, issues.

The second report Martin asked us to develop is a *Liquidity Report*. This report provides the user with historical traded volumes and the corresponding percentage of the total amount outstanding (or outstanding notional). From this information, a trader is able to easily see the liquidity of each bond, and approximately how long it would take to invest a specific amount of money in a particular bond.

Similarly to the first report, Martin wanted to have a set of tickers for which some statistics could be calculated. In contrast to the first one, he wanted this report to run on request rather than daily. The fields that this report needed to contain were:

- Ticker symbol
- Coupon Rate
- Maturity Date
- Outstanding Notional
- Aggregate amounts of traded activity for each of the past 12 months
- The Average Monthly Traded Notional (the average aggregate amount traded per month)
- Calculations that would represent the percentage these numbers are from the Outstanding Notional

The third report we were asked to implement was a Trade Outliers Report. This report calculates the amount traded per bond for the last trading day, as well as the average amount traded in the last five trading days, or in the last one, three, six or twelve months. This enables the user to see if a bond is trading at volumes significantly higher or lower than historic averages. The table header for this report would contain:

- Ticker
- Coupon Rate
- Maturity Date
- Recent Traded Volume (Last Day and Last 5 Days)
- Average Traded Volume (Last 1, 3, 6 and 12 Months)

For the first and third report, we were asked to filter out all the trades that had an amount less than one million dollars and only include those above. The reasoning for this is that trades with volumes below one million dollars are not significant when trading on the investment banking level. These requirements gave us the foundation to begin searching for the needed data and developing an application that would meet them.

4.3 Data Mining

After receiving the abovementioned requirements, we began searching for reliable sources of information that our application could use in order to generate the requested reports. We began by examining *RealTic* and its databases for bond trade history information. Of the data requirements set forth by Martin, we were able to find the following:

- Coupon rate
- CUSIP
- Execution Date
- Maturity Date
- Number of trades greater than $1 million par value
- Number of trades greater than $5 million par value
- Price
- Ticker symbol
- Traded notional
- Yield

From this we were able to search by CUSIP in *ALICE* for the following static bond information:

- Optionality
- Outstanding notional
- Ratings
  - Fitch
  - Moody's
  - S&P
- Whether the bond has a floating coupon rate or not

This left only the treasury spread fields to be calculated. We were able to find a table in the *RealTic* database which provided real-time Treasury yields. By applying a simple metric[3] to determine which Treasury security to use, we could calculate fairly accurate Treasury spreads.

[3]

Time to Maturity	*Benchmark*
0 to 2.5 Years	2 Year Treasury Note
2.5 to 3.75 Years	3 Year Treasury Note
3.75 to 6.5 Years	5 Year Treasury Bond
6.5 to 15 Years	10 Year Treasury Bond
Greater than 15 Years	30 Year Treasury Bond

We were also able to get the pricing data we needed from this *RealTic* database. However, *ALICE* was the only database from which the call and put data could be attained. This left us dependent on *ALICE* only for the amount outstanding, bond ratings and call/put features.

The aforementioned process applied to the *Daily Trade Summar*y report, which needed the most diverse sources of information. The *Liquidity* and *Trade Outliers* reports did not require us to mine for any additional data, but we needed to search data archives (since they required trade history up to a year), which posed new problems. These archives exist to improve *RealTic* database performance. All trades older than three months are moved to the archive database, which keeps information up to three years. Given the huge size of this data source, querying it took longer, but the most challenging part was to combine the information from both the archive and current databases in order to come up with a coherent report.

Consequently, we had two significant data sources for our project. The main one is the *RealTic* database, which contains information about all the daily bond trades and is updated with new information in real-time. The other one is the *ALICE* database and it contains static information about bonds, which does not change with every trade. Instead, this database is updated upon request, retrieving the new information from a reliable outside source, such as Bloomberg.

At this point we had all the information we needed to begin the design phase of our project.

4.4 High Level Architecture

After drafting and understanding the requirements for the project, as well as determining the data sources our application would use, we were ready to conceptualize and design a high-level architecture for our project. We decided to use Java JRE 1.5 as our development platform

with Eclipse 3.2 as a development environment. Java is highly compatible with the other systems this application would interact with, and it is a language that facilitates the Object-Oriented paradigm (essential for a successful implementation of a complex software system).

We determined that our application would be organized into eight packages, each responsible with a different task, as described below. A high-level view of our system can be seen in Figure B.1, and a more detailed view can be seen in Figure B.2.

The **com.bankofamerica.alice** package is the only link to the *ALICE* database. It uses a Simple Object Access Protocol (SOAP) framework to connect to *ALICE* via web requests. This package is composed of two sub-packages (*asset* and *service.asset*), and provides a high-level access to the remote system, without having to deal with protocols, database schema changes or other issues. This package was provided to us by the *ALICE* development team and requires external libraries to be linked to the project that will allow our application to connect to the remote system. However, we encountered some problems while importing bonds from Bloomberg into *ALICE* [Meeting Minutes, Appendix A]. It appears that while we could use the production server to retrieve existing bond information, we could not use it to import new bonds from Bloomberg. We could, however, use one of the two Quality Assurance servers (QA and QA2) to perform that action. In order to do this, we needed to add two new almost-identical sub-packages that would provide a similar interface to the QA2 server, through which we could safely import new bonds from Bloomberg. These two packages (*alice.qa2.asset* and *alice.qa2.service.asset*) were also provided to us by the *ALICE* development team and even though from a design point-of-view this introduced some redundancy in the architecture, this action was necessary since the production and QA2 servers were not fully compatible and required to have separate packages for each of them. These two packages (each having two sub-

packages) provide a façade to the ALICE servers and are independent from each other (so accessing one will not affect the other one). In order to facilitate their use, an adapter was developed by us that shielded the program from having to know which server to use or whether to import bonds or not, as described later.

The main package, which is also the single entry point into our application, is the **com.bankofamerica.tracemon** package. It contains the *Main* Class, which is responsible with initializing the application, reading the command-line arguments and configuring it accordingly. It is also where it is decided whether to display the configuration GUI or start processing reports.

The **com.bankofamerica.tracemon.adapter** package provides high-level adapters that allow the application to communicate with outside sources of information. The *RealTicDB* class provides an interface to send queries and retrieve data from the *RealTic* database without the hassle of having to know connection strings, opening/closing connections, etc. The *AliceAdapter* class provides a simplified interface to the com.bankofamerica.alice package that, in turn, provides an interface to the remote *ALICE* system. This adapter also shields the rest of the application from the burden of deciding which *ALICE* server to use, as well as dealing with bonds that needed to be imported from Bloomberg, using a predefined algorithm[4] that was discussed with the *ALICE* development team. The *TreasuryBondAdapter* class is responsible for retrieving pertinent Treasury security information from other Bank of America systems and providing them to the application. Another adapter that was deemed necessary was the *WebRequest* class, which allows the application to easily send a request over the internet using HTTP (Hyper Text Transfer Protocol).

[4] The algorithm for retrieving a bond from *ALICE* is:

1. Look for it in the Production *ALICE* server, and if found there, return it
2. If not found, look for it in the *ALICE* QA2 server, and if found there, return it
3. If not found in either locations, import the bond from Bloomberg into *ALICE* QA2 and return it.

The package **com.bankofamerica.tracemon.asset** contains entity classes, mainly responsible for storing information and delivering it in a meaningful form. This package contains the *Bond* class, which, given a bond unique identifier[5], retrieves all the necessary information about it that will be used to generate further reports. Also, the *TreasuryBond* class holds information about Treasury Bonds as extracted by the *TreasuryBondAdapter*.

The **com.bankofamerica.tracemon.config** package is responsible for the application configuration; from reading/writing it to an external file to providing a GUI to the user in order to change it. It contains several classes, of which *Config* is responsible for reading and writing the user preferences from/to an external file, *Settings* keeps hard-coded general parameters for the application, and *MailAccount* and *ReportInfo* are used as entity classes in order to store different information about the application configuration. The sub-package **config.ui** contains the GUI through which the user can change his/her preferences on reports.

All the reporting-related logic goes into the **com.bankofamerica.tracemon.reporting** package. This contains an abstract class, called *Report*, which provides a general structure on how a particular report should behave. Thus, it provides abstract methods for retrieving and generating data, as well as outputting the results both in HTML and Text-only format. All reports in the program are derived from this class.

All the miscellaneous logic, that does not have a clear package designation, and which is intended for a more general use, is part of the **com.bankofamerica.tracemon.util** package. This package contains the following classes: *HTML*, *Logger*, *Mailer*, *StopWatch* and *Tools*. The *HTML* class provides useful methods to generate HTML documents; the *Logger* class provides methods to log the actions of the program for further study and debugging purposes; *Mailer* is used to send emails to different Bank of America email addresses; *StopWatch* is a simple class

[5] Also known as CUSIP (Committee of Uniform Securities Identification Procedures) identifier

that allows for timing between two different places in the code, and *Tools* has different methods that perform miscellaneous tasks.

As the project grew larger and larger, identifying and fixing problems became more and more tedious. This was also a direct result of the fact that all the information was retrieved in parallel for multiple bonds at the same time (a technique known as Multi-Threading). Consequently, a debugging strategy had to be developed in order to save us numerous hours of unprofitable problem-fixing work. Therefore, we created a unique log file that keeps track of all the actions a specific instance of the program performs. This log file can later be interpreted through the use of a new subsystem that can be found in the **com.bankofamerica.tracemon.-eventlogger**. This system provides a simple user interface that shows all the logs from all the runs of the program, and for each of them, an intuitive grouping of the events for easy access. Information such as errors, warnings, or simple events is recorded in this file with the relevant time stamps, which can make debugging and performance tuning much easier.

4.5 Development

4.5.1 Sketching the flow of events

The *Configuration Mode* is the only flow in which the user can interact with the application (except when he/she starts the program). Upon startup, the application loads the user's preferences (if any exist) and then shows the Configuration Window. If this is the first time the user ran the program, then an empty profile is created for him/her, which then has to be edited. From this window, the user is able to change the preferences for each report (including tickers, report name, whether the report should be run) and is also able to import preferences from other reports. Additionally, the Configuration Window allows the user to change general information not pertinent to any type of report, and gives him/her the option of saving his/her

profile or exiting without saving. A use-case diagram of the user's options can be seen in Figure B.5. After the Configuration Window is closed, the application will save the configuration, if necessary, and then exit. See Figure B.4 for a complete sequence of events.

The *Execution Mode* is an automated process. It was designed to run in the background without any user input in order to facilitate batch runs (most likely at nights, when the computer is not used). Similarly to the Configuration Mode, it loads the user profile (and any other relevant information from the command-line arguments) and then other information that may be needed, such as Treasury Bond Information and Holiday Schedule (so that it knows when the bond market is closed). After that, it determines which reports to run (if the user overrode the preferences in his/her profile using command-line arguments, then the given reports will be run; otherwise the reports selected in the profile will be generated) and starts generating reports. Each report is responsible for keeping track of its own list of bonds. It searches for the necessary bonds in the *RealTic* database and then, for each of those bonds, retrieves the static bond information (from *ALICE*) and the trade history (from *RealTic*) and calculates all the statistics that are necessary to generate the report. As soon as all the data is in place, the final report is generated and sent by email to the designated recipients in the user's profile. The sequence diagram for this flow can be seen in Figure B.3.

After the sequence diagrams for both flows were created, we unified them into a single state machine diagram that describes all the states in which our application can be, as well as the flow of events to and from a particular state [Figure B.6].

4.5.2 Actual Development

We used an iterative approach to designing and implementing our system. Based on the architecture outlined above and the described sequence diagrams, we broke the development phase into several stages, where during each step we added more functionality to the system. After each step was complete, we tested the newly added functionality using unit tests and functional tests, if necessary, and also tested the existing functionality. For more details regarding testing, see Section 4.6.

- **Initial Layout of the System.** The first step in the development stage was to create the packages and class stubs, according to the general architecture of the system. These classes would then be populated with member variables and functions that would accomplish the tasks for which they were designed. After this step, our system was in the form of an *executable framework*; that is, it could run, but it would not carry out any processing.
- **Adapters.** The second step in our design involved implementing the data adapters that would provide a high-level interface to the various data systems at Bank of America. The first one was the *RealTicDB* Adapter, which encapsulated methods and information to connect to the *RealTic* database. This class used the *Sybase 6.0 Database Driver* that connected to the external source of data. The second adapter which was implemented was for the *ALICE* data source. This class used the underlying *ALICE* subsystem, which was provided to us by the *ALICE* Development Team, and allowed the application to easily access it, using only one function call. The introduction of this adapter proved to be of utmost importance, since it saved us a lot of tedious work (only had to change one class instead of several) when we had to resolve the production issue we came across during our development process [Meeting Minutes, Appendix A]. Also, the need for a Web information retrieval led to the development

of the *WebRequest* class, which facilitates sending and retrieving information from remote websites by means of HTTP requests.

- **Data Retrieval.** The third step involved data retrieval. All the data that our application needs is retrieved through the aforementioned adapters. As per our design, the only class that needs to retrieve external data is the *Bond* class. Upon instantiation, it automatically retrieves all the relevant information for the bond it represents, including the static bond information from *ALICE* (through the appropriate adapter), and, on a need basis, the rest of the information (i.e., if the high price for a day was needed, then it would request and calculate all the necessary price information, but not other data – which will be retrieved when needed). Since a request for retrieval from *ALICE* takes a long time on the average (around 3 seconds), this would significantly impact the overall performance of the system. Thus, when each bond object is created, a separate thread is also instantiated that retrieves the desired information from the *ALICE* data source. For synchronization with the main thread, a method in the bond class has been made available that would return true only after all the data is retrieved, which allows the main thread to know when this Bond object is done retrieving data. After several performance tests, we decided to also perform major calculations as part of our threads (depending on where a Bond object would be used, it would automatically calculate either Recent Average Trades, Monthly Traded Amounts or the Spreads and Prices upon instantiation). If more data was required, it would be retrieved on a need basis, as described above. This multi-threaded approach resulted in an 84% decrease in the overall running time for our application[6].

[6] We measured this for three runs (one containing 10 bonds, one with 40 bonds and one with 200 bonds) and determined that, for a smaller set of bonds, it does not yield a very substantial change (only 40% less), but for a larger set it can significantly improve running time (we obtained an 84% decrease for the 200 bond set)

- **Data Processing.** The next logical step in our design included the processing of the retrieved data. As stated above, besides being responsible with obtaining appropriate data, the Bond class is the single point in the application where it is being processed, This is also the place where all the spreads and statistics are calculated and where all the logic is being performed. As explained above, upon instantiation, a Bond object automatically retrieves all the static information about it from *ALICE*, as well as relevant information regarding the report where it would be used. It contains logic that computes all the necessary statistics mentioned in the user requirements. Due to the high amount of data that needs to be processed, complex SQL[7] queries had to be developed in order to filter out irrelevant data and perform most calculations on the remote database server. This also resulted in better performance for our system; in general, the closer computations are made to the data source, the better the performance of the system is.
- **Report Generation.** After the data retrieval and processing systems were in place, the next step was to present it in a human-readable form, which was the responsibility of the reporting system. The Reporting package is responsible for keeping all the reports. As stated in the architecture section of this chapter, there is an abstract class that provides the skeleton of any report (the Report Class). In general, a report object (instance of the Report Class) should have a section that deals with data retrieval, another one to process it and another one to generate an actual report. In our case, since we need roughly the same data between reports, we moved all the data processing part inside the Bond class, thus eliminating the need for a similar step while generating a report. In conclusion, our report would only contain a step that retrieves the data and another one to report it. Each report is supposed to provide both an

[7] Structured Query Language. Standard programming language used to extract and modify information in database servers

HTML and a text-only representation of itself, although only the HTML representation is currently used.

- **Emailing system.** The last thing that had to be implemented before we met the user requirements was a way to have all the generated reports be sent by email to the desired recipients. In this case, we met with a developer of another application that had already implemented such a system, and used their solution (after adapting it to our needs). The resulting solution was implemented into the *Mailer* class, which allowed our application to easily send email using only one function call.
- **Configuration Management.** Finally, after we made sure that all of the above worked, it was time to give the user an easy way to set up his/her preferences, including the type of information he wanted reports on, the actual reports that would run as well as the email addresses of all the recipients of those reports.

At that point, we had developed our first report generating mechanism. The following two reports did not require re-architecting or other major changes in the system, since they needed roughly the same type of information as the first report. The only things we needed to do in order to make another report were to create a separate class that would represent it, as well as add the necessary logic to calculate the fields in it. The class that represents the new report was derived from the Report Class (thus inheriting the basic functionality and overriding abstract methods) and the logic to compute new statistics was added to the Bond class. No further data mining or research had to be done. The object-oriented design of our system allowed for easy modification to the current configuration, thus adding new functionality would not require existing features to change.

The development steps we took *after* the first report was finished were:

- **Develop Second Report (Liquidity)**. This report shows trading activity for the past twelve months, grouped by month. Its development involved creating a complex SQL query that is sent to the database system in order for it to make the required computations and deliver the data to us in a form that does not require too much processing. The delegation of such computations to the database server was essential, since we could be retrieving data simultaneously for hundreds or thousands of bonds, thus the local machine's CPU can easily become overwhelmed with computations – which can result in a poorer performance. After the results are retrieved from the database server, a simple routine is performed to interpret them and then present them to the user, therefore significantly reducing the computing power needed on the local machine.
- **Develop Third Report (Trade Outliers)**. This report shows averaged trading activity for the past one or five days, as well as for the past one, three, six and twelve months. The development of this report was very similar to the Liquidity one, having a complex SQL query that delegates the data processing to the database server and then only interprets it using a simple, low-cost routine. Also, since this report needs to know the holidays (in which no trading activity occurred) so that it will accurately report the average amount traded for the past five days, a special feature had to be implemented that would determine whether a given day is a holiday. An external XML file (accessed by the program on every run) was created that keeps all holidays, except Saturdays and Sundays, until 12/31/2007. After that date, someone in the bank will have to update that file each year with new holidays in order for the program to continue generating reports accurately.

4.6 Testing

The next phase of our project required significant tests to ensure there were no bugs in *TraceMon* and that it was reporting and calculating information accurately. To do this, we first checked the data being calculated by hand. We then submitted the application to its primary end-user, Martin, for his acceptance and approval. Through this, we were able to refine our application into its most functional and accurate form.

4.6.1 Quality Assurance

This section of testing was done predominantly by us. We thoroughly examined the data being reported by *TraceMon* to ensure it was correct. We used spreadsheet programs, such as Microsoft Excel to calculate data 'by hand' and validate our program output against it. Although most of the data was accurate, we were able to fix many not-so-obvious bugs that caused the program to display the wrong output.

After we decided that the numbers were accurate, we ran several performance tests, in which we measured the memory usage, CPU utilization and the overall time it took to run it. To do this, we selected the most traded bonds of some of the largest bond issuers (using official statistics which were publicly available) and input their identifiers into our program. We discovered that, even though the multi-threading approach significantly reduced the running time, it introduced other problems, such as over-threading (having too many threads running in parallel which results in too much time spent by the Java Virtual Machine to switch between them), out-of-memory issues or request denials from remote servers (we identified the cause to be too many simultaneous requests, which would overwhelm the remote server, thus causing it to refuse any other connections until it has less demand for bonds).

We concluded that the common cause of the above problems is the fact that we initially had too many simultaneous requests for information. We decided to spread them out (at the cost of overall running time), and introduced a lag between each ticker (group of bonds with the same issuers), as well as a lag between each bond in a ticker. This fix significantly reduced the number of denied requests, as well as all of the out-of-memory errors, and reduced the CPU utilization to an acceptable level (this level varies with the machine configurations – can be lower on better machines, or higher on less powerful computers). As for the remaining problem – that of having denied requests – we decided that the best solution was to retry the connection up to four times, with a lag of 30 seconds up to 2 minutes (in 30 second increments). Further testing proved that this approach fixed the problem.

Additional tests were run for performance and we were able to tweak the application to such an extent that, on the average, it uses 40% of the CPU, with 90MB of memory and it takes 2.2 seconds to load a bond. One test that employed around 2200 bonds took 70 minutes to complete (however it could take several more hours if it has to import a lot of information from Bloomberg into *ALICE*).

4.6.2 User Acceptance Testing

After the unit and functional testing was complete, we were ready to launch a preliminary version to the main user. We could therefore get valuable feedback from him, as well as suggestions for improvement and possible undetected problems. We went to Martin's desk and showed him how to configure and launch the application. We experienced some minor difficulties as outlined in the Meeting Minutes in Appendix A, but managed to overcome them in a fairly short amount of time and were able to prevent them from happening again. During the following week, we were in close contact with Martin and further adjusted our program based on

his experience with it. Having completed the user acceptance testing stage, we were ready for the final release of our application.

4.7 Final Product – *TraceMon*

After the development and testing phases were complete, we presented our application to the stakeholders (Martin Gonzalez and Kurt Vile) and got their approval for release. We packaged all the compiled classes into a single JAR (Java Executable Archive) file and created the necessary shell scripts in order to start the program (one for the configuration mode, one for the execution mode and one for the event logger). After that, we moved it to a public folder on a shared drive (so that it can be accessed by anyone who needs it) and ran the last series of tests in order to make sure that that moving it to the server did not introduce unexpected problems.

4.7.1 Configuration Mode

The configuration mode can be accessed by running the shell script "*config.bat*". This will open up a window that allows the user to view and edit his/her profile with respect to our application, *TraceMon*.

In the "TraceMon Configuration" window, the user is shown four tabs, three of them referring to each report this program supports and the fourth one relating to general information, which is not pertinent to any report in particular. All report tabs are structurally identical in order to make configuration easier. Therefore, for each report, the user can perform the following operations [Figure 4.1]:

- Change the name of the report as it will be identified in the email
- Add/remove/modify the tickers for which he/she wants the report to run
- Copy tickers from one report to another
- Enable/disable a report from being generated on the next scheduled run

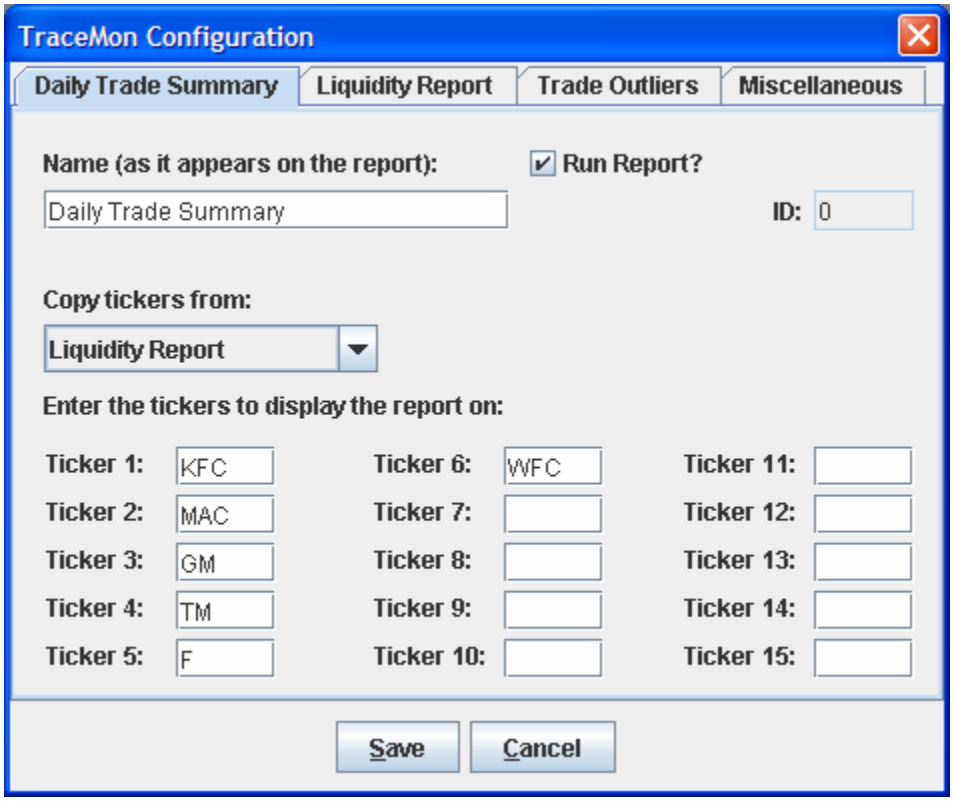


Figure 4.1: Configuration Mode Screenshot 1

TraceMon will prevent the user from entering erroneous information (such as numbers in the ticker text fields or an invalid email address), as well as highlighting duplicate tickers [Figure 4.2].

These settings will take effect the next time the program is run in execution mode. The option of enabling/disabling reports was born from the concept that not all reports need to run on a day-to-day basis, and sometimes the user may want to generate a specific report on-command (without also running the others). For a more detailed explanation of on-command generation of reports, see section 4.7.3.

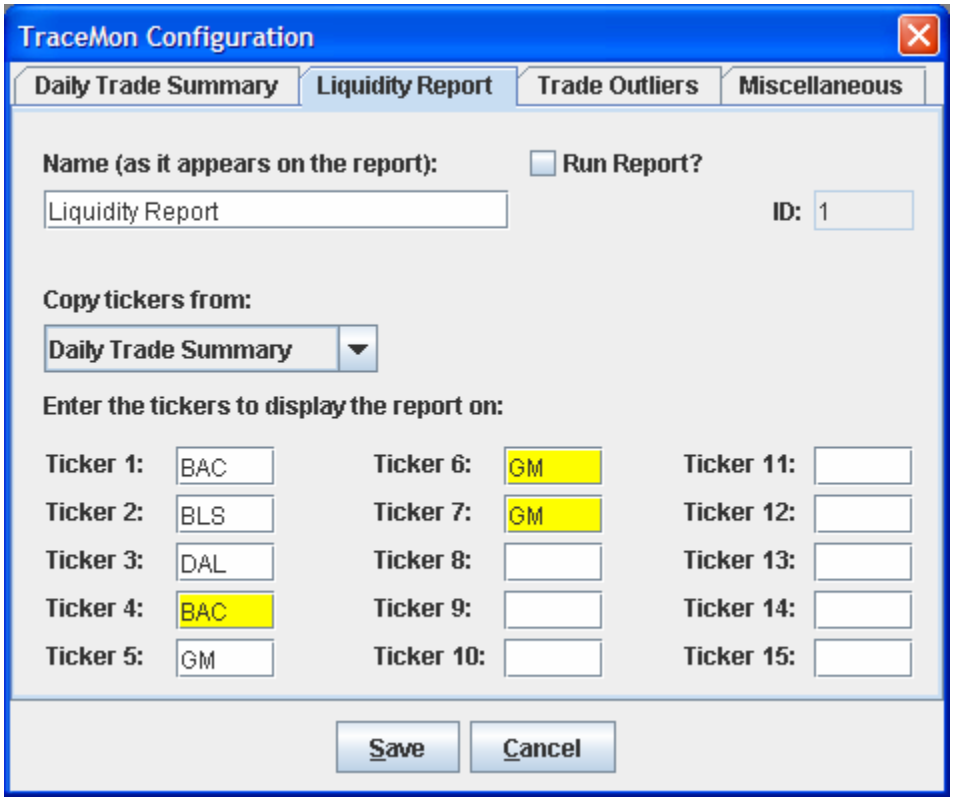


Figure 4.2: Configuration Mode Screenshot 2

Also, the Configuration Window allows the user to set more general settings [Figure 4.3], which do not relate to a particular report. Such information includes the Estimated Capped Amount for trades whose exact amount is not disseminated by the NASD (*Average Investment Grade Amount* and *Average High Yield Amount*) and email addresses of the recipients of the reports.

All the information that is entered is saved to the user's own profile, thus allowing multiple users to use our application at the same time as well as protecting the confidential information of each user. All the user profiles are located in the *profiles* directory from where the application is run, and each profile is stored in its own file bearing the user's login name.

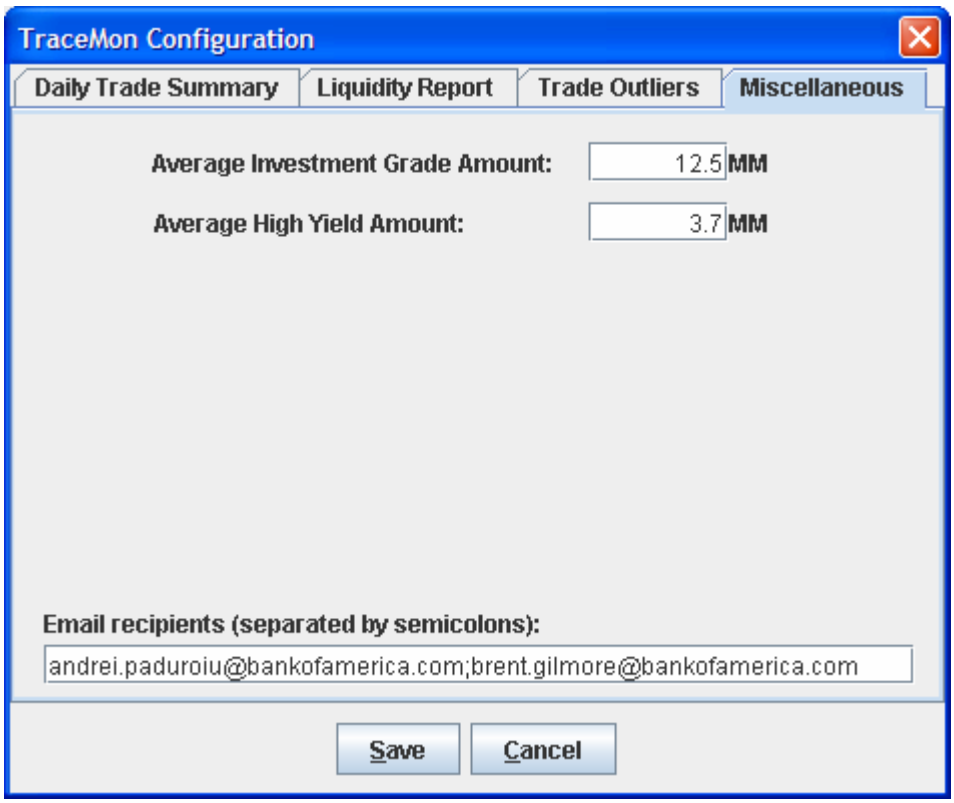


Figure 4.3: Configuration Mode Screenshot 3

4.7.2 Execution Mode

The user can carry out the execution mode through the *"run.bat"* shell script. This will open a command prompt window and automatically begin retrieving the necessary data and performing calculations to generate the reports selected in configuration mode. When all the chosen reports have been completed and sent through email, the command prompt window automatically closes and the program concludes. The *"run.bat"* script can be executed either manually or on a predetermined schedule. In most cases, this function will be scheduled to run overnight. This way, upon logging on in the morning the reports are waiting in the user's inbox.

Daily Trade Summary

The first report, Daily Trade Summary, provides the user an overall synopsis of trading activity for the previous day. Figure 4.4 shows a sample output for the ticker KFT (Kraft Foods Inc.) generated on November 28, 2006 and containing data from November 27, 2006. The report can be broken into three sections: descriptive bond information, trade data and analytical instruments. The descriptive section is comprised of the first six columns which contain static information that is commonly used to identify bonds (such as *Ticker, Coupon, Maturity* and *Cusip*) and other statistics that provide the user with more details about the bond (*Opt* and O*utstanding Notional*). The purpose of this first section is to provide the user with a brief description of the characteristics of each bond.

The next section of the report provides the bulk of the actual trade data. It contains the *Traded Notional* and *Number of Trades* columns. The user is able to plainly see the total volume of trading activity, as well as the size of the trades. This gives the user a good deal of insight into which bonds have been traded heavily or lightly in the preceding day. The remaining columns provide basic analytical information to the user. The report lists the high and low spreads-to-Treasury and prices for each bond during the prior day. This is important because investment grade bonds are valued based on spread-to-Treasury. For high yield bonds, the price statistic serves this purpose. The treasury spread is based on a certain benchmark, which is described in the *UST Benchmark* column. Lastly, the ratings tell the user whether the bond is investment grade or high yield, and what level of risk it is viewed as having. This is extremely beneficial to the user by allowing trading activity and pricing action to be easily monitored.

Report for ticker KFT on 11/27/2006:

Ticker	Coupon	Maturity	Opt[1]	Cusip	Outstanding Notional (mm)	Traded Notional (mm)	Number of Trades		High Price	High Spread	Low Price	Low Spread	Avg. Price	Avg. Spread	UST[2] Benchmark	Ratings		
							>1M	>5M								Moody	S&P	Fitch
KFT	5.250	06/01/2007	NC	50075NAG9	1000.0	1.0	1	0	99.900	73	99.900	73	99.900	73	UST 4.875 10/08	A3	BBB+	A-
KFT	4.000	10/01/2008	NC	50075NAK0	700.0	2.2	2	0	98.000	48	97.913	43	97.957	45	UST 4.875 10/08	A3	BBB+	A-
KFT	4.125	11/12/2009	NC	50075NAM6	750.0	6.0	2	1	97.377	50	97.358	50	97.368	50	UST 4.625 11/09	A3	BBB+	A-
KFT	5.625	11/01/2011	NC	50075NAB0	2000.0	21.4	5	1	102.001	68	101.650	62	101.827	65	UST 4.625 10/11	A3	BBB+	A-
KFT	6.250	06/01/2012	NC	50075NAH7	1500.0	2.0	2	0	104.430	74	104.398	73	104.414	73	UST 4.625 10/11	A3	BBB+	A-

Legend

[1] Optionality. **P** stands for Puttable Bond, **C** for Callable Bond, **P/C** for both, **NC** for none.
[2] U.S. Treasury Benchmarks. "UST 4.5 02/36" represents a Treasury Bond with coupon rate of 4.5 and maturity date of Feb. 2036.

Report generated on: 11/28/2006 10:19

Figure 4.4: Daily Trade Summary Sample Output

Liquidity Report

The next report available is the Liquidity Report. It is designed to provide the user with a basic concept of the overall liquidity of each bond. This is shown through historical trade volumes and also as a percentage of the total amount outstanding, with both being reported on a monthly basis. A sample output of this report can be seen in Figure 4.5. Similar to the Daily Trade Summary, the first three columns contain descriptive data and the remaining columns contain analytical data. The columns contain data for each month within the past year, as well as average monthly values for each bond. From these numbers, the user is able to estimate how long it will take to invest a certain amount of money in a given bond. The user can also get a general idea of the seasonality of a certain bond by examining the percentage of outstanding notional traded in each month. To a trader, this report can serve as a tool for benchmarking

Report for ticker KFT as of 11/28/2006:

Ticker	Bond Description		Outstanding Notional	Avg Monthly Traded Notional	Nov 2006	Oct 2006	Sep 2006	Aug 2006	Jul 2006	Jun 2006	May 2006	Apr 2006	Mar 2006	Feb 2006	Jan 2006	Dec 2005
	Coupon	Maturity														
KFT	5.250	06/01/2007	1000.0	121.4 12.14%	29.1 2.91%	85.8 8.58%	103.1 10.31%	160.5 16.04%	101.6 10.16%	137.1 13.71%	77.4 7.74%	201.6 20.16%	213.6 21.36%	107.0 10.70%	44.7 4.47%	194.9 19.49%
KFT	4.000	10/01/2008	700.0	55.6 7.94%	11.6 1.66%	27.3 3.90%	77.2 11.03%	97.2 13.89%	54.3 7.75%	48.5 6.93%	41.8 5.97%	6.1 0.87%	27.5 3.93%	102.8 14.69%	163.2 23.32%	9.3 1.33%
KFT	4.125	11/12/2009	750.0	77.0 10.27%	71.7 9.57%	21.8 2.91%	32.1 4.28%	145.2 19.36%	212.6 28.35%	86.1 11.47%	30.9 4.12%	29.8 3.97%	41.3 5.50%	101.2 13.50%	86.5 11.54%	64.9 8.65%
KFT	7.000	06/15/2011	200.0	1.1 0.56%	1.4 0.68%	0.9 0.47%	1.1 0.53%	0.9 0.43%	1.0 0.48%	3.1 1.54%	1.2 0.62%	0.7 0.36%	0.8 0.41%	0.4 0.21%	0.9 0.47%	1.1 0.56%
KFT	5.625	11/01/2011	2000.0	179.8 8.99%	217.8 10.89%	307.0 15.35%	113.6 5.68%	181.9 9.10%	78.7 3.94%	273.7 13.68%	166.5 8.32%	169.1 8.46%	198.8 9.94%	207.5 10.38%	224.3 11.21%	18.0 0.90%
KFT	6.250	06/01/2012	1500.0	207.9 13.86%	335.8 22.39%	144.4 9.63%	163.9 10.93%	364.0 24.27%	213.3 14.22%	124.8 8.32%	337.0 22.47%	244.4 16.29%	67.3 4.49%	181.6 12.11%	164.8 10.99%	153.8 10.25%
KFT	5.250	10/01/2013	800.0	67.9 8.48%	120.3 15.04%	52.6 6.57%	45.8 5.73%	18.6 2.33%	39.2 4.90%	110.6 13.82%	62.6 7.82%	72.2 9.03%	71.9 8.98%	91.9 11.49%	59.6 7.46%	69.1 8.63%
KFT	6.500	11/01/2031	750.0	61.5 8.20%	73.6 9.82%	40.6 5.41%	90.9 12.12%	67.7 9.03%	90.3 12.04%	70.0 9.33%	61.8 8.24%	24.4 3.26%	74.4 9.91%	57.1 7.61%	58.4 7.78%	28.4 3.79%

Report generated on: 11/28/2006 10:20

Figure 4.5: Liquidity Report Sample Output

regular trading activity for a bond. It provides the user with context from which a potential trading strategy may be developed. This is extremely useful information for a trader attempting to establish a position on a bond.

Trade Outliers

The Trade Outliers report provides the user with more information about traded volumes as can be seen in Figure 4.6. As in the first two reports, standard descriptive bond data is displayed in the first three columns under the heading *Bond Information*. The next two columns show the traded volume for the current day and an average of the past five trading days. Holidays when the market is closed and weekends are not taken into account in the five day average. The remaining columns include historical averages for comparison. This report gives the user the benefit of being able to easily see if a bond is being traded at volumes vastly above or below usual averages.

Report for ticker KFT as of 11/28/2006:

Bond Information				Recent Traded Volume			Average Traded Volume		
Ticker	Coupon	Maturity	Outstanding Notional	Last Trading Day	Last 5 Trading Days	1 Month	3 Months	6 Months	1 Year
KFT	5.250	06/01/2007	1000.0	1.0	0.2	2.3	2.6	4.0	4.6
KFT	4.000	10/01/2008	700.0	2.2	1.1	0.4	1.4	2.0	2.1
KFT	4.125	11/12/2009	750.0	6.0	1.4	3.1	1.5	4.0	3.2
KFT	5.625	11/01/2011	2000.0	21.4	4.5	14.1	9.1	8.4	7.6
KFT	6.250	06/01/2012	1500.0	2.0	3.4	16.7	9.8	9.9	9.2
KFT	5.250	10/01/2013	800.0	0.0	0.0	5.6	3.0	3.0	2.9
KFT	6.500	11/01/2031	750.0	0.0	0.0	3.1	3.0	3.1	2.6

Report generated on: 11/28/2006 10:20

Figure 4.6: Trade Outliers Sample Output

4.7.3 Advanced Options

Generating Reports on-command

TraceMon can support both reports that need to be run on a scheduled timetable as well as reports that can be run on-command, whenever the user wants. Through the Configuration Window, the user can set the reports that will be run on a scheduled basis (by checking the "*Run Report?*" checkbox on each report tab). However, the user will want to run certain reports on-command, either because information in it does not change very often or because he/she needs to know the results as soon as possible.

Our application supports this feature, which can be activated through a command-line parameter. By supplying the parameters "/run <report_ids>" to the "run.bat" script file, the user can force the reports identified by <*report_ids*> to be run. The IDs of the reports in <*report_ids*> can be obtained (visually) from the Configuration Window (located in the top-right corner of each report tab) and have to be separated by commas (and no spaces).

For example, the command:

```
run.bat /run 0,2
```

will execute only the *Daily Trade Summary* (0) and the *Trade Outliers* (2) reports, thus disregarding the user preferences in the user's profile (however this will not affect the profile, so on the next scheduled run the program will run normally again). Other preferences, such as Estimated Capped Amounts and email addresses will still be loaded from the profile and used to generate the reports and sent their outputs.

Running Reports for Other Users

Another feature of *TraceMon* is running reports for other uses. While normally the application can determine who is logged in on the machine where it is executed, this feature may

be useful when the application may need to run on a remote server with a certain user's profile. Reasons for this may include, but are not limited to, the unavailability of the user's machine or need for increased performance (as noted in the previous chapter, it could take a while before a report is generated, depending on the number of bonds and number of available bonds in *ALICE*)

This feature can be activated through a command-line parameter. By supplying the parameters "/usernbk <userid>" to the "run.bat" script file, the user can specify which user's profile to load. The only pre-requisite for this action is that the user specified by the given id already has a profile (for privacy concerns, a profile can only be created or changed for the currently logged in user, and not for another user).

For example, the command:

```
run.bat /usrnbk nbkht5q
```

will load the user profile of *nbkht5q* and generate the reports on his behalf (thus sending the results to whatever email addresses that user specified in his/her profile).

Also, both of the above-mentioned features can be combined. As a conclusion, one or more reports can be forcibly generated using the preferences of a specified user.

For example, the command:

```
run.bat /run 0,2 /usernbk nbkht5q
```

or

```
run.bat /usernbk nbkht5q /run 0,2
```

will execute only the *Daily Trade Summary* (0) and the *Trade Outliers* (2) reports using the settings in user *nbkht5q's* profile, but disregarding whatever reports that user enabled or disabled.

4.7.4 Identifying and fixing problems

For debugging purposes, *TraceMon* records all actions that it is performing. Such actions include errors, warning and regular events. An *error* happens when something unexpectedly goes wrong and it may negatively affect the output of the program or the stability of the program itself. A *warning* is recorded when something does not happen as planned, but this does not affect the accuracy of the outputted data. An *event* represents a notification that some action has happened or is about to happen; it is not as important as a warning or an error, but it can give valuable information about what the program is doing.

TraceMon records all its actions, including information retrieval for all the bonds it needs. Since multi-threading is used to extract this information in parallel, the log can become very difficult to comprehend by simply looking at it (since messages from more than a thousand bonds can be intermixed in it). Therefore, in order to ease our problem identification and fixing process, we have developed another module of the application, named the *EventLogger*. As it can be observed in Figures 4.7 and 4.8, this subsystem, when launched, shows all the logs from the previous five days (logs older than that are automatically erased). When the user selects a specific log, the program parses it and organizes the information in it based on the bond for which it relates and groups the other information under another tab, namely "*General*" [Figure 4.7]. The *General* tab contains notifications not related to any particular bond, but to the application as a whole (e.g., loading the configuration, the Treasury Information, processing reports or sending emails).

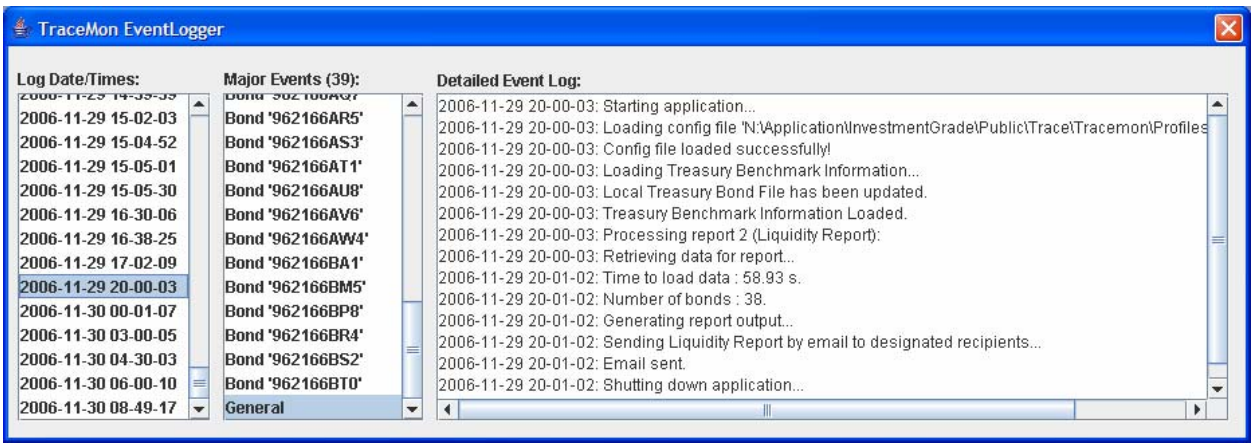


Figure 4.7: EventLogger Screenshot 1

Figure 4.8 shows how the information is filtered out and grouped for each bond in part. It shows all the relevant information for it, including where it was retrieved from and how long it took to fully do that. This example pertains to a bond that was not found in any Bank of America systems, thus it needed to be imported from Bloomberg. While this screenshot does not show it, this would also be the place where errors and stack traces would be shown, as well as other failures and warnings related to this bond in particular. This module has been of utmost importance in our successful deployment of the application, as it allowed us not only to identify and fix various problems, but also to fine-tune our program for performance.

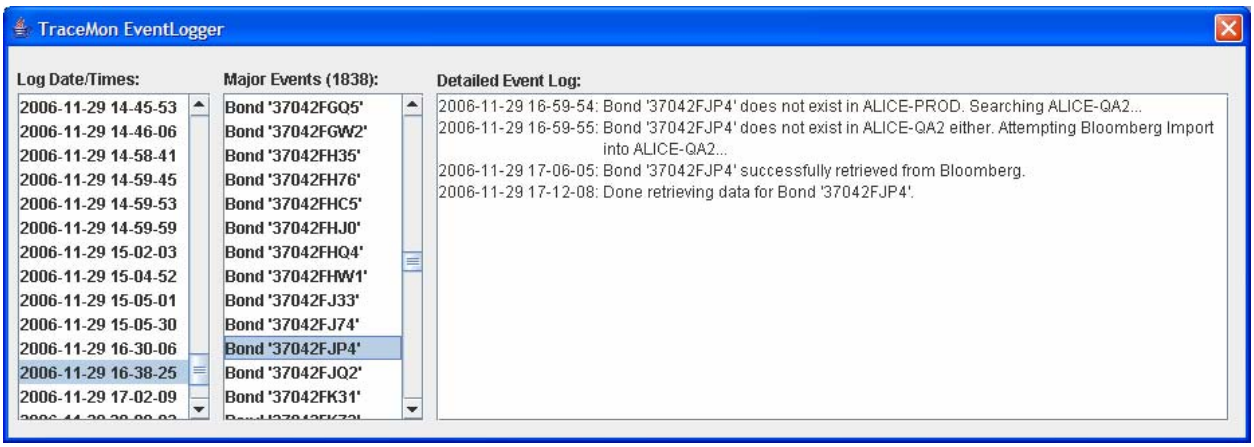


Figure 4.8: EventLogger Screenshot 2

4.7.5 Documentation

The final and most important step that had to be done in order to have a full release was to create appropriate documentation for users of the program and other developers that would support it. This included a *User Manual*, *Software Development Guide* and *Troubleshooting Guide*.

User Manual

The purpose of the User Manual is to make new users familiar to the application, as well as instruct them on how to use it and take full advantage of its features. It contains a brief description of the product, information on where to find it and how to start it, as well as detailed instructions (with accompanying graphics) on how to configure it. After the user learns how to set up his/her profile, the manual shows how to schedule reports or how to use more advanced features, such as running reports on-command or running the program using another user's profile.

Software Development Guide

Besides the User Manual, we wrote a *Software Development Guide*, whose primary audience is the developers who have to support our application after our departure from the site. It contains a detailed outline of the architecture, as well as information regarding the program behavior that would make understanding the application easier. Also, we included comprehensive code comments and used the JavaDoc utility that was shipped with Java JRE 1.5 to generate a complete code guide for our application, which will enable the developers to easier interpret our code.

Troubleshooting Guide

Finally, we decided to compile a list of known issues that were either beyond our control or that would take too long for us to fix. While these problems do not generally affect the user experience (they may appear in rare cases, in less than 5% of all runs), a guide to handling them would definitely save users time in case such a problem occurs. It includes symptoms, a brief description of the cause that may have led to that problem, as well as step-by-step instructions on how to remedy it or direct users to the appropriate developer that will address the issue.

5. Conclusions and Recommendations

The creation of *TraceMon* provided bond traders at Bank of America with a very useful tool. Previously, the data reported by our application was either tediously searched for or simply not used. By processing a vast amount of information and condensing it into a straightforward series of reports, this program quickly provides statistics to the user that would otherwise be unrealistic to determine. With the advent of *TraceMon*, this valuable data is now easily accessible and can provide the user a great deal of insight into the corporate bond market. Because of this insight, traders are able to make better investment decisions and therefore increase overall profitability of the credit and equities trading department.

While *TraceMon* represents a strong beginning, the analysis of TRACE data should be significantly expanded on at Bank of America. Several additions beyond the scope of this project should be made, including improving estimates, increasing the number of reports, and different methods of accessing the report. The expansion of these TRACE analytic measures will further increase the traders' knowledge of the bond market, leading to more lucrative investments.

5.1 Improve Estimated Value Accuracy

One of the few limitations of *TraceMon* is the inaccuracy of the estimated average values for capped trades. Currently, the user is required to input estimators for trades that are reported as greater than $1 million and $5 million with default values of $3.7 million and $12.5 million, respectively. However, traded volume is one of the most important statistics reported by TRACE, and in order to properly analyze the data reported by *TraceMon* the estimates for the capped volumes must be as accurate as possible. Two measures must be taken to maximize accuracy of these estimates: they must be calculated on a daily basis from the aggregate data

reported by NASD, and they must be adjusted based upon the size of the issuer. Accomplishing this would require a significant time investment, as calculating the averages on a daily basis requires considerable computations.

After speaking with Ola Persson at NASD [Meeting Minutes, Appendix A], we were able to understand how to calculate the averages, but did not have the necessary resources. Ideally, Bank of America would create an application to compute these figures. The program would need to extract the total traded volume for both investment grade and high yield trades. The program would then need to sum the volume all of uncapped investment grade trades (those with volumes of $5 million or less) and subtract that number from the aggregate volume. This would yield the total volume of all capped trades. By dividing this by the number of capped trades from that day, a moderately accurate average could be attained. This same process would then be repeated for high yield trades.

Despite the increased correctness of these estimates, they must still be adjusted for the size of the issuer. Bonds with certain tickers are more likely to trade at ten times the estimated value, while other tickers may trade at half of that number. Because of this, the user should be able to input multipliers for tickers that will make the necessary adjustments.

If both of these measures are implemented in future versions of *TraceMon*, the reports it generates will be much more accurate and informative. While the estimates currently used by *TraceMon* are effective, further developments could enhance this aspect of the program considerably.

5.2 Diversify Reports and Organize Them

During this project, we could only develop three reports that pertain to TRACE data analysis. Those include *Daily Trade Summary*, *Liquidity Report* and *Trade Outliers*. While the

reports themselves were not difficult to design, finding all the sources of information, researching technologies that we could use, as well as developing the backbone of our application occupied most of the time that we spent on this project. The project stakeholder mentioned to us that there are many more such reports he would find useful, and these three are just the beginning. We believe that a future project on this topic could be to further extend our application in order to generate more useful reports. Since the foundation and business logic layers are already developed, the development of such reports should not take too long to complete.

Also, our three reports can be currently categorized into daily (*Daily Trade Summary* and *Trade Outliers*) and weekly reports (*Liquidity Report*). Given our current design, it is not complicated to generate them daily (by scheduling the program to run every night and enable some reports) or scheduling a particular report to run after a longer time (using the on-command generation feature of our application). However, if other reports are added, the situation may change. It is possible that several reports need to be run daily, some weekly and some at a longer interval of time (e.g., monthly). Therefore, a further extension of our program would include a customized internal scheduler, which will allow each report to be scheduled as the user wishes, without having him/her use the Task Scheduler in Microsoft Windows.

5.3 Expand Report Format

Currently, the information in each report is compiled into a single email and then sent to the user. While this may not be a problem in the general case, in some of our experiments (where we tested the limits of our program), we realized that some emails can be quite big (5000 lines) and thus very difficult to read. To facilitate navigation through the email, we included links at the beginning of the report to where each new ticker starts (thus the user can “jump” to where he/she

wants). In any case, there are a number of other ways to make these emails more readable (but implementing them would go beyond the purpose of our project).

A first idea would be to generate all these reports and post them to an internal web server. This way, they can be easily accessed at a later time and, by removing the one-page constraint (all tickers need to be on the same page, so that they would fit into an email), the user can click on a hyperlink on the report webpage and be shown another page with the report results for that particular ticker. Using this approach the user would avoid having to search through thousands of possible lines of the email for a particular ticker. Also, a good report-organizing technique would allow the user to see all reports generated for a ticker (i.e., Daily Trade Summary, Liquidity Report and Trade Outliers). Right now, each such report is in a different email, and the user has to open that and search for the ticker before finding the information that he/she needs.

On the same topic, another suggestion that would improve the readability of large reports is the use of DHTML (Dynamic Hypertext Markup Language) combined with JavaScript (as opposed to simple HTML, which is used at this moment). DHTML would be used to "hide" the tables and only show the tickers (on the vertical), with a "+" button on their left. When the user clicks on that button, the corresponding table would be shown and explored by the user. It could later be hidden by clicking the same button that was used to show it.

5.4 Other Suggestions

The abovementioned suggestions for expansion refer mainly to aesthetics and user experience. However, there are several things that a future project could concentrate on, such as an improved data retrieval mechanism or a centralized user repository.

TraceMon has two main sources of data, the *RealTic* database and *ALICE*. While *RealTic* is a reliable and fast data source, it does not contain all the information we need. *ALICE* has a

comprehensive set of static bond information, which is updated on request, and complements the information we get from *RealTic*. However, the major drawback of *ALICE* is that it is sluggish, and can be extremely slow when we need to have some bond information imported from Bloomberg into it. This is the main reason our application can perform very poorly in some instances, especially when the bonds it is dealing with are not found in it. Also, if it attempts to import several bonds into *ALICE* at the same time (which may happen during the execution of our program), it may be denied access until some resources are freed up. This caused several inconveniences from the development point of view, and forced us to implement failsafe mechanisms (such as having to deal with more than one *ALICE* server or retrying to retrieve bond information if it failed due to a server problem – see Section 4.4) – thus adding complexity to our application. A suggestion for improving performance in the future would be to look for alternate data sources or find a better way to get the necessary information from the systems already in place. However, our limited time here prevented us from doing further research into this matter.

If *TraceMon* proves itself to be useful, it may be adopted by several traders that work at Bank of America. While at the current moment it can successfully support multiple users using it at the same time, several design modifications could be made to it in order to facilitate multi-user access. Therefore, a centralized user repository can be created, as well as a pool of predetermined scheduled runs for each report. Thus, every time a report is generated, tickers from all the users can be compiled into a single list, which can be used to generate the report. After it is completed, individual reports can be sent to each user based on the tickers he/she specified. This would significantly improve system performance, as it will not have to generate duplicate reports for the same ticker at the same time.

Appendix A: Meeting Minutes

Initial Project Meeting

Tuesday, October 24, 2006 9:30-10:00

Purpose: Introduce basic project parameters and requirements
Present: Brent Gilmore, Martin Gonzalez, Andrei Paduroiu, Kurt Vile

TRACE:

Data reporting and disseminating service developed by the NASD to enhance transparency in the corporate debt market

Initially, only transactions involving the highest rated bonds were reported and disseminated

TRACE collects a large amount of bond pricing information, but disseminates only select statistics

TRACE can help bond traders gain a better understanding of what the market is doing by:

- Determining ease of accumulation
- Flagging spikes in activity between quarterly reports
- Determining volume outliers

Daily, weekly and monthly bond volumes can help find high volume bonds with unnaturally high liquidity

Outcomes:

Background report requested to gain a better understanding of TRACE data

NASD Call

Monday, November 6, 2006 15:45-16:15

Purpose: Inquire about NASD reports containing capped trade averages
Present: Brent Gilmore, Ola Persson

Discussion:

When asked if the NASD periodically reported average values for capped trades (those greater than $1 million or $5 million), Ola told us that the NASD no longer produced those reports because the averages were being easily calculated by many financial institutions. He also informed us that the NASD publishes aggregate trade volumes on its website on a daily basis, and that it is not difficult to calculate the desired averages from these figures.

ALICE Database Conference Call

Wednesday, November 08, 2006 15:00-15:30

Purpose: Discuss alternatives to using *ALICE* production database
Present: Brent Gilmore, Hoang C Nguyen, Andrei Paduroiu, Charles Waddington

Problem[8]:

Our application, *TraceMon*, was accessing the *ALICE* database to obtain static bond data such as coupon, maturity and bond ratings on bonds listed in the *RealTic* database. However, when a bond was listed in *RealTic* and not in *ALICE*, *TraceMon* would force *ALICE* to retrieve the static data from Bloomberg. This was a problem because it was causing bonds to be imported into *ALICE* without undergoing the proper processing.

Potential Solutions:

Charles and Hoang suggested two possible alternatives to pulling this data into *ALICE*: work in either a Quality Assurance environment (QA or QA2) or the User Acceptance Testing (UAT) environment. If *TraceMon* was to import any new bonds into either the QA or UAT environments, it would not disrupt production as the current process was doing.

Outcomes:

As a collective we decided to work in the QA2 environment because it was not refreshing on a nightly basis like the other environments were. The refreshing presented a problem to our application because it would delete any bonds *TraceMon* pulled the previous night. However, in the QA2 environment we were able to import bonds without disrupting other processes and without them being deleted on a nightly basis.

[8] A detailed explanation of the reason this problem occurred is described below:

Workflow Concerns

One of the features of *Alice* is associating securities with an issuer from ACID. This ensures that the associated issuer information is valid and accepted by the bank. This usually means that before any security is allowed into *Alice*, the issuer and guarantor name is checked against a collection of verified issuers from ACID. If the name is not recognized as the legal name or an alias, the security is sent to a workflow process. There, a person will either associated the unknown name as an alias to an existing issuer or create a new issuer in ACID.

However, this presents a problem for the retrieve operations since it is possible the application issuing the requests could be a batch or service one. As a compromise, any security imported by the retrieve operations will not trigger a workflow process. Instead, if the issuer is not known, the issuer data in the security will be "null", but the issuer name will be available in "*sourceIssuerName*" and "*longCompName*" fields on the Bond element. "*sourceIssuerName*" is equal to Bloomberg's short issuer name and "*longCompName*" is equal to Bloomberg's full issuer name.

TraceMon Demonstration

Monday, November 20, 2006 16:00-17:00

Purpose: Provide primary project stakeholder with overview of *TraceMon*
Present: Brent Gilmore, Martin Gonzalez, Alex Gregory, Andrei Paduroiu

Presentation:

We briefly showed Martin how to use both the configuration and execution modes of *TraceMon*. We demonstrated the program's capabilities with respect to speed and accuracy. Martin was able to see what *TraceMon* is capable of and he provided us with very positive feedback.

Concerns:

The *Outstanding Notional* value was being displayed as either zero or an incorrect value that did not align with the information reported by Bloomberg. We discussed this issue with Martin and after comparing the numbers in *TraceMon* (which are retrieved from *ALICE*) with those reported through Bloomberg and those reported by the issuer, we found that in some cases none of the three figures corresponded. Martin informed us that the numbers reported by the issuer are the most accurate, but the Bloomberg numbers would be sufficient for *TraceMon*.

Potential Solutions:

We conferred with Alex and Martin about possible solutions to this inconsistency. Some of our options included attempting to gain access to external services to which Bank of America is subscribed (such as Mark-it Partners), or a higher level internal service.

Results:

Other than the *Outstanding Notional* value problem, the demonstration was a success. Martin was very pleased with what we showed to him. We were able to solve the *Outstanding Notional* problem by communicating with the *ALICE* design team. They were able to import all the missing values from Bloomberg and update the remaining values to ensure data accuracy.

TraceMon Release for User Acceptance Testing

Wednesday, November 29, 2006 13:30-14:30

Purpose: Provide Martin access to *TraceMon* for testing and review
Present: Brent Gilmore, Martin Gonzalez, Andrei Paduroiu

Release:

We created a public folder on a shared drive to enable all future users to access *TraceMon*.

Issue #1:

We had difficulty with Java Runtime Environment (JRE) compatibility. *TraceMon* requires version 1.5 to run. However, even after successfully installing JRE 1.5 on Martin's computer, we could not run *TraceMon*. We realized that Martin's computer had JRE 1.5, but also JRE 1.3 and JRE 1.4. Further investigation showed that JRE 1.3 was the default one. This introduced a problem that needed to be addressed: the users' machines may have various configurations, and while JRE 1.5 may be installed there, it may not be the default one

Solution #1:

On the spot, this problem was solved by modifying the batch files that started the application. We forced the JRE 1.5 to be the one that executes our application by hard-coding the path to the JRE bin directory. The initial problem was solved and we could get the application up and running on Martin's machine. But this introduced a new issue.

Issue #2:

By hard-coding the path to the JRE bin folder, we gave up the flexibility in being able to run on multiple update versions of the JRE (Sun, the manufacturer of Java, releases several updates for each JRE version, and each has its own installation folder). The JRE installed on Martin's machine was update 7 (installed in folder jre1.5.0_07) and ours was update 9 (in folder jre1.5.0_09). Thus we needed come with a fix for this problem as soon as possible to accommodate different updated versions of JRE 1.5.

Solution #2:

This problem did not require our presence at Martin's desk, so we could solve it from our own computers. We took advantage of the following property of any Microsoft Windows operating system: when issuing a command from a script host, the operating system first searches the current folder to find that program, and then searches the PATH variable. We have decided to temporarily add some folder paths to the PATH variable representing all the JRE 1.5 update versions. Thus, when the script is executed on any client machine (who needs to have at least one JRE 1.5 installed), at least one of them will be picked up and successfully execute our program).

Outcomes:

With *TraceMon* scheduled to run on a daily basis, Martin was able to experience the application as it will operate once completely launched. Through this, he will be able to provide us feedback and we can make necessary adjustments and changes to *TraceMon* prior to full release.

The Future of *TraceMon*

Thursday, December 7, 2006 15:00-16:00

Purpose: Show other developers/managers how to support and extend *TraceMon* after our departure
Present: Dave Bulthuis, Andrei Paduroiu

Summary:

This was an informational meeting in order for other developers to understand our applications and give them an insight into how it works and how to change it. Andrei had a technical discussion with Dave Bulthuis (Application Development Senior Manager), who oversees a team of software developers in the Chicago office, in order to describe *TraceMon*. The discussion had the following topics:

1. Description and Purpose of TraceMon
2. Sample runs (Configuration and output format)
3. Location of source code, compiled code, as well as documentation
4. Overview of architecture (packages, classes, etc)
5. Description of Flow of Events and explanation of sequence diagrams and use-cases
6. Explanation of some complicated pieces of code and why we chose one method over another
7. Questions and Answers from Dave (in case he wanted any clarifications on this)

Appendix B: Diagrams

Figure B.1: Package Diagram

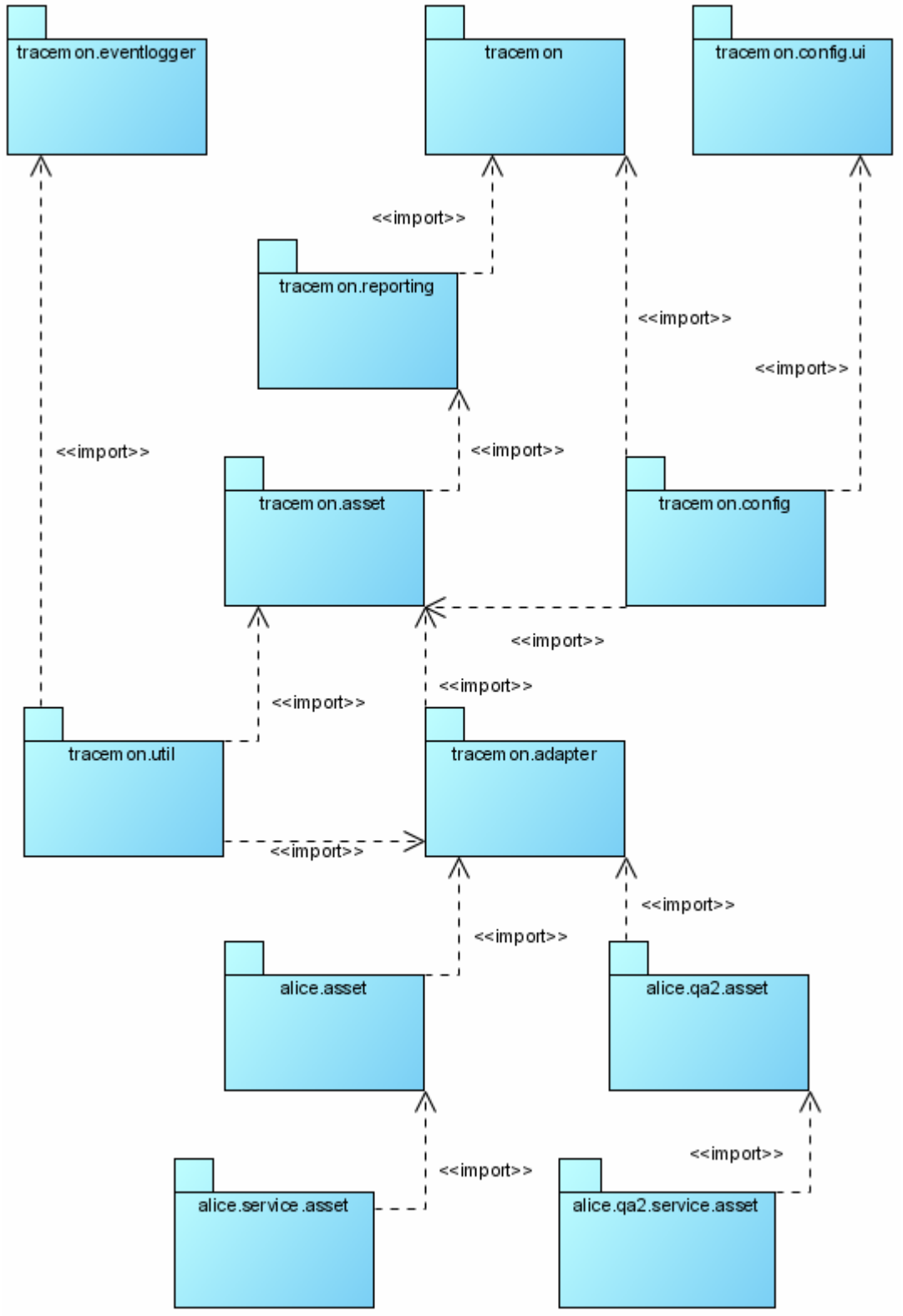


Figure B.2: Class Diagram

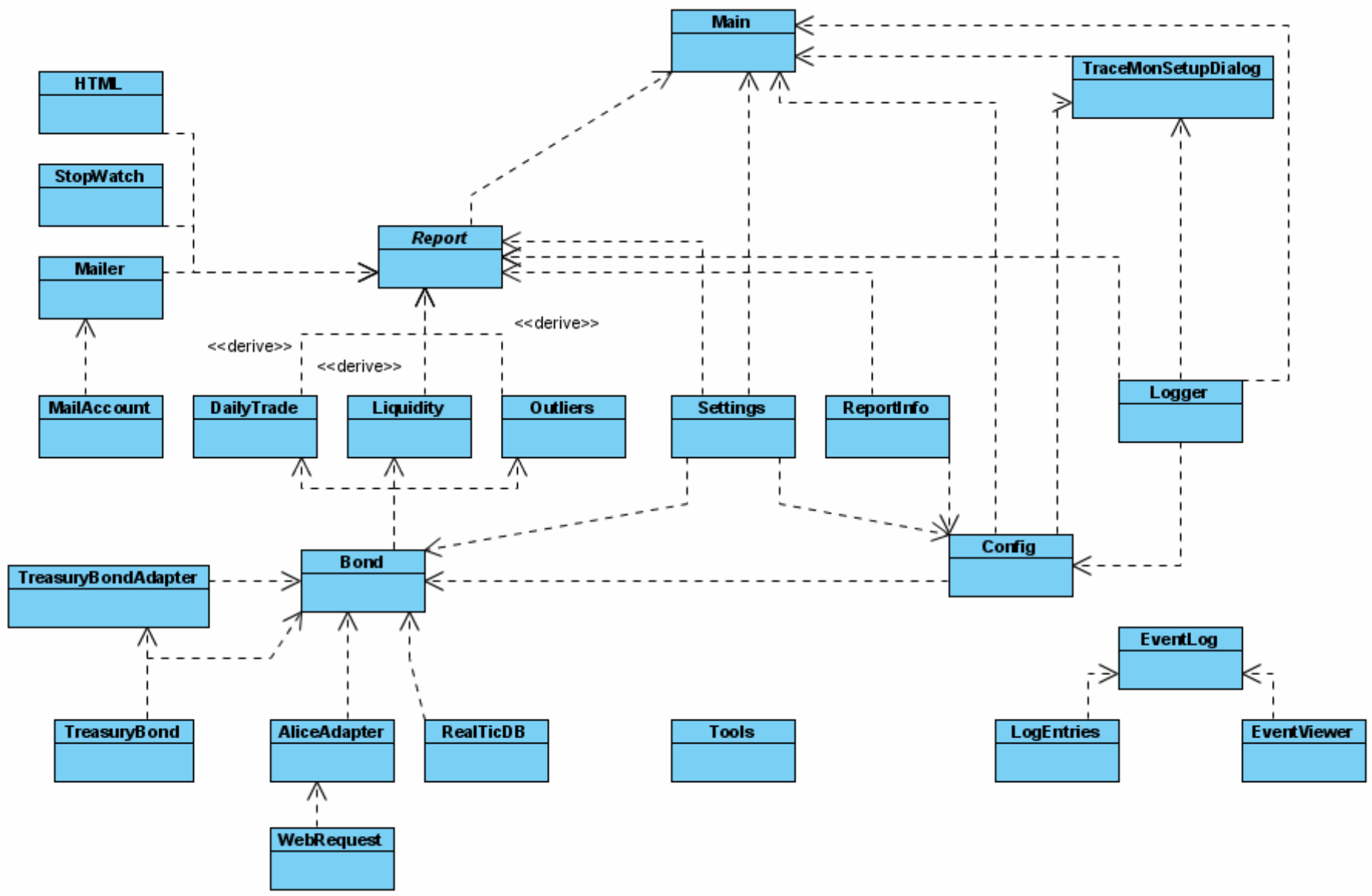


Figure B.3: Sequence Diagram for Execution Mode

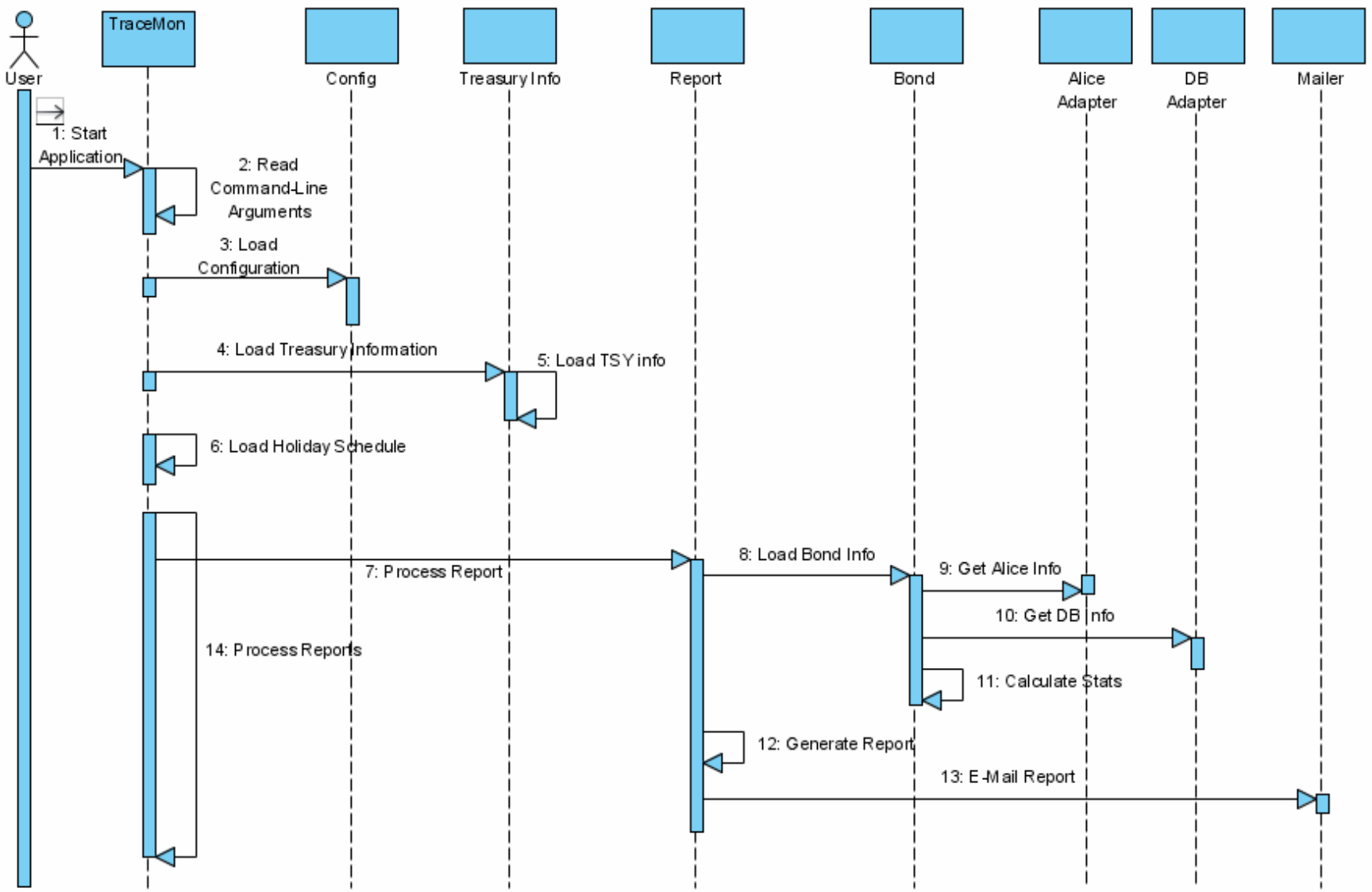


Figure B.4: Sequence Diagram for Configuration Mode

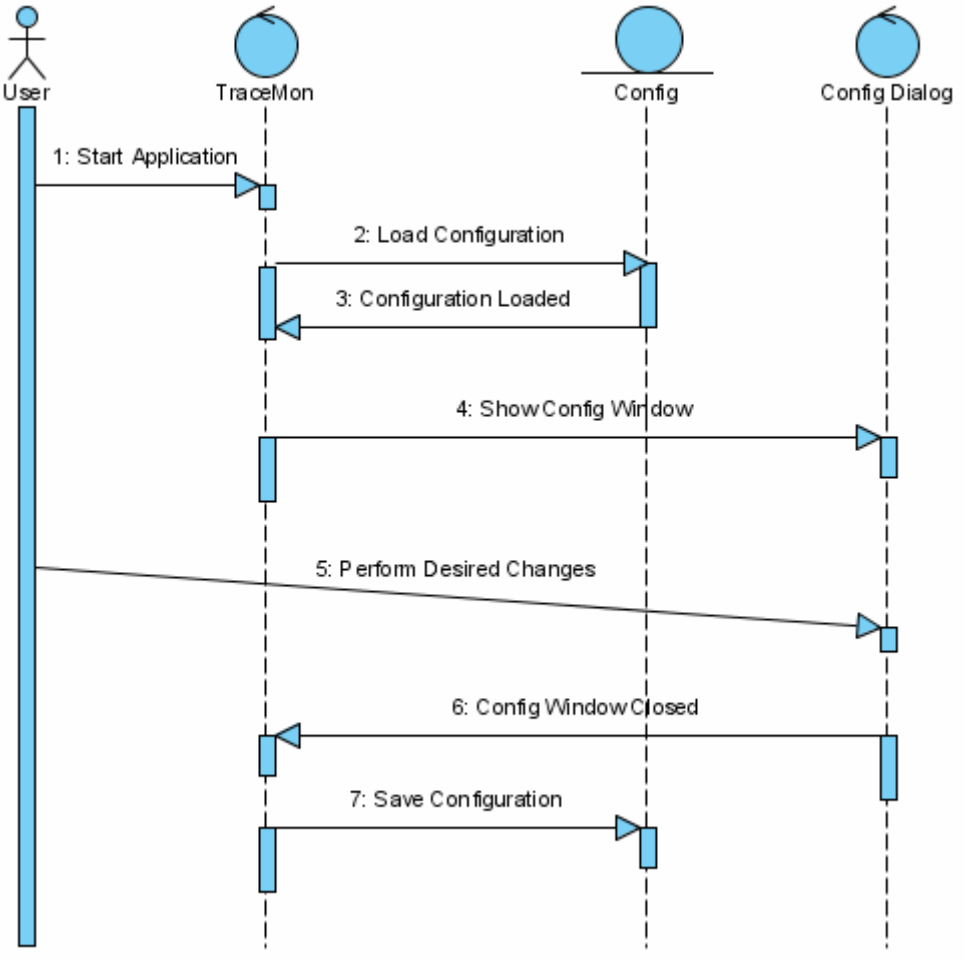


Figure B.5: Use Case Diagram for Configuration Mode

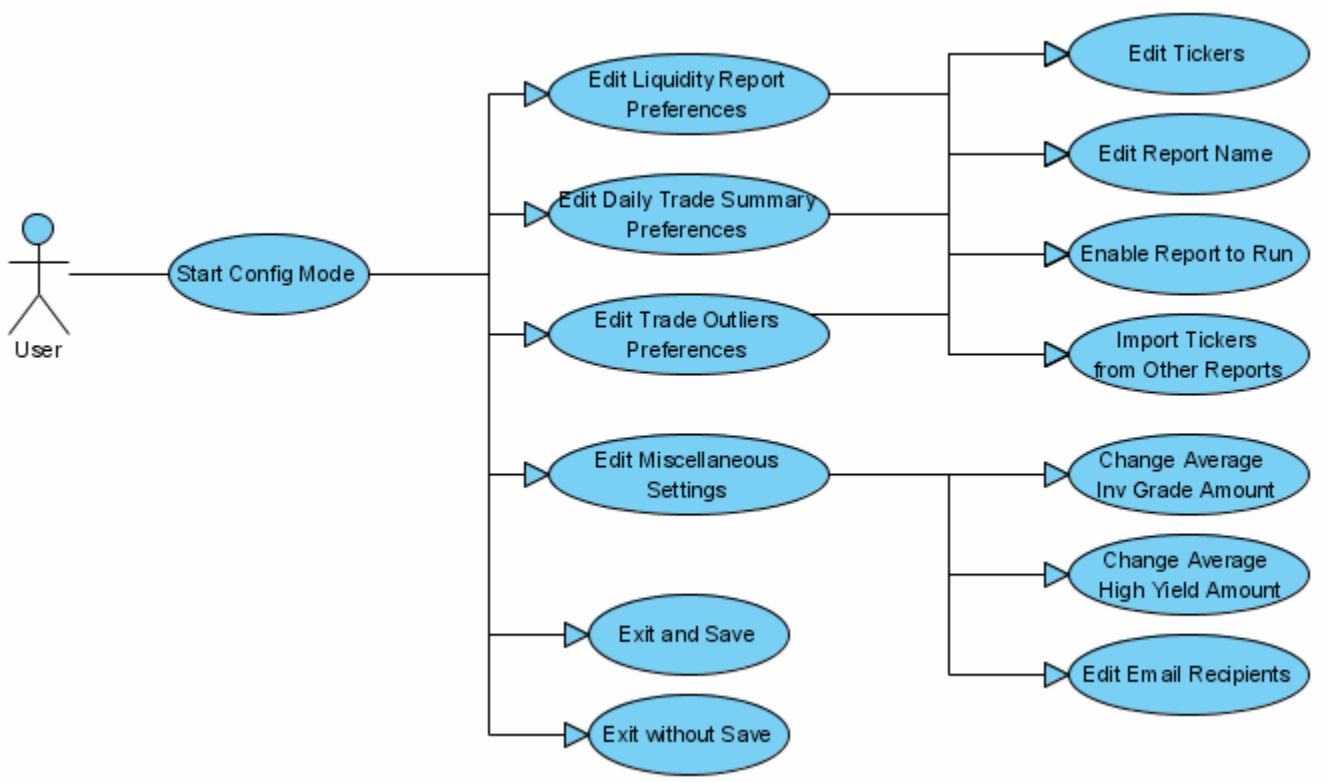
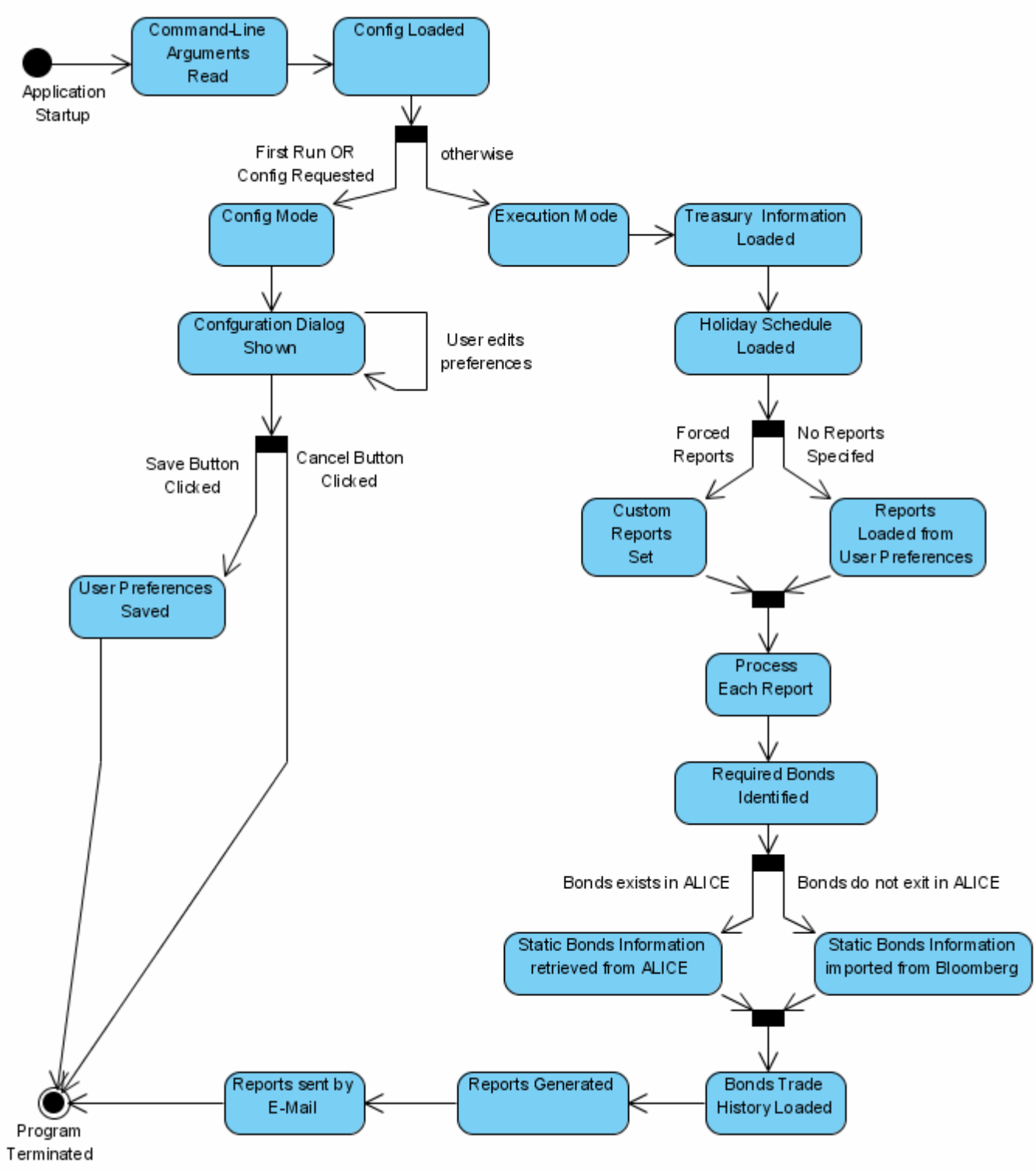


Figure B.6: Flow of Execution Diagram



Bibliography

Bank of America. 2006. *Our Heritage*. Available from Bank of America Intranet, http://flagscape.bankofamerica.com, accessed 7 November 2006.

Bessembinder, Hendrik, William Maxwell, Kumar Venkataraman. 2005. Market Transparency, Liquidity Externalities, And Institutional Trading Costs in Corporate Bonds. Available from Internet, http://www.ivey.uwo.ca/research/IRS_Papers/Bessembinder.pdf, accessed 6 November 2006.

Bond Market Association. 2006. *Corporate Bonds.* Available from Internet, http://www.investinginbonds.com/learnmore.asp?catid=5&subcatid=18, accessed 6 November 2006.

Bond Market Association. 2006. *Association Supports Releasing Historic TRACE Data; Cautions Against Revealing Proprietary Information Which Could Harm Market Participants*. Available from Internet, http://www.bondmarkets.com/story.asp?id=2554, accessed 25 October 2006.

BondsOnline. 2006. *Bond Rating Definitions.* Available from Internet, http://www.bondsonline.com/Bond_Ratings_Definitions.php, accessed 7 November 2006.

NASD. 2006. *NASD Bond Trade Dissemination Service (BTDS).* Available from Internet, http://www.nasd.com/web/groups/reg_systems/documents/regulatory_systems/nasdw_011216.pdf.

NASD. 2006. *NASD's TRACE Completes Real-Time Public Dissemination of Public Corporate Bond Transactions, Beginning Today*. Available from Internet, http://www.nasd.com/PressRoom/NewsReleases/2006NewsReleases/NASDW_015839. accessed 25 October 2006.

NASD. 2006. *Notice to Members - June 2006*. Available from Internet, http://www.nasd.com/web/groups/rules_regs/documents/notice_to_members/nasdw_016989.pdf, accessed 26 October 2006.

SEC. 2006. *The Investor's Advocate: How the SEC Protects Investors, Maintains Market Integrity, and Facilitates Capital Formation.* Available from Internet, http://www.sec.gov/about/whatwedo.shtml, accessed 6 November 2006.

Shulman, Doug. 2004. *An Overview of the Regulation of the Bond Markets.* On-line. Available from Internet, http://www.senate.gov/~banking/_files/shulman.pdf, accessed 26 October 2006.

REUTERS. *An introduction to bond markets*. Singapore, John Wiley & Sons, 1999.

Tuckman, Bruce. *Fixed income securities: tools for today's market, 2nd Edition.* New Jersey, John Wiley & Sons, 2002.

Wikipedia. 2006. *Fitch Ratings*. Available from Internet, http://en.wikipedia.org/wiki/Fitch_Ratings, accessed 6 November 2006.

Wikipedia. 2006. *Standard & Poor's*. Available from Internet, http://en.wikipedia.org/wiki/Standard_%26_Poor%27s, accessed 6 November 2006.

Yahoo Finance. 2006. *Banc of America Securities LLC Company Profile*. Available from Internet, http://biz.yahoo.com/ic/47/47381.html, accessed 7 November 2006.