



Worcester Polytechnic Institute
100 Institute Road, Worcester, MA 01609

Project Number: SJB-GPS3

Formation Flying Test Bed Data Analysis Software Programming Project

A Preliminary Qualifying Project Proposal: Submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
In partial fulfillment of the requirements for the
Degree of Bachelor of Science

By

Ye Wang

Cody Holemo

Date: October 14, 2004

NASA Mentors
Richard Burns
rich.burns@nasa.gov

Dave Gaylor
dave.gaylor@emergentspace.com

WPI Advisors
Stephen Bitar
sjbitar@wpi.edu

Fred Looft
fjlooft@wpi.edu

This document represents the work of WPI students. The opinions expressed in this report are not necessarily those of the Goddard Space Flight Center or the National Aeronautics and Space Administration

Abstract

Our senior design project related to software development for the Formation Flying Test Bed (FFTB) at the National Aeronautics and Space Administration's (NASA) Goddard Space Flight Center (GSFC). Our project consisted of enhancing existing software tools that provide real-time data plotting and data communication for the FFTB. The existing software tools consisted of a simulation plotting package and the software created by the 2003 WPI project team. We improved the data communications of the FFTB by implementing a middleware communications system to replace point-to-point connections. We also developed a real-time plotting application that utilized the middleware.

Acknowledgements

We would like to thank the following people for their assistance and support to our project.

NASA mentors:

Rich Burns
Dave Gaylor
John Higinbotham
Bill Bamford

WPI advisors:

Professor Bitar
Professor Looft

Authorship

Introduction	<i>Both</i>
Background	<i>Both</i>
Project Statement	<i>Both</i>
Methodology	<i>Holemo</i>
Software Design and Development	<i>Wang</i>
Implementation	<i>Both</i>
Testing and Evaluation	<i>Holemo</i>
Conclusions	<i>Wang</i>
References	<i>Both</i>
Appendix A: WPI 2004 Software User Manual	<i>Holemo</i>
Appendix B: XML Schemas	<i>Both</i>
Appendix C: Sample XML Configuration Files	<i>Both</i>
Appendix D: FFTBPlot Software Design	<i>Victor Lu</i>

Executive Summary

Our senior design project relates to software development for the Formation Flying Test Bed (FFTB) at the National Aeronautics and Space Administration's (NASA) Goddard Space Flight Center (GSFC). The FFTB software that we worked on consisted of a simulation plotting package and data communication system.

Our project work benefited the engineers at the GSFC in many ways including enhancements to software that performed processing and analyzing data from simulations and starting the first steps towards employing a middleware to handle data communication within the FFTB. The improvements made to the plotting package allow for real time plotting of simulation data as well as offer easy extendibility for future applications. Overall, our plotting software assists the engineers by making the plotting of simulation data simple, efficient and functional.

Background

We studied several background areas to understand our project and learn essential material to complete the project. The background topics included formation flying, formation flying test beds, the global positioning system, and numerous software resources.

Formation flying is a relatively new concept that refers to multiple spacecraft flying together to achieve a common goal. Utilizing this idea for satellites opens up many capabilities and applications for NASA to use in space. Using distributed formations provides great potential for space science to perform observations that require high resolution imaging. In order to determine the application of the technology, several testing facilities have been created at universities, such as Stanford, MIT, and Boston College to test related hardware and software techniques. Through the simulations and experiments conducted at these facilities, called Formation Flying Test Beds (FFTB), the risk associated with formation flight missions can be reduced. NASA realizes the potential of formation flight for many science applications and has created their own FFTB at the GSFC.

A previous project team from WPI assisted the GSFC, Guidance Navigation and Control Center (GNCC) by developing software components of the FFTB. This previous team created an adapter and a client program (software) that controlled data flow between GPS receivers, flight software and test engineers. The adapter software combined the functionality of the FFTB that was originally divided over several programs, while the client program interfaced with the adapter and implemented a Graphical User Interface (GUI) to control and display data from the adapter.

Project Statement

The goal of our project was to enhance existing FFTB simulation support tools that provide real-time plotting and communications for FFTB test data. The existing software tools consist of a simulation plotting package and the previous project teams' contribution to the FFTB. The software development entailed improving the existing simulation plotting package, FFTBPlot, and improving the data communications between components of the test bed. Each of these improvements had many objectives that worked towards the main goal of our project. An alternative that led to a major design change was abandoning FFTBPlot and creating a new simulation plotting package that we named NetPlot. To

accomplish our objectives we also had several tasks that we completed such as learning advanced topics in Java, becoming familiar with Java development tools, documenting source code, and testing and verifying our software.

Methodology

We took a step-by-step approach to fulfill our goals and objectives in an orderly and timely fashion. First, we discussed with our mentors the specific needs of the test engineers working with the FFTB to narrow the focus of our goals. Then we studied background areas that included formation flying, formation flying test beds, the Global Positioning System (GPS), and various software and programming topics such as Java, Eclipse, the Spread Toolkit, and XML, to become familiar with the application of our project. By studying the background information we gained the intellect to accomplish our goals as well as understand the “big picture” of our project. After researching the essential information we began examining our goals in further detail and decided on a strategy to achieve them. Each goal had several objectives that we prioritized in order to complete the goal in a logical manner. The main objectives also required several stages of implementation and testing to prove the proper functionality and determine possible changes.

Software Design and Development

The software design and development that we performed for this project encompassed work on several different pieces of software. The software product of our project consists of both original software that we designed and developed and existing software that we modified. In both cases of software design, we had to understand the problem in order to identify the software requirements that would solve the problem. In the case of developing new software, we used a top-down design approach, where we first determined the top-level operation of our program. In the case of modifying existing software, we reverse-engineered the existing software to determine how it functioned and needed to be modified in order to meet the software requirements.

The major software applications that we designed or modified were our middleware interface package SpreadXML3, the 2003 WPI project team’s Adapter/Client, our real-time plotting application NetPlot, and our middleware Data Logger utility. Below we document the design and development of these various pieces of software.

Middleware Interface design:

The middleware design concept involves the use of a central application to relay messages between connected clients. This communications scheme has many benefits over the existing point-to-point communications scheme employed by the FFTB. With point-to-point communications, each component needs to handle a connection to every component it needs to communicate with. With the middleware concept, each component only needs to handle one connection to the middleware. All messages sent to the middleware are sent to a specific “subject” (a unique name for a group of messages). In order for a component to receive messages from a specific “subject” it only needs to “subscribe” to that specific “subject”.

SpreadXML3 is the software package that we developed as part of an implementation of the middleware concept. This package employs the libraries of the Spread toolkit to implement the network communication aspect of the middleware concept. The data sent over the middleware is also standardized using a custom XML formatting scheme. Applications utilize the SpreadXML3 package as an interface to the middleware. The actual middleware application is the Spread daemon included with the Spread toolkit.

Adapter/Client modification:

The Adapter/Client package is a key component of the FFTB setup that communicates with various other components of the FFTB. The purpose of the Adapter is to relay data from the GPS receivers to the flight software component. The Client application has the purpose of viewing and archiving the data that is being received by the GPS receivers. The Adapter/Client was a testing ground for our SpreadXML3 package and a start toward implementing the middleware communications scheme for the FFTB. Our work with the Adapter and Client consisted of modifying the communications aspect of the Adapter and Client to utilize the middleware instead of point-to-point communication connections. The Adapter was also modified to support a new GPS receiver type.

The Adapter was modified to broadcast GPS data messages to the middleware by utilizing our SpreadXML3 package, which replaced the source code that managed the point-to-point connections for the Client and flight software. The data is broadcast to subjects based on the type of the message and the GPS receiver that it came from. For example the “F40” GPS data message from an Orion receiver on port 1 would be sent over the subject “Orion1.F40”. The Client was then modified to receive the GPS data messages from the middleware. This consisted of the replacing the source code to handle the point-to-point connection with code that utilizes the SpreadXML3 package.

The Adapter communicates to the GPS receivers via the serial ports in an ASCII format. The Adapter created by the 2003 project team only supported communication with the Orion GPS receivers. As another modification that we made to the Adapter, we added the support to communicate with the Pivot GPS receivers. These receivers also communicate via the serial ports, and supply some of the same data messages as the Orion receivers. The main difference with the Pivot receivers is that the data is formatted slightly differently. We modified the Adapter to automatically recognize whether a Pivot or an Orion receiver is connected to each serial port. The Adapter then processes the raw data from each receiver type appropriately and broadcasts to the middleware with the appropriate subject. For example the “F40” GPS data message from a Pivot receiver on port 2 would be sent over the subject “Pivot2.F40”.

NetPlot design:

An original objective of our project was to modify the existing real-time plotting software, FFTBPlot. The main design requirement for the modifications to FFTBPlot was to add the ability to acquire data over a network connection. Our efforts toward accomplishing this requirement consisted of utilizing our SpreadXML3 package to connect and receive messages from the middleware. We also modified the FFTBPlot user interface to reflect this addition of functionality. However, after some preliminary testing and evaluation of our modifications to FFTBPlot and design discussions with our mentor, we decided to abandon FFTBPlot and develop a new plotting application. The motivating factors

toward abandoning FFTBPlot included the performance issues with the plotting engine, bulkiness of the source code, and ambiguous bugs that are difficult to remove. Attempting to resolve these factors could have actually taken more time than creating a new application.

The plotting application that we designed was called NetPlot. We set the design requirements for this application in order to prevent the design problems seen with FFTBPlot. The design requirements for NetPlot were:

- 1) Data acquisition from Spread middleware
- 2) Utilize an efficient plotting engine
- 3) Simple, intuitive GUI
- 4) Customizable appearance of the data representation
- 5) Command line executable (for scripting)
- 6) Save/load configuration files

In order to support the network communications, we utilized our SpreadXML3 package. We also used a more efficient plotting, PTPlot, which has functionality designed for real-time plotting and customization of data representation appearance.

The GUI for NetPlot is very simple and intuitive. The user can specify a subject that data is being sent across and the application will automatically detect the data on that subject that can be plotted. The user can select multiple data plots in a plot window along with the data representation options for each plot window. Also, multiple plot windows can be opened at the same time. The plot windows display automatically adds data points to the plot as messages are received from the middleware. Each plot window even has the built-in functionality from the PTPlot engine to modify the data representation appearance while it is plotting.

The NetPlot application has the functionality to save and load XML-formatted configuration files that hold the configuration information for the middleware connection and the plot windows. This allows the user to easily save and load frequently used plot setups. NetPlot can also be run from the command line with arguments that specify the configuration files for NetPlot to automatically load. These features allow NetPlot to be completely scripted.

Data Logger design:

The Data Logger utility is a simple application that we created to log the data transmitted across the middleware. We developed this application because the applications that used to the middleware needed some method for archiving the data. This application works as a plug-in to the middleware. Because of the design of the middleware, we were able to add data logging support to all data communicated across the network without modifying any of the programs sending or receiving the data. The application connects to the middleware utilizing our SpreadXML3 package.

The DataLogger application allows the user to specify the subject to log and the location to store the log file. The GUI allows the user to specify and manage the logging of multiple subjects at the same time and gives real-time information such as number of messages received and log file size for each subject. The log files are written in the Microsoft Excel compatible comma-separated-value format. Another important feature of the Data Logger is the ability to save and load XML-formatted configuration files. This allows the user to easily save and load frequently used logging configurations. The Data

Logger can also be run from the command line with arguments that specify the configuration files for Data Logger to automatically load. These features allow Data Logger to be complete scripted.

Implementation

The implementation of our design required the use of various tools and implementation techniques. Some of the implementation techniques that we utilized are advanced or uncommon in most applications. These techniques included creating interface objects, employing multi-threaded processes, and utilizing complex data objects. We also used different programming tools and software packages for other advanced functionality within our software, such as XML interpreting (JAXB libraries) and network communications (Spread toolkit).

For general Java development, we utilized the Eclipse Java integrated development environment from IBM and the Java Standard Developer's Toolkit (SDK) from Sun Microsystems. We used Eclipse version 3.0, which can be found at <http://www.eclipse.org>. Eclipse is open-source software, available for both Linux and Windows operating systems. We used the Java SDK version 1.4.2, which can be found at <http://java.sun.com>. The Java SDK, which is available for several operating systems including Linux and Windows, includes a Java runtime environment and provides the functionality of the Java Application Programming Interface. We developed and compiled our software on both Red Hat Linux machines and Windows XP machines. All of our software was documented using the standards for JavaDoc documentation. This allowed us to use Eclipse to generate a series of HTML files that described all of the classes, methods, variables and structures associated with our software packages.

Testing and Evaluation

Throughout the design and implementation process we tested and evaluated our software to ensure the proper functionality and operation to the design requirements that were established. With the exception of FFTBPlot, we were able to meet the design requirements for the middleware, Adapter/Client, NetPlot and Data Logger. Our tests on the middleware showed it to be reliable, efficient and flexible. The Spread middleware still remains in the first stage of being implemented for the communications within the formation flying test bed, therefore further testing and evaluation will be necessary for handling the future role of being a complete middleware for the test bed.

Conclusions

The vision of the middleware design concept and the impact of our project work extend beyond the brief ten weeks that we had at the Goddard Space Flight Center. Our project is not a standalone product but only a piece of the larger picture of the FFTB. The actions taken by others to utilize, integrate, and improve our software determine the benefits that the FFTB will receive from our work. Therefore, we have given the following recommendations that specify some of the possible future improvements.

Middleware Recommendations

- Develop a C++ version of SpreadXML3 that provides a similar, if not identical, interface to the middleware
- Redesign and optimize the message structure to reduce the message size

- Define a naming standard for the subjects used for communications across the middleware
- Create a standard set of “key” names used to format the data sent across the middleware
- Test the extremes of the middleware performance

NetPlot Recommendations

- Investigate the performance lag exclusive to Windows machines
- Improve the GUI

Data Logger Recommendations

- Enable the feature of selective data logging as opposed to logging all of the data
- Enable the ability to choose the log file format
- Create the feature to read a log file and rebroadcast the logged data
- Improve the GUI

General Recommendations

- Create a top-level application to facilitate the start up of the multiple pieces of software used in an FFTB simulation run
- Develop a real-time data manipulation application
- Document all new source code with the JavaDoc standard

Table of Contents

1	INTRODUCTION.....	1
2	BACKGROUND	2
2.1	NASA.....	2
2.2	FORMATION FLYING	3
2.3	FORMATION FLYING TEST BED.....	5
2.3.1	<i>Description of NASA FFTB</i>	<i>5</i>
2.3.2	<i>Previous WPI Contribution to NASA FFTB</i>	<i>9</i>
2.4	THE GLOBAL POSITIONING SYSTEM.....	12
2.5	JAVA	13
2.6	EXTENSIBLE MARKUP LANGUAGE (XML).....	14
2.7	SUMMARY	16
3	PROJECT STATEMENT.....	17
3.1	GOAL STATEMENT	17
3.2	SPECIFIC GOALS.....	17
3.3	TASKS	19
3.4	SUMMARY	19
4	METHODOLOGY	20
4.1	RESEARCH	20
4.2	DESIGN	21
4.3	IMPLEMENTATION	22
4.4	TESTING.....	23
4.5	SUMMARY	23
5	SOFTWARE DESIGN AND DEVELOPMENT	24
5.1	MIDDLEWARE DESIGN CONCEPT	24
5.1.1	<i>Middleware Design Requirements.....</i>	<i>24</i>
5.2	SPREADXML3 DESIGN.....	25
5.2.1	<i>SpreadXML3 Design and Development Path</i>	<i>26</i>
5.2.2	<i>Object Interaction within SpreadXML3.....</i>	<i>26</i>
5.2.3	<i>Message XML Schema Description</i>	<i>27</i>
5.2.4	<i>Message Sender Component.....</i>	<i>28</i>
5.2.5	<i>Message Receiver Component.....</i>	<i>28</i>
5.2.6	<i>Message Handler Component.....</i>	<i>29</i>
5.2.7	<i>Message Wrapper Component.....</i>	<i>29</i>
5.3	ADAPTER/CLIENT MODIFICATIONS.....	29
5.3.1	<i>Adapter/Client Design Improvement Requirements</i>	<i>29</i>
5.3.2	<i>Previous Operation of Adapter/Client.....</i>	<i>30</i>
5.3.3	<i>Adapter Modifications</i>	<i>31</i>
5.3.4	<i>Impact of Adapter Modifications</i>	<i>33</i>
5.3.5	<i>Client Modification.....</i>	<i>34</i>
5.4	FFTBPLOT PLOTTING SOFTWARE MODIFICATIONS.....	36
5.4.1	<i>FFTBPlot Design Improvement Requirements.....</i>	<i>36</i>
5.4.2	<i>Previous Operation of FFTBPlot.....</i>	<i>37</i>
5.4.3	<i>Efforts to Introduce the SpreadXML3 Package.....</i>	<i>37</i>

5.4.4	<i>Design Change Decision</i>	38
5.5	NETPLOT PLOTTING SOFTWARE DESIGN	38
5.5.1	<i>NetPlot Design Requirements</i>	39
5.5.2	<i>Main Window Design</i>	39
5.5.3	<i>Plot Frame Design</i>	42
5.6	DATA LOGGER UTILITY	44
5.6.1	<i>Data Logger Design Requirements</i>	44
5.6.2	<i>Data Logger User Interface Design</i>	44
5.6.3	<i>Data Logger Operation Details</i>	46
5.7	TESTING AND EVALUATION DESIGN	47
5.7.1	<i>TestGUI Design</i>	47
5.7.2	<i>TestData Design</i>	49
5.8	SUMMARY	49
6	IMPLEMENTATION	50
6.1	JAVA DEVELOPMENT	50
6.2	UNCOMMON IMPLEMENTATION TECHNIQUES	50
6.2.1	<i>Message Handlers and Message Receiver</i>	50
6.2.2	<i>Multi-threading and Synchronization</i>	51
6.2.3	<i>Array Lists and Iterators</i>	51
6.3	MIDDLEWARE	52
6.4	XML PROCESSING.....	52
6.5	PLOTTING ENGINES.....	53
6.6	JAVA CODING STANDARDS	53
6.6.1	<i>JavaDoc Documentation Standard</i>	53
6.7	SUMMARY	53
7	TESTING AND EVALUATION.....	54
7.1	MIDDLEWARE	54
7.2	ADAPTER/CLIENT	55
7.3	FFTBLOT.....	56
7.4	NETPLOT	56
7.5	DATA LOGGER.....	57
7.6	SUMMARY	57
8	CONCLUSIONS.....	58
8.1	ACCOMPLISHMENT OF OUR GOALS AND OBJECTIVES	58
8.2	RECOMMENDATIONS.....	58
8.2.1	<i>SpreadXML3 Recommendations</i>	58
8.2.2	<i>NetPlot Recommendations</i>	59
8.2.3	<i>DataLogger Recommendations</i>	60
8.2.4	<i>General Recommendations</i>	61
8.3	SUMMARY	62
9	REFERENCES.....	63
	APPENDIX A: WPI 2004 SOFTWARE USER MANUAL	65
	APPENDIX B: XML SCHEMAS.....	74
	APPENDIX C: SAMPLE XML CONFIGURATION FILES	77

List of Figures

FIGURE 1: LANDSAT-7 LEADING E0-1 IN ORBIT ¹⁸	3
FIGURE 2: EXAMPLE OF FORMATION FLYING ¹²	4
FIGURE 3: FFTB SYSTEM ARCHITECTURE ¹²	6
FIGURE 4: DECENTRALIZED CONTROL IMPLEMENTATION ¹²	8
FIGURE 5: HIGH-LEVEL DIAGRAM OF THE 2003 TEAM'S SOFTWARE IN THE FFTB ³	9
FIGURE 6: LOW-LEVEL DIAGRAM OF ADAPTER-CLIENT INTERFACE ³	11
FIGURE 7: THE GPS CONSTELLATION OF REFERENCE SATELLITES ²²	12
FIGURE 8: DESCRIPTION OF HOW GPS DETERMINES LOCATION ²⁶	13
FIGURE 9: BINDING AN XML SCHEMA ²⁰	15
FIGURE 10: UNMARSHALLING AN XML DOCUMENT ²⁰	15
FIGURE 11: MARSHALLING TO AN XML DOCUMENT ²⁰	15
FIGURE 12: MIDDLEWARE DESIGN CONCEPT	25
FIGURE 13: STRUCTURE OF THE MESSAGE CONFIGURATION FILE	27
FIGURE 14: ADAPTER/CLIENT COMMUNICATIONS DIAGRAM ³	30
FIGURE 15: LOW-LEVEL ADAPTER FUNCTIONAL DIAGRAM ³	31
FIGURE 16: ADAPTER WITH MODIFIED COMMUNICATIONS	32
FIGURE 17: SCREENSHOT OF ADAPTER	33
FIGURE 18: NEW NETWORK COMMUNICATIONS MODEL	34
FIGURE 19: SCREENSHOT OF CLIENT	35
FIGURE 20: SCREENSHOT OF A F40 MESSAGE DISPLAY WINDOW	36
FIGURE 21: SCREENSHOT OF THE NETPLOT MAIN WINDOW	40
FIGURE 22: STRUCTURE OF THE NETPLOT CONFIGURATION FILE	42
FIGURE 23: SCREENSHOT OF A PLOT FRAME WINDOW	43
FIGURE 24: SCREENSHOT OF THE PLOT FORMAT WINDOW	43
FIGURE 25: SCREENSHOT OF DATA LOGGER	45
FIGURE 26: STRUCTURE OF THE DATA LOGGER CONFIGURATION FILE	47
FIGURE 27: SCREENSHOT OF TESTGUI	48
FIGURE 28: SCREENSHOT OF TESTDATA	49
FIGURE 29: MESSAGE TIME TEST GRAPH	55

List of Tables

TABLE 1: SOFTWARE AND PROGRAMMING TOOLS	20
---	----

1 Introduction

Our senior design project relates to software development for the Formation Flying Test Bed (FFTB) at the National Aeronautics and Space Administration's (NASA) Goddard Space Flight Center (GSFC). The FFTB software that we worked on consisted of a simulation plotting package and data communication applications to improve the data analysis and functionality of the test bed.

Background

A previous project team from WPI assisted the GSFC, Guidance Navigation and Control Center (GNCC) by developing software components of the FFTB. This previous team created an adapter and a client program (software) that controlled data flow between GPS receivers, flight software and test engineers. The adapter software combined the functionality of the FFTB that was originally divided over several programs, while the client program interfaced with the adapter and implemented a Graphical User Interface (GUI) to control the adapter.

We studied several background areas to understand our project and learn essential material to complete the project. The background topics, which are detailed in Chapter 2, included formation flying, formation flying test beds, the global positioning system, and numerous software resources.

Problem Statement

Our senior design project consisted of enhancing existing software tools that provide data plotting and communication for the FFTB. The existing software tools consist of a simulation plotting package and the previous project teams' contribution to the FFTB. The enhancements necessary for the simulation plotting package included improving the GUI and data acquisition components and implementing improved data representation capability.

Summary

Our project work benefited the engineers at the GSFC in many ways including enhancements to processing and analyzing data from simulations and starting the first steps towards employing a middleware to handle data communication within the FFTB. The improvements made to the plotting package allow for real time plotting of simulation data as well as offer easy extendibility for future applications. Overall, our plotting software assists the engineers by making the plotting of simulation data simple, efficient and functional. All of the specifics to how we accomplished our goals and objectives are detailed in subsequent chapters.

2 Background

This section of the report provides background information for readers that are unfamiliar with NASA, Formation Flying, Formation Flying Test Beds, Global Positioning System, and the Java programming language.

2.1 NASA

The National Aeronautics and Space Administration (NASA) was formed on October 1, 1958, combining the National Advisory Committee for Aeronautics and several other government organizations in response to the Sputnik technology crisis. Since then, NASA missions and projects have furthered human knowledge of the earth and space and stimulated various technological and scientific achievements relating to aeronautics and space exploration. Also, its accomplishments demonstrate that human ingenuity can achieve many feats once thought insurmountable.

On May 1, 1959 in Greenbelt, Maryland, the Goddard Space Flight Center (GSFC) became the first space flight center established by NASA. Its name memorializes Dr. Robert Goddard and his contributions in the pioneering of modern rocketry ⁶. The center is also the largest organization of scientists and engineers dedicated to learning and sharing their knowledge of the Earth, solar system, and Universe⁶. As an organization, the GSFC upholds the following mission statement:

THE MISSION of the Goddard Space Flight Center is to expand knowledge of the Earth and its environment, the solar system and the universe through observations from space. To assure that our nation maintains leadership in this endeavor, we are committed to excellence in scientific investigation, in the development and operation of space systems and in the advancement of essential technologies ².

In order to pursue this mission, extensive research is conducted relating to Earth and Space sciences through measurements from space along with suborbital, ground-based, laboratory, and theoretical investigations. Along with obtaining data, the Center operates several flight missions, such as operating spaceflight tracking and data acquisition networks. GSFC also develops and maintains information systems to keep data records for further analysis, archiving, and distribution.

In order to ensure the success of their mission, another Goddard focus is to develop new technology and instruments. The Goddard Technology Management Office (GTMO) is responsible for the technology programs for Goddard's Space and Earth Science missions. This branch of the GSFC focuses on five areas: Distributed Space Systems, Flight and Science Information Systems, High Sensitivity Detector Systems, Large Aperture and Interferometric Systems and Sub-Orbital Launch Platforms¹⁵. In addition to GTMO, there are many other centers based at GSFC that focus on other specifics relating to Earth and Space sciences.

As part of the Goddard Space Flight Center team, the Guidance and Navigation Control Center (GNCC) manages software development for guidance and navigation control applications ¹⁴. The GNCC researches one of the main technologies, Distributed Space Systems (DSS), which focuses on developing new sensing strategies and implementing system-wide techniques through the use of multiple space platforms ¹⁴.

The GNCC runs the Formation Flying Test Bed (FFTB), a testing platform for new formation flying technologies. The FFTB provides a simulated network complete with Global Positioning Systems (GPS) receivers, flight software and environmental simulation to test control algorithms for formation flight missions.

The concept behind formation flying technology is flying several satellites in formation to create a single large sensor, therefore increasing the resolution of images taken by the satellites. As a relatively new concept, formation flying would cost much less than upgrading a single satellite to acquire the same level of accuracy and resolution.

2.2 Formation Flying

Formation flying refers to multiple spacecraft flying together to achieve a common goal. This concept has been studied for application to the coordination of multiple robots, unmanned air vehicles (UAVs), autonomous underwater vehicles (AUVs), satellites, aircraft and spacecraft⁹. Utilizing this idea for satellites opens up many capabilities and applications for NASA to use in space. The first autonomous formation flying earth science mission conducted by NASA began in 1996, and successfully launched the Earth Orbiter-1 (EO-1) satellite in 2000 as a technology mission designed to fly in formation with another NASA satellite called Landsat-7¹⁸. Figure 1 shows Landsat-7 leading the EO-1 in orbit. Before the EO-1 mission, formation flying technology posed a high risk since it was a new paradigm but recently has gained the interest of earth and space scientists worldwide and marks a milestone on the way to autonomous, multi-spacecraft, formation flying. This is especially true at NASA where approximately 35 missions have been proposed using formation flying technology¹².

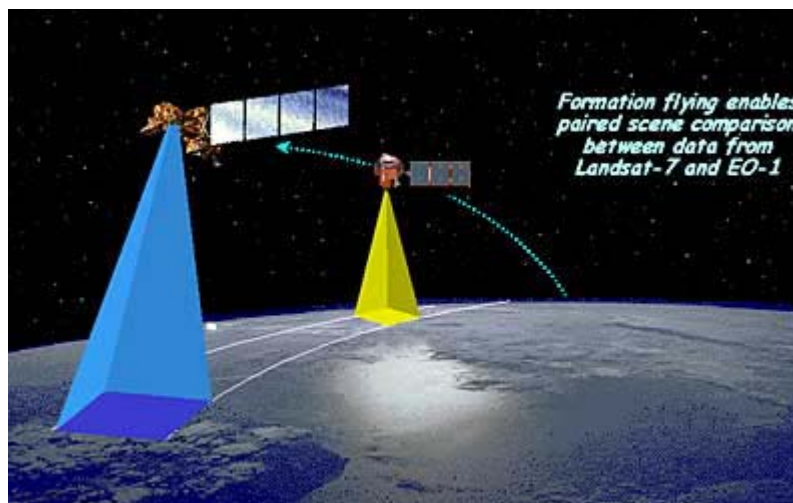


Figure 1: Landsat-7 leading EO-1 in orbit¹⁸

The idea of formation flying missions is to use several small satellites orbiting in close proximity to complete a mission, instead of a single larger satellite. Using distributed formations, as shown in Figure 2, provides great potential for space science to perform observations that require high resolution imaging. There are several other benefits to using multiple formation flying vehicles such as reduction in cost, improved accuracy and robustness, and increased mission performance²⁴. There are many factors that affect the cost of using formation flight missions rather than single satellites, including easier construction

of smaller and lighter satellites, cheaper to launch, and less expensive distributed sensors. The cluster is also less sensitive to failure due to the malfunction of one satellite. If one satellite has problems, the others can maintain their mission until the problems are resolved. The formation flight technology creates a large increase in mission performance through “distributed computing, sensing, data acquisition and data transmission of the satellite formation”²¹. Overall this technology addresses NASA’s mission statement to advance the essential technology by building satellites that are better, faster and cheaper ².

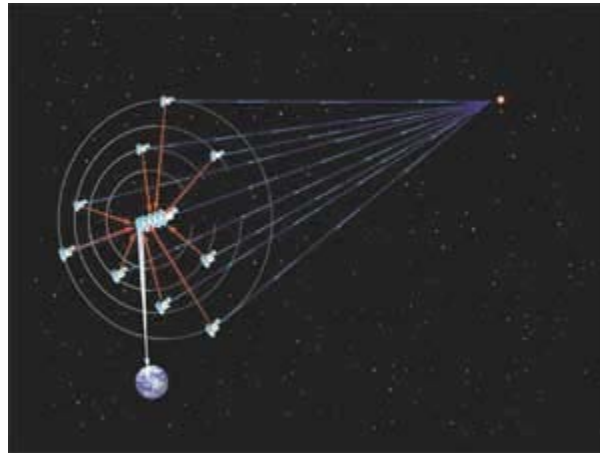


Figure 2: Example of formation flying ¹²

As explained in *Distributed Coordination and Control of Formation Flying Spacecraft*, “the goal of spacecraft formation flying is to maintain a fleet of vehicles in a desired relative geometry over extended periods of time while minimizing fuel cost”²⁴. Accurately controlling the spacecraft requires the selection of a common reference point for the fleet. There are three different methods used in autonomous formation flights to coordinate the flight patterns: reference orbit, leader-follower, and virtual center. All of these methods use a reference point to determine the position of each spacecraft in formation.

The first and simplest method uses a reference orbit, a point in space that describes the average fleet orbit, as the reference point. The center of the formation is used as the reference point, which is described by nonlinear orbit equations. These equations neglect to model any disturbances that may be encountered during a mission that would cause the fleet to go off track ²⁴.

The next method specifies the reference point as the *leader* of the formation and the other satellites as *followers*. This is advantageous over the reference orbit method such that the reference point is on a spacecraft and any disturbances would be recorded. Since the *leaders’* state doesn’t experience error it will use minimal fuel while the *followers’* may need to use excess fuel to maintain formation. Fuel management is a disadvantage to this method because the *leader* doesn’t represent the average fleet motion and *followers* may experience larger disturbances than others ²⁴.

The final method uses a *virtual center* as the reference point estimated using measurements from all spacecraft in the fleet. This approach signifies the average motion of the fleet, including an average of disturbances by coordinating the movement of the formation and individual spacecraft. Advantages of this control method are analyzing the behavior of the formation as a group and potentially lowering fuel costs compared to the other methods. A disadvantage to using a *virtual center* is that the centralized control creates a possible single point of failure and requires increased communication by the fleet ²⁸.

2.3 Formation Flying Test Bed

For the most part, formation flying technology remains a new concept that is still being developed and tested. In order to determine the application of the technology, several testing facilities have been created at universities, such as Stanford, MIT, and Boston College to test related hardware and software techniques. Through the simulations and experiments conducted at these facilities, called Formation Flying Test Beds (FFTB), the risk associated with formation flight missions can be reduced.

Currently MIT has developed a hardware-in-the-loop test bed for developing and testing estimation and control architectures for formation flying spacecraft. This test bed is comprised of computers linked together over Ethernet, where each computer represents one spacecraft in the fleet. MIT also has a truck test bed consisting of three trucks previously used to test carrier phase differential GPS sensing techniques⁸. The plan is to demonstrate the formation flying software using the three trucks as an autonomous fleet of vehicles.

2.3.1 Description of NASA FFTB

NASA realizes the potential of formation flight for many science applications and has created their own FFTB at the GSFC. The NASA FFTB contains a network of computers that simulate space, flight, and GPS receivers. The following information from an article in *GPS World* presents a detailed description of the NASA FFTB system architecture and decentralized control implementation¹².

Formation Flying Test Bed

A simulation or emulation testbed must support the widest range of relevant programs. The diagram in Figure 3 defines the system architecture in its most general form, without respect to whether centralized or decentralized navigation or control are employed. Its components include:

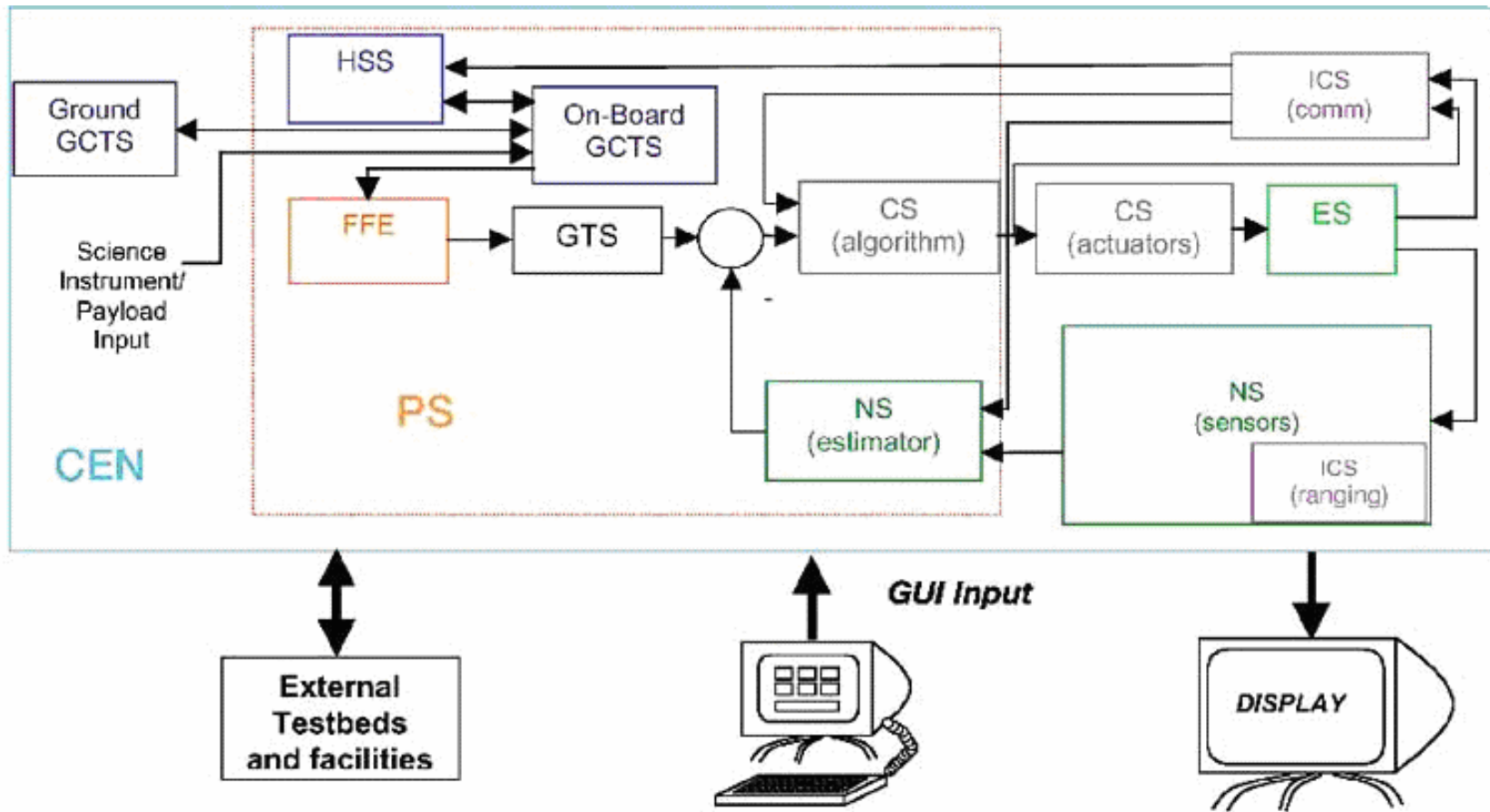


Figure 3: FFTB system architecture ¹²

- Central Simulation Controller (CEN), software and hardware infrastructure monitoring all subsystem activities, composed of spacecraft and laboratory formation components
- Intersatellite Communications Subsystem (ICS), hardware and software related to satellite crosslinks for communication and ranging, and onboard intersatellite command and data handling of the formation
- Onboard Processing Subsystem (PS)
- Ground Control and Telemetry Subsystem (GCTS), an onboard transition of traditional ground-based operations functions
- Navigation Subsystem (NS), sensors, GPS receivers and antennas, and so on, models and simulations thereof, and software (lying entirely on the PS) providing estimated state vectors
- Guidance and Trajectory System (GTS), holding and computing maneuver plans for individual satellites
- Environment Subsystem (ES), models of spacecraft dynamics (orbit, attitude, possibly flexible modes) and space environment, including the ionosphere and atmosphere, and any sensor noise models and motion of GPS satellites, where applicable
- Vehicle Control Subsystem (CS), hardware (actuation devices), models and simulations thereof, and software computing forces and moments applied to each individual vehicle
- Health and Status Subsystem (HSS), onboard processing to track system mnemonics, look for out-of-range variables, perform fault-detection, and perform higher-level vehicle control
- Formation Flying Executive (FFE), the connectivity between the desired motion of the virtual platform and that of individual satellites. This onboard software component will take a high-level virtual platform guidance command and allocate guidance to each individual vehicle. In the simplest form, it defines the initial conditions of each satellite based on a desired cluster configuration.

This guideline for the FFTB ensures general capability to integrate a component or set of components into the testbed for integrated system analysis, and the ability to solve required customer problems while simultaneously building infrastructure and capability to support vastly differing formation flying projects and programs.

Decentralized Control

One of the first FFTB projects, a decentralized formation control algorithm development, applies linear quadratic Gaussian (LQG) control theory implemented over a distributed cluster of vehicles. This project was completed in late 2001, and it has produced a high-fidelity GPS-based implementation of decentralized formation control in the formation flying testbed.

Figure 4 shows the specific implementation of the integrated closed-loop analysis, with key elements:

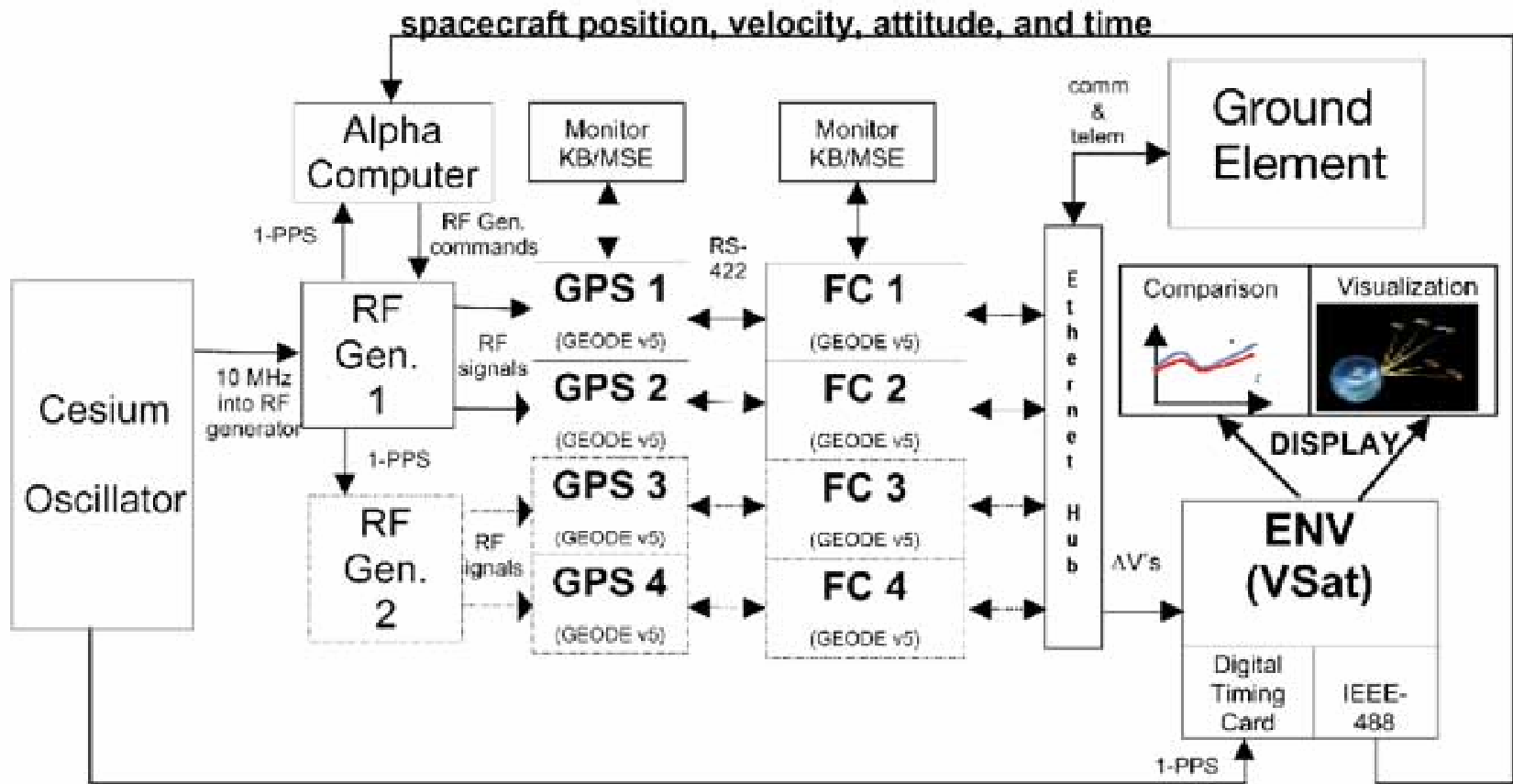


Figure 4: Decentralized control implementation ¹²

- An environment simulation of two to four vehicle orbital dynamic models running in a software tool called VirtualSat (VSat) Pro
- A dual frequency, four RF-output (each with 16 channels) GPS RF signal simulator, driven through an IEEE 488 interface from the VSat Pro model
- Two to four open-architecture PiVoT GPS receivers, each running a GEODE extended Kalman filter producing absolute navigation solutions
- Two to four rack-mounted PCs, communicating with each other through an ethernet hub, each running formation command (FC) and data handling, a relative navigation version of GEODE, and the decentralized control law
- A Cesium timing source to ensure proper synchronization.¹²

2.3.2 Previous WPI Contribution to NASA FFTB

Prior to our project, the 2003 project team from WPI assisted the GNCC project by developing software components of the FFTB. They created an adapter and a client program that controlled data flow between Global Positioning System (GPS) receivers, flight software and test engineers. Their adapter software combined the functionality of the FFTB that was originally divided over several programs, while their client program interfaced with the adapter and provided a Graphical User Interface (GUI) for the test engineers. Their software, which was based on Java technology, facilitated the use of the FFTB for running simulations and logging and presenting data. Figure 5 shows how the product of the previous team's work fit into the FFTB³.

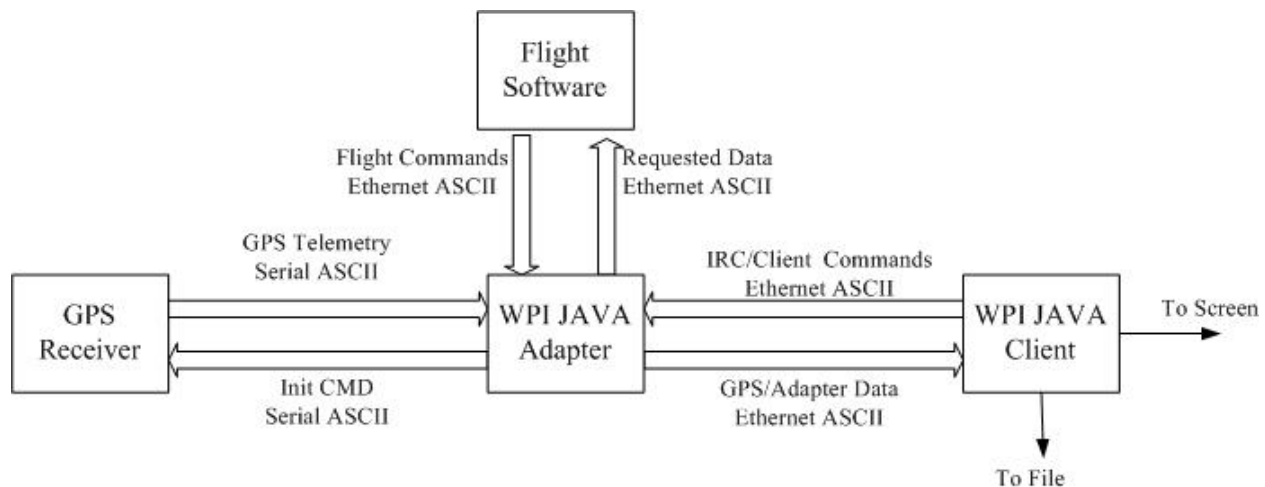


Figure 5: High-level diagram of the 2003 team's software in the FFTB³

Component Descriptions: Adapter

The functionality of the 2003 team adapter program can be divided into three tasks: i) communicate with the GPS receiver, ii) communicate with the Flight Software, and iii) communicate with the remote Client program. The adapter's interface with the GPS receiver consists of initializing and commanding the GPS receivers, and passing the information from the GPS receivers to the Flight Software and Client program. The adapter's interface with the Flight Software consists of processing data request commands from the Flight Software and responding by transmitting the requested information. The adapter's interface with the Client program consists of accepting GPS initialization commands from

the client, reformatting and transmitting these commands to the GPS receivers, and transmitting GPS telemetry information back to the client³.

Component Descriptions: Client

The functionality of the client program includes: initializing the GPS receivers, commanding the adapter to transmit GPS data, archiving the GPS receiver data, and displaying the GPS data. Initializing the GPS receivers and commanding the adapter are performed through the adapter-client interface as described in the above paragraph. Telemetry data from the GPS receivers are archived into a file by the client program. The data is also displayed in real-time in the client's GUI, which was designed to be simple and straightforward. Figure 6, which also is from the previous team's project report, shows the interfaces between the adapter, client, GPS receivers and flight software in a low-level functional diagram³.

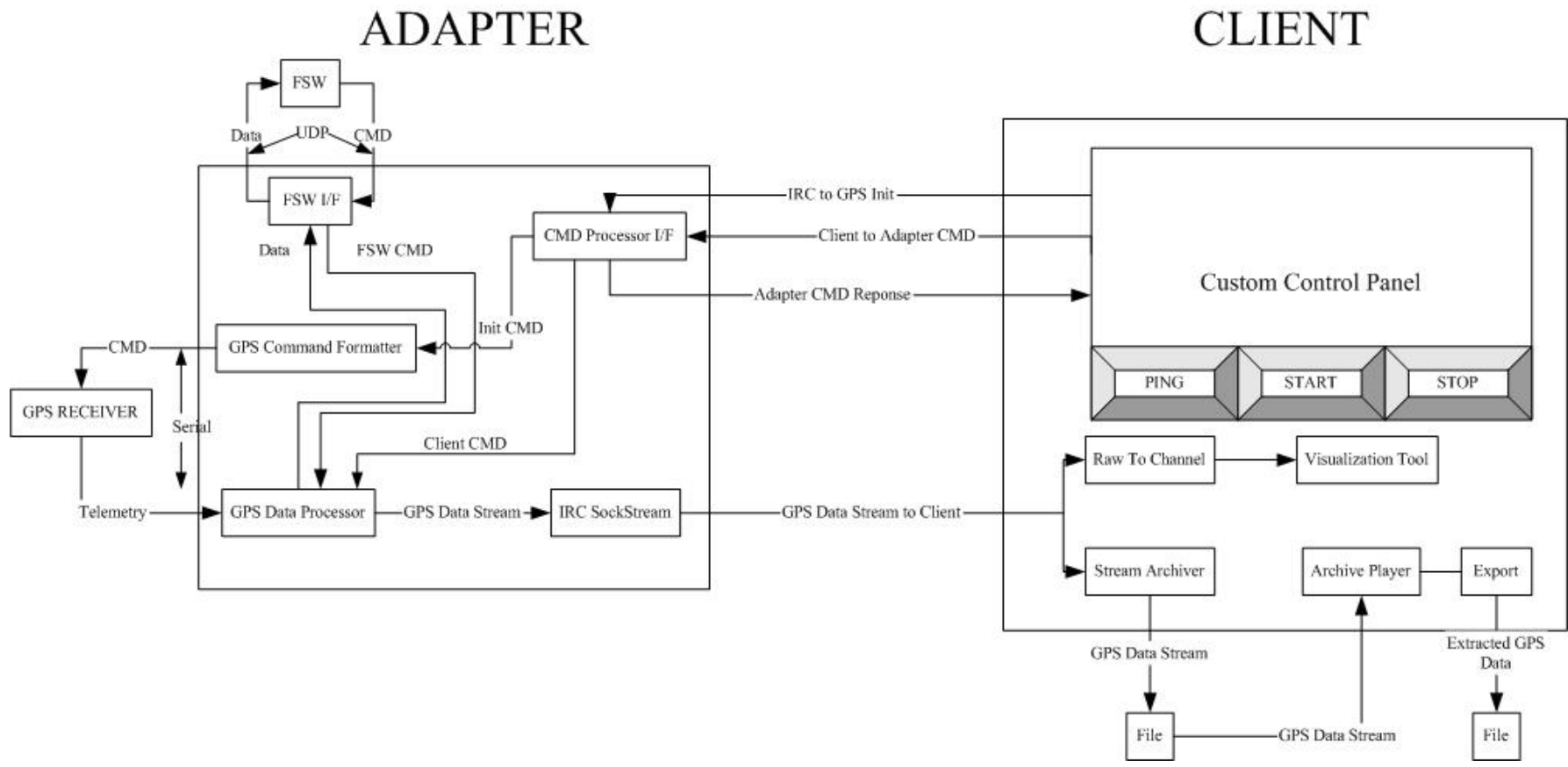


Figure 6: Low-level diagram of adapter-client interface ³

2.4 The Global Positioning System

The global positioning system (GPS) is a modern technological resource that has revolutionized navigation. The GPS consists of a constellation of 24 geo-synchronous reference satellites and covers the entire globe. From anywhere in the world, information such as coordinate location, altitude, velocity, and time can be determined with a GPS receiver unit. As GPS technology advances, the measurement accuracy has increased and the cost and complexity of GPS receiver units has decreased, making GPS accessible and viable for a wide range of applications. Current GPS positioning technology is precise within one centimeter and GPS receiver units have become small and relatively inexpensive, allowing their use in cars, boats, planes, construction equipment, movie production gear, and many other applications that require precise positioning. This technology has also been utilized by satellites in low earth orbits, and is employed by formation flying technology²⁵. The diagram in Figure 7 shows the constellation of reference satellites in the GPS.

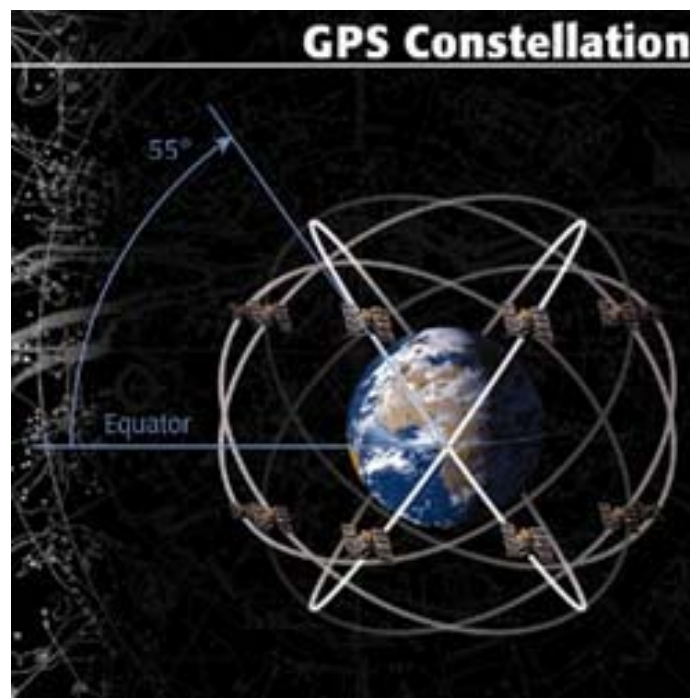


Figure 7: The GPS constellation of reference satellites²²

Accomplishing precise position measurements with GPS technology relies on very sophisticated systems but can be easily accessed through a relatively simple GPS receiver. The 24 GPS reference satellites are in extremely precise geo-synchronous orbits and can be considered fixed points in relation to the earth. A GPS receiver can determine its position (both coordinate location and altitude) through a method called “triangulation”. The GPS receiver measures its relative distance from four reference satellites by measuring the travel time of radio signals between itself and each satellite. From these travel times, distances can be computed and then geometry applied to determine the receiver’s relative position to the “fixed point” satellites. Coordinate location and altitude information is translated from the relative position. Taking multiple position measurements and comparing change in position to change in time is performed to compute velocity. Also, atomic-accuracy time measurements can be acquired from signals

sent by the satellites, which each have their own atomic clock²⁵. The diagram in Figure 8 summarizes the process of determining location with GPS.

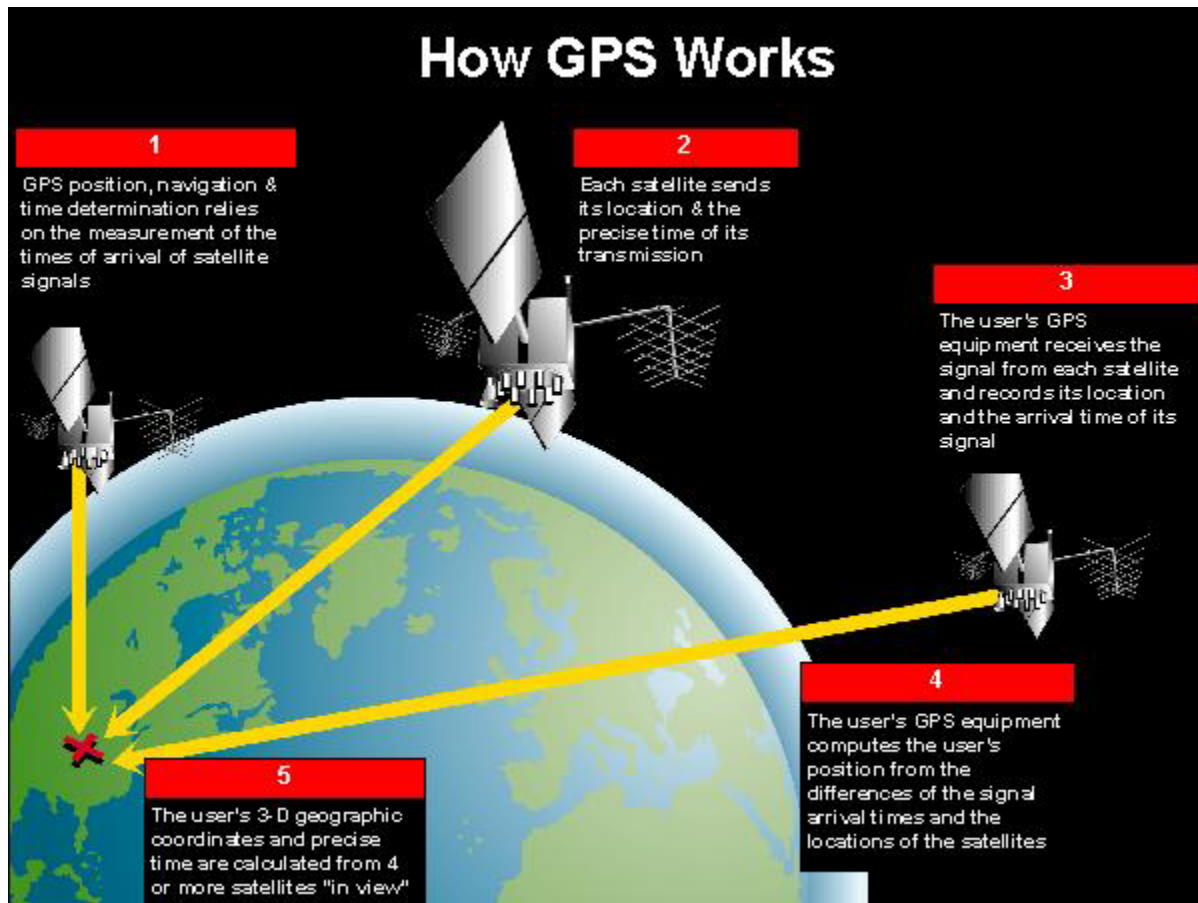


Figure 8: Description of how GPS determines location²⁶

2.5 Java

Java technology was commercially released by Sun Microsystems in 1995¹¹. Java consists of the Java Platform and the Java programming language. The concept of Java was to enable uncomplicated cross-platform applications. A Java application will operate universally on any machine that has a compatible Java Platform, known as the Java Virtual Machine. This allows computers and devices of various types to communicate and work with each other. Although Java is most commonly known for its use on personal computers and web-based applications (called applets), it can also be used on a wide variety of electronic devices, such as televisions, cell phones, appliances, and much more¹¹.

Java is also an object-oriented programming language. The concept behind object-oriented programming is that everything within a program can be represented as an "object". Objects are separated into different "classes", where objects of the same class have the same characteristics and "methods" that they can perform on data. Through hierarchical class definitions, a class can inherit the characteristics and methods of its "super-class" and pass on its characteristics and methods to its "sub-classes". The methods of an object are the private/protected and public operations that an object can perform on data. The private/protected methods are internal and can only be accessed from inside that object. The public

methods of an object are the methods that another object can perform on it through messages known as “public procedure calls”. The list of the public procedure calls of an object is known as the object’s “interface”⁷.

2.6 Extensible Markup Language (XML)

The extensible markup language, or XML, is a generic text-based markup language similar to the hypertext markup language (HTML). XML was first developed in 1996 by an XML Working Group of the World Wide Web Consortium (W3C)²⁷. The working group for XML expressed several design goals for the language, which are listed verbatim below:

1. XML shall be straightforwardly usable over the Internet
2. XML shall support a wide variety of applications
3. XML shall be compatible with the Standardized General Markup Language (SGML)
4. It shall be easy to write programs which process XML documents
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero
6. XML documents should be human-legible and reasonably clear
7. The XML design should be prepared quickly
8. The design of XML shall be formal and concise
9. XML documents shall be easy to create
10. Terseness in XML markup is of minimal importance²⁷

This markup language, unlike HTML, is used primarily for transmitting structured data that is identified by tags (term enclosed in brackets <...>), which are defined by the user⁴. One characteristic to recognize XML is that all XML files must declare a prolog at the beginning of the document. A powerful tool used in XML is the ability to create a schema that specifies the hierarchal structure and the data that is expected within an XML file. The schema also identifies the elements in the document, the order they must appear, any associated attributes, and the relationship between different components²⁰. By following a schema other users have the ability to check the validity of data within an XML document.

Now that XML has become a standard for exchanging data across systems there have been several Java technologies developed for processing XML files with applications written in Java. The different Java technologies relating to XML are Java Architecture for XML Binding (JAXB), Java API for XML-based RPC (JAX-RPC), Java API for XML Processing (JAXP), Java API for XML Registries (JAXR), and SOAP with Attachments API for Java (SAAJ)²⁰. The library that is most applicable towards our project is the JAXB, which allows access to XML documents from Java applications. A notable benefit of the JAXB libraries is the ability to incorporate XML data and processing functions, through binding a schema, into applications written in Java without knowing much about XML²⁰.

In order to access an XML document within a Java application the document must be presented to the application on Java format. To accomplish this task the JAXB libraries are needed to bind the schema that identifies the document. Binding the schema creates a set of Java classes that represents the schema. Figure 9 illustrates the process of binding a schema. The binding compiler provided in the JAXB libraries generates a set of Java interfaces as well as a set of Java classes that implement the interfaces²⁰. The auto-generated classes allow the user to retrieve and assign the data for each element within the schema.



Figure 9: Binding an XML schema ²⁰

Another tool provided when implementing the JAXB library is an unmarshaller. The process of unmarshalling an XML document, shown in Figure 10, results in the creation of a tree of content objects, which are instances of the classes generated by the binding compiler. The content objects represent the content and organization of the document ²⁰. After unmarshalling the document, the schema-derived classes can be utilized within the Java application to access the data held in the document.

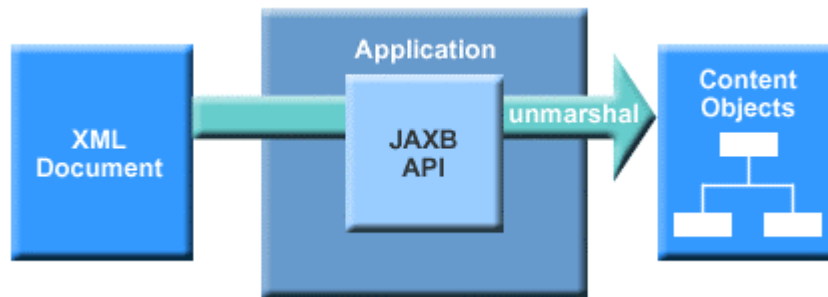


Figure 10: Unmarshalling an XML document ²⁰

Contrary to unmarshalling is the process of marshalling, represented in Figure 11, which creates an XML document from a tree of content objects ²⁰. The tools to perform this operation are also supplied in the JAXB libraries. In the Java application the user sets the data of the content tree using the schema-derived classes and this data is then stored in the created XML document through the marshalling process.

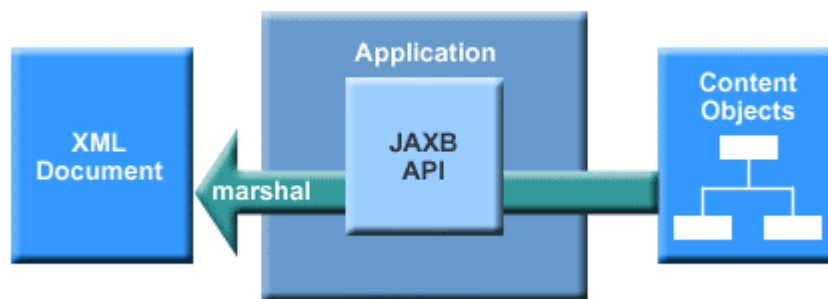


Figure 11: Marshalling to an XML document ²⁰

In general, XML formatting is beneficial to share data in a common and clear manner. The design goals followed by the developers of XML assisted in making the language a standardized text format for transmitting structured data. Also, the technologies designed for accessing XML documents within Java applications provide many functional advantages to the software developer such as defining schemas, binding schemas, unmarshalling an XML document, marshalling to an XML document, and many other tools provided in the JAXB libraries.

2.7 Summary

In our background section we presented various topics including formation flying, formation flying test beds, the Global Positioning System, the Java programming language, and the extensible markup language (XML). The formation flying test bed section described both the operation of the test bed at the GSFC and the contribution of last year's project team to the test bed. In the following section we presented a detailed description of our project goals and objectives.

3 Project Statement

In the autumn of 2003, a design team from WPI assisted the Guidance and Navigational Control Center (GNCC) at the Goddard Space Flight Center (GSFC) by developing software components of the Formation Flying Test Bed (FFTB). Their software facilitated the use of the FFTB for running simulations and provided a useful interface for presenting and logging data. At the completion of that project it was evident that several improvements could be made including graphical representation of the data and a standard form of communication.

3.1 Goal Statement

The overall goal of our project was to enhance existing FFTB simulation support tools that provide real-time plotting and communications for FFTB test data. Within the context of this overall goal, we identified specific goals, as well as numerous objectives that we needed to complete to meet our goal. Below, our specific goals and their related objectives are described in detail.

3.2 Specific Goals

Our project can be broken down into two specific goals that are central to completing the overall goal. First, improve the existing simulation plotting package, FFTBPlot. This objective was important because it allows test bed engineers to plot and view simulation data. Enhancement of this software included modifying the data acquisition components and the GUI to improve the range of data sources that the software supports. Originally, the only method of real-time data acquisition was to continuously read a file, which was being updated by another piece of software that was constantly writing data to that file. This process was near real-time, but has several shortcomings such as constant file system access, which causes inefficient data acquisition and the difficulty of extendibility and implementation across a network

Second, improving the data communications between FFTB components was crucial to providing a simple, efficient means for transmitting data within the test bed. To implement this goal we had to change the current point-to-point connections to utilize the middleware. For each goal, we also identified numerous objectives that we needed to accomplish to meet each goal. Below, we summarize those objectives, organized by goal.

- 1) Improve the existing simulation plotting package, FFTBPlot
 - a) Add the capability of data acquisition from multiple sources, such as network,
 - b) Enable users to better customize of the appearance of the data representation,
 - c) Update the GUI to reflect any new features,
 - d) Improve GUI functionality, flow, ease of use, extensibility, aesthetics,
 - e) Resolve bugs in the current plotting package,
 - f) Clean up and document the code,
 - g) Add features for data manipulation,

- h) Improve threading and efficiency of code
- 2) Improve the data communications between FFTB components
 - a) Develop a new way to communicate data over the network,
 - b) Use XML formatting on the data passed over the network

The simulation support tool that we enhanced was the simulation plotting package. The plotting package is called FFTBPlot and was authored by a member of the FFTB software development team. FFTBPlot processes a file, which is updated continuously by another program, and plots the data that it obtains in near real-time. This application creates strip charts based on the data using JFreeChart, a free Java based data plotting library. The user interface and data acquisition components are implemented with Swing, a Java graphical user interface (GUI) library package. The software also employs the Java API for XML Binding (JAXB) libraries, part of the Java Web Services Developer Pack (JWSDP), to enable the ability to save configuration options in plaintext files formatted with Extensible Markup Language (XML).

The solution and an objective of our first goal for the FFTBPlot software was to enable direct communication between FFTBPlot and the data-generating source through a network connection. In order to enable network communication we utilized a software package called Spread, which is an open-source Java library for handling network communications. This objective was also extended to become our second goal of using the Spread network communications for other FFTB components. Along with enabling the ability to receive data over the network, the ability to receive an XML-formatted plaintext file was also an important objective.

Another objective was to update the GUI to reflect any new features that we implement in the FFTBPlot software. Along with updating the GUI, other modifications to the GUI are necessary such as functionality, ease of use, and extensibility. The GUI can be modified to include support for user customization of the appearance of the data plots. Functionality improvements would be the addition of support for performing calculations and processing on data sets to create new data sets.

Finally, there were many other objectives that dealt with the FFTBPlot software that retain a lower priority than improving data acquisition and updating the GUI. These objectives included resolving bugs, cleaning up the code, and creating documentation to allow for the ease of future development. Another feature to benefit the test engineers was to allow manipulation of the data from within the FFTBPlot software. Improvements to the FFTBPlot software such as improving the threading, efficiency of the code, and aesthetics of the GUI are also minor objectives that were dealt with after the main objectives were accomplished.

An alternative that we investigated as we worked with the FFTBPlot software was to abandon this implementation and start over. The reason for abandoning FFTBPlot was due to the various issues that we addressed with FFTBPlot:

- 1) FFTBPlot had a very complicated class hierarchy
- 2) FFTBPlot used JFreeChart which is inefficient and not designed for live plotting
- 3) FFTBPlot had an unnecessarily complicated GUI
- 4) FFTBPlot had numerous GUI and plotting bugs that needed to be resolved

All of these problems made it easier to start over rather than trying to modify the existing FFTBPlot. Therefore, this became a major design decision that we had to make during the course of our project. The details for this design change are discussed in Chapter 5 Section 4.

3.3 Tasks

In order to accomplish the goals of our project we had to complete several tasks. These tasks ranged from becoming familiar with the software that we modified to the programming environments and tools that we used. It was also necessary for us to refresh our knowledge of programming with Java and learn about some advanced topics in the language, such as GUI development, multi-threading, file input/output (IO), and data stream IO, network communications, and XML parsing and manipulation.

Below is a summarized list of the tasks necessary to accomplish the goals of our project:

- 1) Refresh our knowledge of Java
- 2) Learn advanced topics in Java
 - a) GUI development
 - b) Multi-threading
 - c) File IO
 - d) Data stream IO
 - e) Network Communications
 - f) Communications using XML formatted text
- 3) Become familiarized with Java development tools
 - a) Eclipse, Java Integrated Development Environment (IDE)
 - b) SmartCVS, Java development tool
 - c) JFreeChart, Java data plotting library
 - d) Spread, Java network communications library
 - e) JAXB, Java libraries for XML manipulation
 - f) Other Java libraries that we may decide to use
- 4) Document source code and provide useful manuals
- 5) Develop testing and verification procedures for the software

3.4 Summary

This section has presented an overview of the goals, objectives, and tasks related to our senior design project. The goals dealt with modifying and developing software components of the FFTB at the GSFC, more specifically the simulation plotting package and component data communications. Each goal had several related objectives to break the goal into several steps. To accomplish the goals and objectives we also had several tasks that we completed such as learning advanced topics in Java, becoming familiar with Java development tools, documenting source code, and testing and verifying our software. In the following chapter we will discuss the methods to complete our project.

4 Methodology

In this chapter we discuss the many steps necessary for completing our goals and objectives. To begin our project we discussed with our mentors the specific needs of the test engineers working with the FFTB to narrow the focus of our goals. Then we studied background areas that included formation flying, formation flying test beds, the Global Positioning System (GPS), and various software and programming topics such as Java, Eclipse, the Spread Toolkit, and XML, to become familiar with the application of our project. By studying the background information we gained the intellect to accomplish our goals as well as understand the “big picture” of our project. After researching the essential information we began examining our goals in further detail and decided on a strategy to achieve our goals. Each goal had several objectives that we prioritized in order to complete the goal in a logical manner. The main goals also required several stages of implementation and testing to prove the proper functionality and determine possible changes. Overall, we took a step-by-step approach to fulfill our goals and objectives in an orderly and timely fashion.

4.1 Research

Once we obtained the project description from our mentors we identified topics relating to background information that we had to become familiar with to complete the project. These topics included formation flying, formation flying test beds and the Global Positioning System. In order to obtain primary and secondary sources of information for these topics we utilized resources from NASA, the Internet, previous work associated with the GSFC FFTB and the WPI library.

After establishing knowledge within these topic areas we began looking at our goals and objectives in more depth and inquired with our mentors in regards to the software and programming tools that we needed to become familiar with. This led us into investigating the Java programming language, Eclipse integrated development environment (IDE), SmartCVS development tool, JFreeChart plotting library, and Spread Toolkit. Research within these areas was mainly done on the Internet by going directly to the main website for each. We also used books as an additional source of information. Table 1 below lists the software and programming tools and their corresponding website.

Software tools	Website
Java	http://java.sun.com
Eclipse IDE	http://www.eclipse.org
SmartCVS	http://www.smartcvs.com
JFreeChart	http://www.jfree.org/jfreechart
Spread Toolkit	http://www.spread.org

Table 1: Software and programming tools

The majority of our research dealt with the Java programming language. Since our project dealt with improving current software written in Java we had to review the basics and learn numerous concepts that were unfamiliar to us, such as GUI development, file and data stream input/output, multi-threading,

network communications, and interpretation of XML formatted text. A useful process to learn these new concepts was through online tutorials and the Java Application Programming Interface (API) specifications provided on the Sun Microsystems website. To learn how to use the third-party Java libraries (JFreeChart, Spread Toolkit), we used the information pages and API specifications that we found on their respective websites.

Along with learning the Java concepts, we also had to understand how to use the different Java language development tools that the engineers at the GSFC use. In the beginning we researched and examined the purpose and use of the tools to gain a general understanding. Once we obtained the software for the plotting package we began examining the code using the Eclipse IDE tool, which through experience assisted in understanding how the Eclipse environment operated. One tool that we were not too familiar with was SmartCVS, but our mentors explained its function as a source file repository, which is a source code management system. We also had to learn about JFreeChart, which is the open-source Java library used as the plotting engine within FFTBPlot. After understanding the Java language and how to use the variety of available tools we were ready to begin working towards our goals and objectives.

Our objective of network data acquisition led us into implementing custom Java libraries based on the Spread network communication libraries. Our mentors recommended the Spread Toolkit because it was open source and provided us with the essential network communications functionality in a simple, easy-to-implement manner. The ability of communicating over the network also directed us into the idea of formatting the transmitted data with Extensible Markup Language (XML). To gain an overall understanding of XML we attended a self-paced course entitled “Introduction to XML and Associated Technologies” provided by The Learning Center at the GSFC. This course was web-based and helped us to understand XML, which we previously knew very little about. From the course we obtained documentation of the topics covered that we later used as a reference. Implementing XML-formatting required us to employ another Java library called JAXB or Java Architecture for XML Binding. Our mentors supplied us with relevant tutorials to assist in understanding and utilizing JAXB in order to implement XML within our application.

4.2 Design

After we understood the essential background material, such as the formation flying test bed, the Adapter/Client and FFTBPlot software, we were ready to move forward into formulating design decisions. Since most of our design dealt with improving existing software, the Adapter/Client and FFTBPlot, we had to analyze on the function and purpose of various elements within the software before making any critical design decisions. We obtained the specifics of the software through examining the code, discussing the programs with our mentors, and creating block diagrams of how the current system behaved and how the hierarchy of objects within the program was arranged. Using the system block diagrams we were able to determine where design changes needed to be made as well as any other necessary changes. Following a basic approach, as listed below, aided in making and following through design decisions.

1. Review relevant background material
2. Understand current software (examine code and create block diagrams)

3. Determine design criteria (by examining code or discussing with mentors)
4. Determine changes that are needed to meet the design criteria
5. Implement changes
6. Test and evaluate

One of the first design decisions we undertook was figuring out how to utilize the Spread Toolkit that our mentor recommended as a library for network communication. This toolkit was the foundation of our communication classes that provided communication between various applications on separate computers within a network. Therefore when we created our communications libraries we needed to understand Spread to assure that the design would allow the communication to be reliable, robust, and extendable. These design criteria were expressed by our mentors and through examination of the current system.

The next design changes we dealt with were modifications to FFTBPlot, the simulation plotting package. To gain a clear understanding of this complicated software package, we created block diagrams of the class hierarchy within the program and how the objects interacted with one another. The block diagram and design criteria addressed by our mentors assisted in realizing the design changes to be made. In order to approach each criterion methodologically we prioritized them in terms of necessity, difficulty, and whether or not it was required by another criterion.

Our design changes and software applications also required design in order to properly test and evaluate their functionality, ease of use, and extensibility. Design for testing and evaluation dealt with creating an application with a simple GUI that allowed the ability to access and analyze the features of our various design changes. The testing design phase related to each design change after it was implemented to demonstrate whether our implementation met the design criteria focused on by the specific design change. If any problems arose while testing we then had to re-evaluate our design changes and make any necessary modifications within our implementation.

Subsequent to identifying the design criteria and defining an approach to meet the requirements, we were able to shift into implementing the changes. Later in the implementation phase we re-examined our design process to make certain we were headed in the right direction.

4.3 Implementation

The implementation phase was the next big step, after forming concrete design decisions, toward achieving our goals. This was the phase where we utilized our previous knowledge and began applying our design towards meeting the design requirements. Our decision to prioritize the objectives and design criteria proved to be helpful in creating a step-by-step approach to employ our design.

The first design decision we undertook was to demonstrate the power and functionality of the Spread Toolkit. In our first step of implementation we generated two general Java classes to utilize the Spread Toolkit that demonstrated the ability to send data over a network connection. These classes were very basic and lacked powerful functionality and extensibility but provided us with experience in working with the Spread libraries. The classes also demonstrated the potential for our application as the communications middleware for components of the FFTB. Next in our implementation process we created Java classes to utilize both Spread and JAXB libraries in order to handle the network connection

and XML formatted text. Upon verifying these classes worked properly we moved onto the next step, which was to apply the new capabilities, provided by the classes, into the communication between the Adapter/Client software. After establishing Spread as a reliable and extendable middleware, we began employing our design changes into the FFTBPlot software to allow for real-time plotting.

As we implemented our design changes in the step-by-step manner, if any problems arose we rethought our design and discussed with our mentors the problems to determine a solution. Throughout our software we made sure to document our code in a recognized standard of Java source code documentation. This facilitates the understanding and improvement of our software by future developers. An important part of demonstrating each step of implementation was the testing and evaluation that we performed on the various features of our software.

4.4 Testing

Throughout implementing our design changes we evaluated each step to verify we met the design requirements and resolved any problems before moving onto the next step. The main process for testing our Java classes was done by creating a simple application that utilized and demonstrated the features of our software. This testing application had a very plain GUI with buttons and text fields that activated various features of our software with the specified parameters. As we progressed through our implementation phases we modified the GUI to represent any new features that had been added. To evaluate the software we reverted back to our objectives and design criteria to ensure that nothing was neglected and to determine if any modifications were necessary. Each test case was demonstrated to and discussed with our mentors to inform them of our progress and so they could also make sure everything was designed to their standards and met the requirements that were expressed. By testing and evaluating each step we were able to identify errors and resolve them quicker and easier than if we waited until the final stages of our implementation to test and evaluate.

4.5 Summary

Overall, the methods we used to research, design, implement, test, and evaluate our project helped us complete our project in a logical and efficient manner. Our mentors supported us at every step of the software development and testing process and also provided assistance as problems arose and helpful recommendations to improve features within our software. In the following section we present our software design and development results.

5 Software Design and Development

The software design and development that we performed for this project required us to develop or work on several different pieces of software. As a result the software product of our project consisted of both original software that we designed and developed and previous software that we modified. However, in both cases we had to understand the problems in order to identify the software changes required to solve the problem.

We also note that when we developed new software algorithms, we used a structured, top-down design approach where we first determined the top-level operation of our program. When we modified existing software we first reverse-engineered the software to determine how it functioned and then modified it as necessary to meet the requirements.

Special Note - the software developed for this project and described in this and the following sections is included on the CD containing the final report. All routines, testing procedures, the final document(s) and all presentation material are also found on the CD.

5.1 *Middleware Design Concept*

In order to run a simulation with the FFTB, many applications on separate computers and devices need to be run. As a result communications between FFTB applications is an important issue. These separate applications need to communicate with each other in order to pass data, commands, information, etc. Implementation of the communication was previously done through various methods using point-to-point protocols (such as TCP, UDP, Sockets, etc.). This implementation was successful, however as the components change or new ones are added, the tedious task of modifying communication handling source code in several applications is usually necessary.

5.1.1 **Middleware Design Requirements**

A design concept that solves the problem of using point-to-point protocols is handling communications through a middleware. This concept of middleware is illustrated below in Figure 12. Instead of having each application manage a separate point-to-point connection for every other component that it needs to communicate with, each component would have to manage only one connection to the middleware. The middleware handles all of the data messages and redirects the messages to the designated destinations. In order to accomplish this, all messages sent to the middleware are sent to a specific “subject” (a unique name for a group of messages). In order for a component to receive messages from a specific “subject”, it only needs to “subscribe” to that specific “subject”. This allows messages to be communicated over these subjects in a scheme where data senders need only to broadcast the data and data receivers need only to listen to the “subject” (or channel) of data that they need to receive.

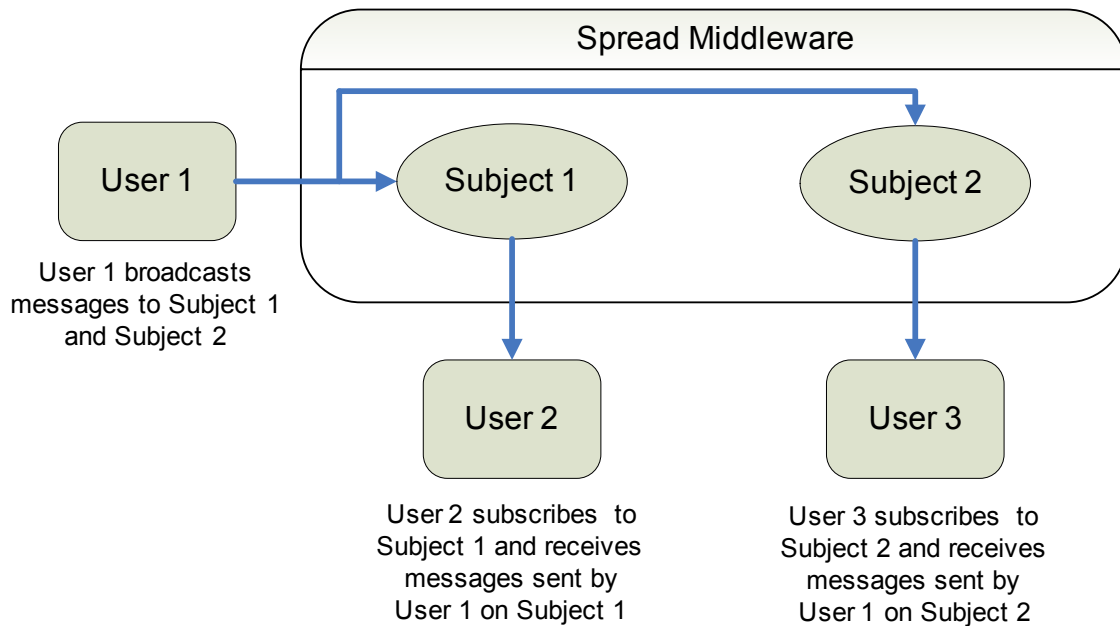


Figure 12: Middleware design concept

The middleware design concept was the product of discussions with our various mentors. From this concept, we developed the follow design requirements for a middleware implementation:

- 1) Middleware software replaces point-to-point connections between applications
- 2) Messages broadcast to channels, known as “subjects”
- 3) Messages received by “subscribing” to “subjects”
- 4) All applications will use a standardized data format
- 5) The middleware will be developed in such a manner as to insure reliable communications (no dropped messages)

Another important requirement for the middleware was to standardize how information and data was formatted for transmission across the middleware. With all of the various communication connections in the previous system design, information exchange was handled and formatted in a many different ways. Standardization of data formatting along with the communications simplification of the middleware concept will make the FFTB much more modifiable, extendable and flexible. Reliability is also a critical requirement because the middleware will be used for communicating data for simulation runs. Lost messages would mean lost data, which could lead to a corrupted simulation run.

5.2 SpreadXML3 Design

SpreadXML3 is the name of the software package that we developed as part of an implementation of the middleware concept. This package employs the Spread toolkit to implement the middleware concept. It also uses a custom scheme of XML formatting to standardize the format of the transmitted data. In order to use Spread, an instance of the Spread Daemon (which is an executable that comes with the Spread toolkit) needs to be running on a computer within the network. Applications that utilize the SpreadXML3 package can communicate through the middleware by using its objects. The

objects in SpreadXML3 utilize the libraries in the Spread toolkit, handle the communications at a higher level of abstraction, and provide better ease of use than the Spread toolkit libraries. The rest of this section describes how we developed SpreadXML3 and how its classes operate.

5.2.1 SpreadXML3 Design and Development Path

SpreadXML3 is actually the third generation of our design for a middleware software package. Our first design, called Thrower-Catcher, was a very simple design and developed as a proof-of-concept demonstration for the Spread toolkit, but it did not include the XML formatting feature. This first design was developed to learn how to use the libraries in the Spread toolkit. In our next design, JmbProject (which meant “Java Message Blaster” project), we added the functionality of parsing XML data. The JmbProject design was fully functional, however we were not satisfied with how we arranged its structure and provided its functionality. This caused JmbProject to be somewhat confusing to use. Therefore, we restructured the functionality of JmbProject into a different class structure and created the current version of our middleware package, SpreadXML3. SpreadXML3 divided the functionality of JmbProject into a more intuitive design that used four different Java objects. These objects are *MessageReceiver*, *MessageSender*, *MessageHandler*, and *MessageWrapper* and their function and interaction is described below.

5.2.2 Object Interaction within SpreadXML3

The objects in SpreadXML3 were designed to divide the four functionalities of i)sending, ii)receiving, iii)processing, and iv)XML parsing. In order utilize the full functionality of SpreadXML3, an instance of each of the four objects need to be created. However, if the programmer using SpreadXML3 does not need the full functionality, they only need to use the objects that pertain to what they need. For example, if the functionality to send messages was not needed by an application then the *MessageSender* object is not necessary.

The *MessageSender* object handles a connection to the Spread middleware and its methods are called to establish the connection and to send messages to specified subjects. The *MessageReceiver* object handlers a connection to the Spread middleware and its methods are called to setup the connection and to specify what subjects it will receive messages on. The *MessageReceiver* runs in threaded loop where it polls and receives messages from the Spread connection. The *MessageReceiver* then distributes each received message to every *MessageHandler* registered in its internally stored list of *MessageHandlers*. The *MessageHandler* object is actually a Java interface, which means that it is only a partially implemented framework. The application that uses SpreadXML3 must fill in the appropriate message processing code for *MessageHandler*.

The previous three objects, *MessageSender*, *MessageReceiver*, and *MessageHandler*, fully encompass the network communications aspect of SpreadXML3. The sole purpose of the *MessageWrapper* class is to provide the XML parsing functionality. *MessageWrapper* is a class that utilizes the auto-generated XML-serializable objects created by the JAXB toolkit. It serves as a wrapper for the functionality of the JAXB toolkit and presents a more intuitive and easy-to-use interface for the JAXB toolkit and auto-generated objects. The structure of these auto-generated XML-serializable objects is based on the structure defined in our XML schema.

5.2.3 Message XML Schema Description

The XML schema for our “message” object defines the structure of the “message” data object and is utilized by the JAXB toolkit to auto-generate the XML-serializable data objects. This schema can be found in Appendix B. The general purpose of our “message” object is pass data. Therefore, each “message” object represents a data point and contains an unbounded list of “message entry” objects. Each message entry has three elements that define the “key”, “type”, and “value” of the message entry. The “key” is the unique variable name for the datum (ex: “time (sec)”). The “type” is the name that specifies the format of the datum (ex: “integer”). The “value” is the value of the datum (ex: “31435892175”). By adding multiple message entries with corresponding information such as time, position, velocity, etc., you create a message that represents a data point of information. Following this paragraph, Figure 13 shows the structure of an example “message” object.

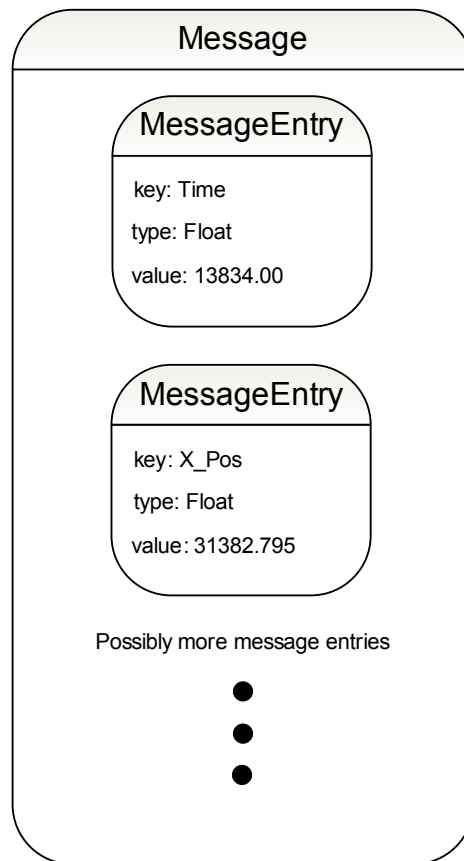


Figure 13: Structure of the Message configuration file

The actual XML formatted text for the above example message would look like:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<Message>
  <MessageEntry>
    <key>Time</key>
    <type>Float</type>
    <value>13834.00</value>
  </MessageEntry>
  <MessageEntry>
    <key>X_Pos</key>
```

```
<type>Float</type>
<value>31382.795</value>
  </MessageEntry>
```

...
There would be more MessageEntry blocks here if there were more MessageEntries

```
...
</Message>
```

5.2.4 Message Sender Component

The *MessageSender* object handles a connection to the Spread middleware for the purpose of sending messages. It has the methods “connect” and “reconnect” to establish and reestablish a connection to the middleware with specified parameters such as the host location and application user name. The host location is the parameter to specify the machine where the Spread middleware is running. The application user name is a unique name that each connection to the Spread middleware must have. No two connections (whether for sending or receiving) may share the same user name. This object also has the method “disconnect” for the purpose of terminating the connection to the Spread middleware and releasing its hold on a specific user name.

The main functionality of the *MessageSender* of sending messages is manifested in the “send message” method. This method has several different modes of operation depending on what parameters it is given. It must be given the data to send either in a Java String format or as a byte array. There is the option to supply a single subject, multiple subjects or no subjects as the second argument. If no subjects are specified, then the *MessageSender* uses its internally stored list of subjects to send messages to. This internally stored list of subjects is held in a list and managed with several methods such as “add subject”, “remove subject”, and “remove all subjects”. The internally specified list of subjects is unnecessary if subjects are supplied with every call of the “send message” method, however the list is a convenience for an application that only sends messages to a set list of subjects.

5.2.5 Message Receiver Component

The *MessageReceiver* object handles a connection to the Spread middleware for the purpose of receiving messages. This object has identical connection setup methods as *MessageSender*. The “connect” and “reconnect” methods are used to establish and reestablish the connection to the middleware based on specified arguments for host name and application user name. The “disconnect” method is used to terminate the connection to the middleware.

In order to receive any messages at all, the *MessageReceiver* must join groups (“subscribe” to subjects). The methods “join group”, “leave group”, “leave all groups”, and “set groups” handle the “subscribing” and “unsubscribing” to subjects. This is different from *MessageSender*, because it is always necessary to subscribe to groups in order to receive messages. The Spread middleware will only redirect messages to a *MessageReceiver* if that *MessageReceiver* has subscribed to the group that the message was sent to.

In addition, for the received messages to be distributed to the appropriate part of the application, *MessageHandlers* must be added to the *MessageReceiver*. These *MessageHandlers*, which are described in more detail below, are stored by reference in a list held in *MessageReceiver*. The list is managed by the

methods “add message handler” and “remove message handler”, which adds or removes a reference to *MessageHandler* specified by the method’s parameter.

Another major difference between the *MessageReceiver* and the *MessageSender* is that the *MessageReceiver* runs in a threaded loop and will automatically call a function when it receives a message. This loop needs to be started by a call to the method “start receiving” after the connection is established. The *MessageReceiver* loop continuously polls the Spread connection for messages. If there is an available message, it receives the message and then distributes it to every *MessageHandler* registered in its list of *MessageHandlers*. It distributes the message to the *MessageHandler* by calling the *MessageHandler* method “handle message” with the supplied arguments of the message received, the subjects that the message was sent on, and the name of the sender.

5.2.6 Message Handler Component

The *MessageHandler* component is an interface, which is a partially implemented framework that must be filled in by the application that uses SpreadXML3. This interface has only one abstract (undefined) method “handle message”. However, the “handle message” method must be defined to take three arguments: message data, a list of the subjects the message was sent on, and the name of the sender of the message. The rest of the method and what the *MessageHandler* will do with those arguments is completely open-ended and definable by the application utilizing SpreadXML3.

5.2.7 Message Wrapper Component

The final component of the SpreadXML3 is the *MessageWrapper*. Its purpose is to handle the XML aspect of the communications package. *MessageWrapper* has methods to build a “message” object by adding “message entries”, removing “message entries”, or clearing all “message entries”. The method “get XML string” serializes the “message” object its “message entries” and returns a String representation of the object formatted with XML tags. The corresponding method “set XML string” takes an XML formatted String, parses the XML, and sets the “message” object in *MessageWrapper* with the data from the XML String. To retrieve data from the “message” object, *MessageWrapper* has various methods to get “messages entries” and the data stored in “message entries”.

5.3 Adapter/Client Modifications

Our design work led us to modifying the Adapter and Client software that the 2003 WPI student team created. The Adapter/Client package is a key component of the FFTB setup that communicates with various other components of the FFTB. The purpose of the Adapter is to relay data from the GPS receivers to the flight software component. The Client application has the purpose of viewing and archiving the data that is being received by the GPS receivers. The Adapter/Client was a testing ground for our SpreadXML3 package and a start toward implementing the middleware communications scheme for the FFTB.

5.3.1 Adapter/Client Design Improvement Requirements

The design requirements for the enhancement of the Adapter/Client are:

- 1) Utilize Spread middleware for communication
- 2) Adapter serial communications support for Pivot GPS receivers
- 3) Flexible Adapter serial communications

The major design enhancement requirement with the Adapter/Client is to utilize the Spread middleware for communications as opposed to utilizing and handling several point-to-point communications connections. Another requirement is serial communications support for the Pivot GPS receivers. These receivers also communicate over the serial ports and send data messages in a format similar to the Orion GPS receivers. The Adapter serial communications should also be flexible and not require any major reconfiguration if an Orion receiver was swapped with a Pivot receiver.

5.3.2 Previous Operation of Adapter/Client

Re-implementing the communications of the Adapter/Client with SpreadXML3 first involved understanding the structure and functionality of the previous applications. This involved reading through source code and reverse engineering how these programs operated. We were also assisted by information provided by the documentation in the final report of the 2003 WPI project team. The below information on the Adapter/Client pertains to our design changes. Additional information about the Adapter/Client can be found in Section 2.3.2 of Background Chapter.

The previous implementation of the Adapter and Client, created by the 2003 WPI project group, utilized various point-to-point communications protocols for the communications between the Adapter, Client, GPS receivers, and flight software. These point-to-point communications protocols were hard coded in each application and adding another program into the communication loop required source code modification in one of these applications. The below diagram in Figure 14 is from the 2003 WPI project team's final report and shows how the Adapter, Client, GPS receivers, and flight software previously communicated.

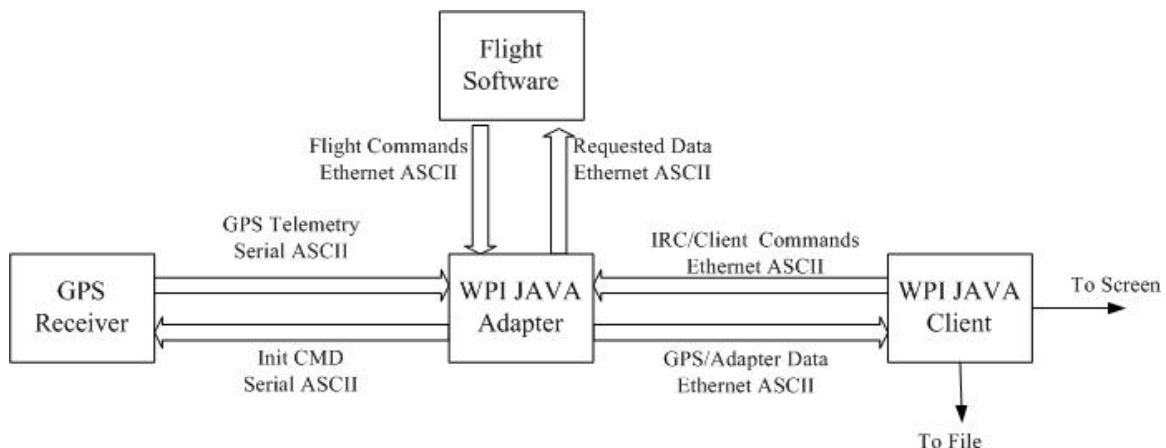


Figure 14: Adapter/Client communications diagram³

The block diagram (from the final report of the 2003 WPI project team) in Figure 15 below shows the various internal modules of Adapter and how it functions. The Adapter communicated to the GPS receivers, specifically the Orion brand GPS receivers, through the serial ports. Therefore, the computer that ran the Adapter program must have the Orion receivers connected to its serial ports. The

Adapter sent an initialization command to the GPS receivers and then listened to the serial ports for data messages from the receivers. The Adapter communicated with the flight software and the Client using a point-to-point communication protocol that utilized the Java socket connection classes. The communication from the Adapter to the Client and flight software was a two-way scheme. The Client or flight software would send a command, requesting a specific GPS data message, to the Adapter. The Adapter responded to these commands by sending the appropriate GPS data message to the software that requested it.

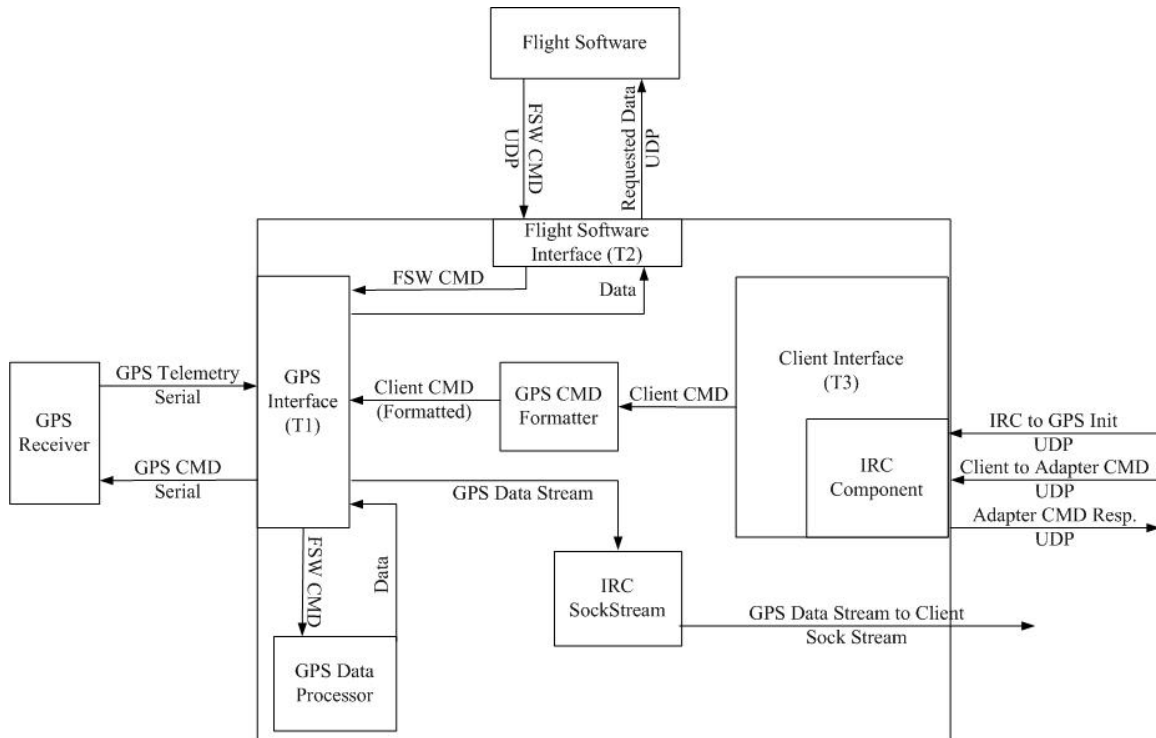


Figure 15: Low-level adapter functional diagram³

We realized that the FFTB software engineers had already made a small modification to the Adapter software. The Adapter no longer sent an initialization command to the GPS receivers at launch; instead this functionality was moved into another program that solely initializes the GPS receivers. In the above two Figures 14 and 15, it still shows a communication channel for commanding the GPS receivers, but that communications channel is no longer used.

5.3.3 Adapter Modifications

Replacing the point-to-point communications concept with the Spread middleware communication concept meant removing many components within the Adapter. We removed all of the various objects for processing data request commands and sending data to other applications. The bulky complicated Adapter that you see in Figure 15 in the above section was cut down into the small, streamlined version that you see in Figure 16 below.

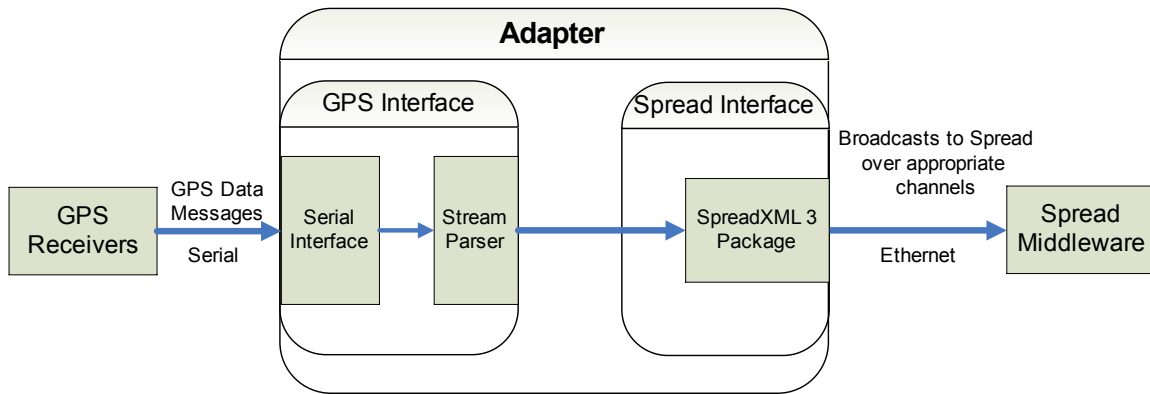


Figure 16: Adapter with modified communications

As you can see the Adapter has been simplified into two major components, the GPS Interface and the Spread Interface. The GPS Interface handles the communications with GPS receivers and consists of the serial port interface component and the stream parser component. The serial port interface component simply listens to the serial ports and passes a stream of data into the stream parser. Both the Orion and Pivot GPS receivers communicate data by sending ASCII string GPS messages over the serial port. These messages are preceded with a specific starting character and a specific terminating character. The stream parser reads the stream into buffers and extracts GPS data messages from the stream, by finding the start and stop characters of the messages.

These ASCII string data messages, along with information that specifies the serial port and receiver that it came from, are then passed into the Spread Interface component of the Adapter. This Spread Interface component parses the string and finds an identifying tag that specifies the message type. These message types specify different messages (such as “F40”, “F42”, “F45”) that are sent by the receivers. For each message type that we may expect from the receivers, we defined functions that parse the data out of the strings and store them in *MessageWrapper* objects. These *MessageWrapper* objects are then serialized into XML formatted strings with a call to the “get XML string” method. The XML formatted strings are then sent over the Spread middleware on a subject name that specifies the GPS receiver type, the serial port, and the message type, following the standard “[receiver type][serial port number].[message type]”. For example, an “F40” message from the Pivot receiver on port 1 would be sent on the subject “Pivot1.F40”. The Orion receiver message types that are processed and sent by the Adapter are “F40”, “F42”, “F45”, and “F46”. The Pivot receiver message types processed and sent by the Adapter only consist of the “F40” and “F42”.

As the Adapter is running it dumps data to the screen showing message statistics and the content of the stream parsing buffers. We made no major visual changes to the Adapter program. In the below Figure 17, there is a screenshot of the Adapter.

```

wpi@dozer:~/workspace/java_suite - Shell - Konsole
Session Edit View Bookmarks Settings Help
470.371 0      0.00      +0.000      +0.00018  21244850.97-159932048.952 +24
1.02122  20315559.51-161643589.779  -156.701 5  22509503.53-163345566.649  -413.
551 0      0.00 >
MSG BUF    : <F401289222048.0000013 +1128536.73 -4833818.90 +3991818.76  -0.0
0886      -0.00527  -0.003572 9 2.06A>
WRITE POS  : 245
CHUNK START : 165
CHUNK LEN  : 80
-----
ETX NOT FOUND
AFTER
-----
DATA BUF   : <F421289222048.0000013 0.100654454 9 22343902.96 +2431866.241 +
470.371 0      0.00      +0.000      +0.00018  21244850.97-159932048.952 +24
1.02122  20315559.51-161643589.779  -156.701 5  22509503.53-163345566.649  -413.
551 0      0.00 >
MSG BUF    : <F401289222048.0000013 +1128536.73 -4833818.90 +3991818.76  -0.0
0886      -0.00527  -0.003572 9 2.06A>
WRITE POS  : 245
CHUNK START : 165
CHUNK LEN  : 80
-----
-----checksum err:0 %=0.0 stx/etx err:41 msgs:6167 bytes: 1018747

```

Figure 17: Screenshot of Adapter

5.3.4 Impact of Adapter Modifications

The Adapter modifications have a substantial impact on the accessibility of the GPS receiver data. The previous communications model shown in Figure 14 in Section 5.3.2 has been completely restructured. The Client and the flight software no longer need to send commands to the Adapter to request GPS data. The Adapter simply broadcasts this data with the Spread middleware. The Client, flight software, or any other component that needs this data simply has to subscribe to the appropriate subject(s) that the data was sent over. The block diagram in Figure 18 below shows the new communications model.

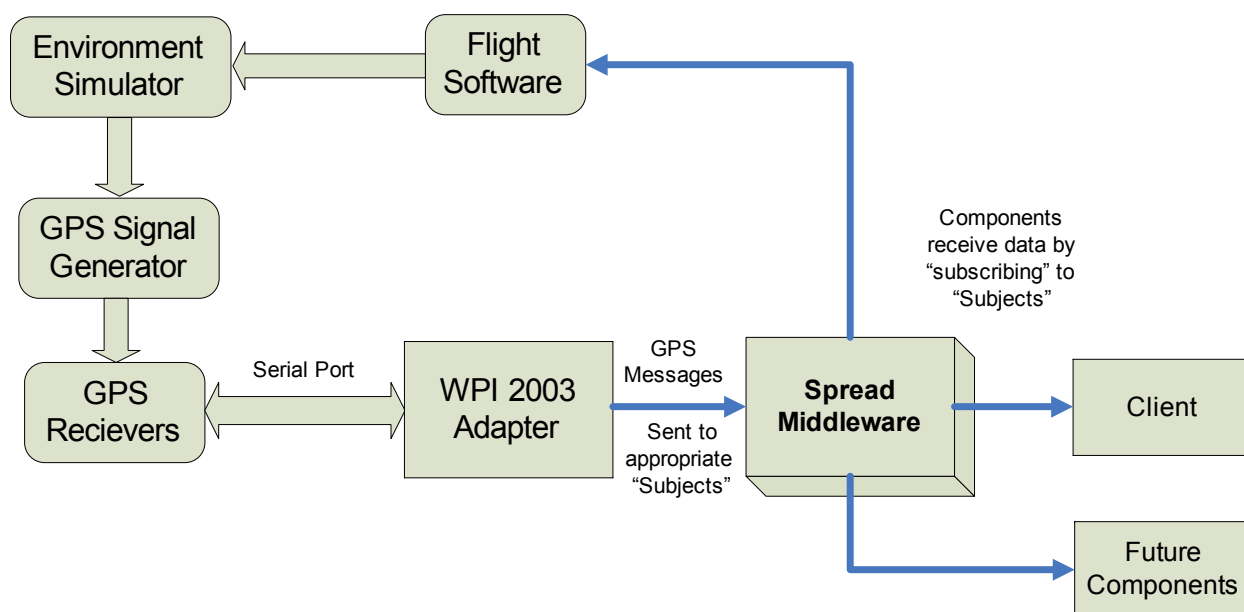


Figure 18: New network communications model

The flight software component is often swapped with new flight software components created by developers from other labs. For the new component to access the GPS data, it will need to be modified to use the SpreadXML3 package. The process of swapping in a new flight software component may seem tedious, however it is actually simpler process than before. With the former, point-to-point communications scheme, both the Adapter and the flight software would need to be modified to set up communications. The flight software would need to be rewritten to handle a point-to-point connection, send command messages to the Adapter to request the data it needed, and process and parse the data. With the new communications implementation, the flight software can use our SpreadXML3 package to access the middleware. The SpreadXML3 package provides an API for receiving and interpreting the messages. Also, the Adapter does not need to be modified for the swapping of a new flight software component.

We even could use other software that we developed, such as NetPlot, our real-time plotting software for the middleware (further described in Section 5.5) and Data Logger, a data logging utility for the middleware (further described in Section 5.6), to access and process the data being broadcast from the Adapter. This involved no modification to the Adapter, its source code, or its communications configuration. NetPlot and Data Logger were not design to communicate specifically with the Adapter, but instead with the middleware. Before we had this middleware communications implementation of the Adapter, setting up another program to communicate with the Adapter would mean source code modifications on both programs.

5.3.5 Client Modification

The purpose of the Client is to display the data being relayed from the Adapter to the flight software and to archive this data to a file. Our modifications of the Client consisted of replacing the communications implementation and left most of the visual functionality alone. The communications object that handled the point-to-point connection with the adapter was replaced with an object that

utilized the SpreadXML3 package to connect to the Spread middleware. This Spread interface object also extracts the data from the messages and redirects the data into the appropriate GUI components and archiving components, based on the subject of the message. We did not modify or redesign the GUI at all. A screenshot of the main GUI window is shown below in Figure 19.

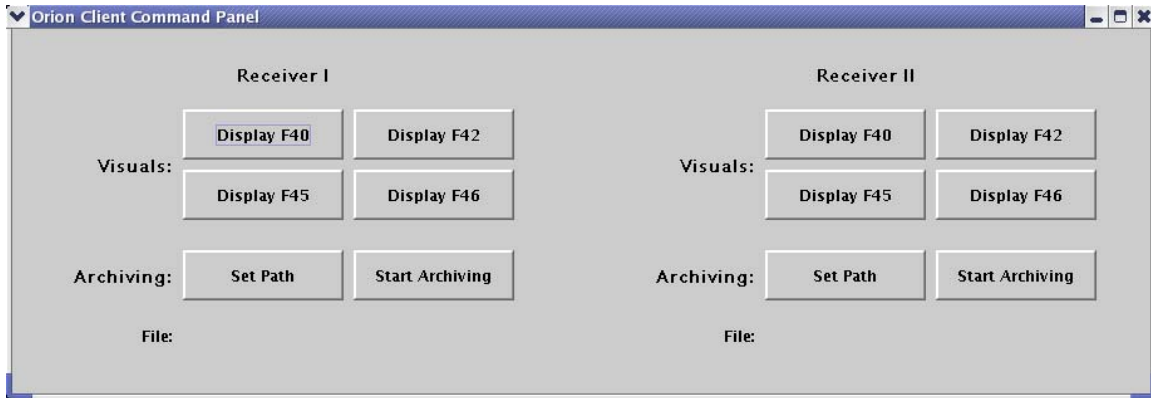


Figure 19: Screenshot of Client

The GUI has two identical halves for each GPS receiver. Each half has four buttons, labeled beginning with “Display F”, that open windows that display data from the message specified on the button. The “Set Path” button opens a dialog that allows the user to specify the path of the archive file. The “Start Archiving” button, which toggles to “Stop Archiving” on clicking, starts or stops the archiving of data to that file.

The left half is labeled “Receiver I” and its functionality pertains to the GPS receiver connected serial port 1 on the computer running the Adapter. The data displayed on the windows that spawn from the buttons on the left half are from that receiver on port 1. Therefore, the messages from the middleware with subjects that begin with “Pivot1” or “Orion1” are redirected to the GUI objects spawned from the left half buttons. The archiving functionality of the left half also pertains to the messages from the receiver on port 1.

For the right half, which is labeled “Receiver II”, all of the functionality pertains to the GPS receiver connected to serial port 2 on computer running the Adapter. The messages from the middleware with subjects that begin with “Pivot2” or “Orion2” are redirected to the GUI objects spawned from the right half buttons. The archiving functionality of the right half also pertains to the messages from the receiver on port 2. Figure 20 below shows a screenshot of an example message display window showing the “F40” message from the GPS receiver on serial port 1.



Figure 20: Screenshot of a F40 message display window

5.4 FFTBPlot Plotting Software Modifications

FFTBPlot emulates real-time plotting by continuously reading data from a file that is constantly updated by the data-generating source. FFTBPlot utilizes JFreeChart, an open-source Java plotting library, as the plotting engine, and the JAXB toolkit for saving its configuration information in an XML formatted file. However, it does not use the JAXB toolkit for any kind of XML interpretation for reading data.

We had several objectives, as listed previously in Chapter 3, relating to the enhancement of the FFTBPlot software including adding the functionality to read data from network sources. FFTBPlot would be able to fit right into our middleware communications scheme, because it would only need to connect to the middleware and subscribe to the groups that it will plot data from.

5.4.1 FFTBPlot Design Improvement Requirements

The design requirements for the enhancement of FFTBPlot are:

- 1) Primary – important, critical to overall goals
 - a) Data acquisition from Spread middleware
 - b) Reliable, less buggy performance
- 2) Secondary – beneficial, but not necessary
 - a) Ability to customize appearance of data representation
 - b) Improved GUI functionality, flow, ease of use, extensibility, aesthetics
 - c) Data manipulation features
 - d) Improved threading and efficiency

We determined the design requirements noted above from discussions with the author of the FFTBPlot software and our mentor. The most important design requirement was to enable data acquisition from the Spread middleware. The former method of data acquisition from a file was a poor emulation of real-time plotting and made it difficult to extend data acquisition from several machines over a network. Another important requirement was to fix known bugs in FFTBPlot. The software would be useless if bugs made it too unreliable to be trusted.

The rest of the requirements were given to us as secondary objectives. For example, customizable appearance for the data, an improved GUI, and data manipulation features were not as important as creating a functional, bug-free FFTBPlot that can plot data from the Spread middleware. Improving the threading scheme and making the program more efficient were also secondary objectives, because they required a substantial amount of time to reverse engineer and understand the current threading scheme as well as its efficiency consequences.

5.4.2 Previous Operation of FFTBPlot

The layout and presentation of FFTBPlot was very complex and structured with many layers of GUI objects. The GUI was mainly broken down into the data source configuration aspect and the plot creation and setup aspect. Each of these two areas involved moving through a series of nested GUI objects that dynamically changed as objects (data sources, plot frames) were added, selected, reconfigured, or removed. This structure is explained in more detail in a documentation file in Appendix D. The author of FFTBPlot, Victor Lu, created this documentation as an explanation for the design and use of FFTBPlot.

On the data acquisition side, users needed to define the data sources specifying information such as the source file location, data to expect in each column, format of data to expect in each column, and column delimiters. This data source configuration information was saved automatically to XML formatted files. FFTBPlot automatically loads these configuration files on launch.

On the plotting side, FFTBPlot allowed the user to define multiple plot windows (internal plot frames). Each of these plot windows could then contain multiple plots (plot graphs and axes). Furthermore, each of these plots could plot multiple data series (plotted line). Choices made about the data series to be plotted were given based on user specifications for the data sources in the data source configuration. The configuration for the plots can also be saved and loaded to an XML formatted file. This allowed the user to load a plot that had been previously created.

After the data sources are configured properly and the plot frames are set up, clicking the “Plot” button starts data acquisition and plotting. An object that performs data acquisition opens a file, reads it one line at a time, and passes appropriate data to the plotting engine. The plotting engine used by FFTBPlot is JFreeChart, a free open-source plotting library.

5.4.3 Efforts to Introduce the SpreadXML3 Package

The first step in order to add the SpreadXML3 package to FFTBPlot was to reverse engineer the FFTBPlot software to understand how it worked. The file in Appendix D helped us, but we had to peruse and analyze the code and draw flow charts to understand object hierarchy and program operation.

Through this process we determined that we needed to modify the data configuration aspect of the program and the object for reading data from files to support the data acquisition from the middleware.

The changes that we made to FFTBPlot consisted of GUI modifications, configuration file structure modifications, and data-reading object modifications. The GUI for the configuration of data sources needed to be modified to allow the user to specify the “network” data source type and other information needed to configure a connection to the Spread middleware. The configuration file structure was modified in order to allow those settings to be saved. A new data-reading object that used the classes from SpreadXML3 was created to handle network data sources and pass data in the appropriate format to the plotting engine. The SpreadXML3 classes that were utilized are MessageReceiver, MessageHandler, and MessageWrapper.

5.4.4 Design Change Decision

The changes described in the above section satisfied the design requirement of adding data acquisition from the Spread middleware, however some problems arose that forced us to consider a major design decision that abandoned FFTBPlot. There were some serious performance issues with the plotting of data from the middleware. The plots would freeze up and update at irregular intervals of time, as opposed to once a second. We attributed this problem to multiple reasons:

- 1) The inefficiency and unreliability of JFreeChart (bugs, glitches)
- 2) The addition of threading for receiving messages from middleware
- 3) The bulky, inefficient object hierarchy of FFTBPlot, and
- 4) The unreliable FFTBPlot source code (bugs, glitches)

We knew that FFTBPlot played a factor in the inefficiency because there were always some performance issues with FFTBPlot. From the beginning, FFTBPlot had a bulky and complicated object hierarchy. The bulkiness of the program caused even the initialization of FFTBPlot to take nearly two minutes. The inefficiencies of JFreeChart also play a factor in the poor performance. JFreeChart was not designed for real-time plotting and it handles data points inefficiently by creating a data object for each point. Another issue was the bugs in FFTBPlot and the bugs caused by JFreeChart. Determining the source of the bugs would be an extensive task because of the length of the FFTBPlot source code. The author of the software was not even able to resolve all of them. Stripping down the bulkiness of FFTBPlot would also be difficult due to the length of code and complexity of the GUI.

After discussing various limiting factors such as hard to fix bugs, inefficiency of the plotting engine, and bulkiness of the code we decided on abandoning FFTBPlot and starting from scratch to develop a new real-time plotting application. Based on the recommendations of our mentor we also made the design decision to discard JFreeChart and instead utilize PTPlot, another open-source plotting engine that would be more appropriate and efficient for the task of real-time plotting.

5.5 NetPlot Plotting Software Design

The purpose of NetPlot was to replace the functionality of FFTBPlot with a simpler design that utilized a different plotting engine. On top of replacing the functionality, we also added some new

functionality such as network data acquisition, customizable appearance of the data representation, and command-line execution for scripting. Another benefit of NetPlot is its simplicity. There are only three non-generated source code files as opposed to the several dozen in FFTBPlot. The GUI is very simple and has easy-to-use plot setup features as opposed to FFTBPlot's complicated GUI that makes plot setup and data source configuration a difficult process. Furthermore, creating NetPlot from scratch avoided the arduous task of resolving the bugs in FFTBPlot. A user manual for the application is located in Appendix A.

5.5.1 NetPlot Design Requirements

The design requirements for NetPlot were:

- 1) Data acquisition from Spread middleware
- 2) Utilize an efficient plotting engine
- 3) Simple, intuitive GUI
- 4) Customizable appearance of the data representation
- 5) Command line executable (for scripting)
- 6) Save/load configuration files

We derived most of these requirements based on what was required for the design improvements on FFTBPlot, but we also had discussions with our mentor to determine new design requirements to help avoid recreating the problems faced in FFTBPlot. Data acquisition from the Spread middleware, for the purpose of real-time plotting, was the main objective of NetPlot since it was the primary objective with FFTBPlot. Utilizing the more efficient plotting engine PTPlot was also more important in order to avoid the FFTBPlot problems associated with the inefficient JFreeChart. A simple, intuitive GUI would make it easier for anyone unfamiliar with NetPlot, or even the middleware concept, to use the application. A simple, efficient design may also help reduce the chance of bugs and improve performance. The customizable appearance of the data representation was an objective that we had hoped to accomplish with FFTBPlot, but now may be more easily accomplished with the features in the new plotting engine, PTPlot. The command line executable requirement was essential for the ability to script NetPlot. Saving and loading the configuration of the NetPlot would allow the application to be completely scripted. Ideally, this application would be started, load the files automatically, and function smoothly, all from a script used to start a simulation run on the FFTB.

5.5.2 Main Window Design

The main application object of NetPlot (which was also called *NetPlot* itself) utilizes the SpreadXML3 package in order to handle the middleware connection. It holds the instance of *MessageReceiver*, which receives all of the data for the application. The *NetPlot* object itself implements the *MessageHandler* interface. This was necessary for the main application to be able to process messages for its ability to auto-refresh data series lists. The *NetPlot* object also has a *MessageWrapper* object to interpret the XML messages that it receives.

The *NetPlot* object was also the main GUI window. The design of this main GUI window was very simple and intuitive. It is a single frame with mostly only text fields, buttons, and checkboxes for

interaction with the application. There is also a combo-box and a list for selecting the data series to plot. All of the GUI components are marked with concise labels that explain their purpose. Additionally, there are tool-tip text pop-ups that appear if you hover the mouse over a text field, button, checkbox, combo-box or list. The tool-tip text gives an explanation (a short sentence) of what the component in the GUI is used for. We also added a status bar at the bottom that displays messages as the program performs certain actions or if any errors occur. Below in Figure 21 is a screenshot of the NetPlot main window.

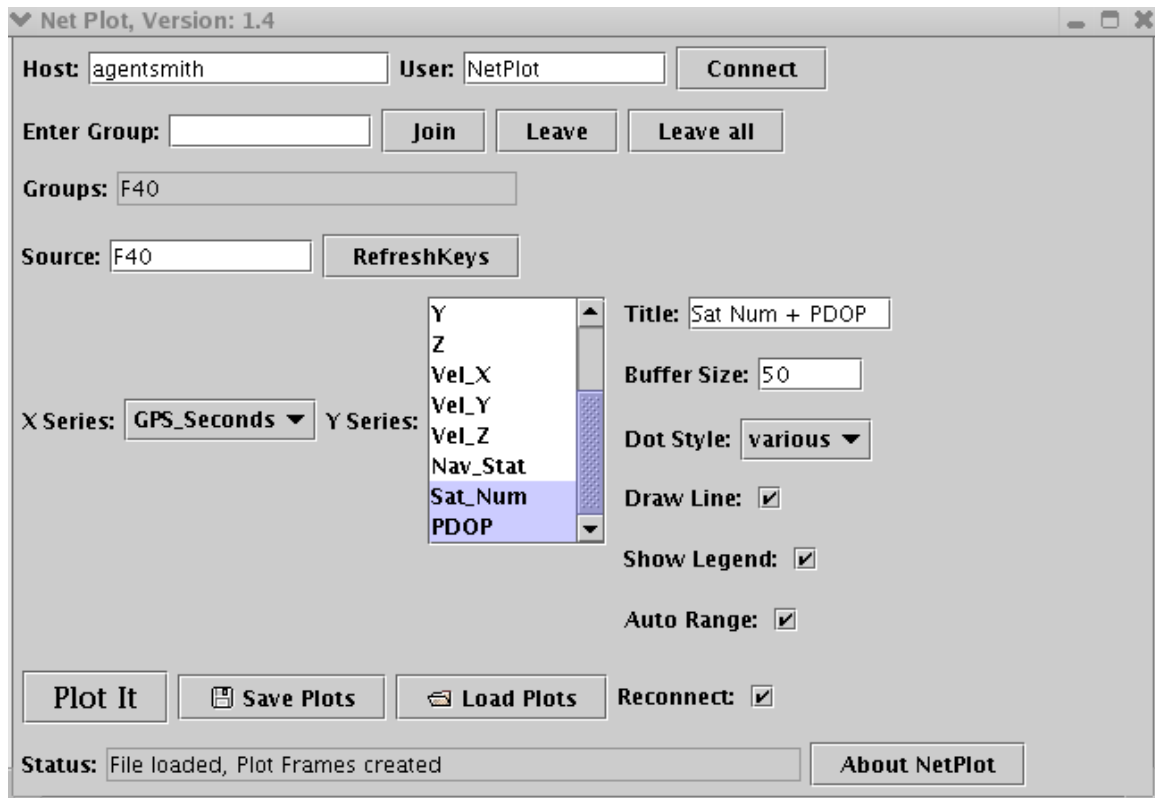


Figure 21: Screenshot of the NetPlot main window

The top row of the GUI has text fields, for specifying the host machine running the Spread middleware, what user name the *MessageReceiver* in NetPlot should connect with, and a button labeled “Connect”, to connect to the Spread middleware. Being able to specify the location of the middleware is important because the location of the middleware may be moved. Being able to specify the username is also necessary because NetPlot may be run on several different computers and each instance needs its own unique username.

The next two rows have GUI objects to manage the subscriptions for the *MessageReceiver* in NetPlot. The “Enter Group” text field is used to specify the parameter for the “Join” and “Leave” buttons, which respectively subscribe or unsubscribe to the specified group. The “Leave All” button leaves all the subscriptions that *MessageReceiver* holds. The “Groups” text field is a list of the current subscriptions of the *MessageReceiver* in NetPlot.

Setting up a plot is rather simple with NetPlot. The “Source” text field is used to specify what source you want to plot data from. The “Refresh Keys” button will subscribe to the specified group if not subscribed to it, receive a message on that specified source and automatically refresh the “X Series” and

“Y Series” selectors with the possible keys (variables) that can be plotted from that source. The “X Series” combo-box can only select one data series to be set as the X-axis, but the “Y Series” list allows multiple selections of data series to be plotted on the Y-axis.

To the left of the series selectors are some checkboxes, text fields, and a combo-box to set the customization of the appearance of the plot. The “Title” sets the title on the plot; the “Buffer Size” specifies how many data points are kept on the graph; the “Dot Style” is the style of marker for each data point; the “Draw Line” option selects whether to connect the points with a line; the “Show Legend” option selects whether to show a legend on the plot graph; the “Auto-Range” option selects whether to auto-scale the axes with every point plotted.

The “Plot It” button creates a *PlotFrame* object based on all of the information given in the various fields. The *PlotFrame* object implements *MessageHandler* and is registered with the *MessageReceiver*. The *PlotFrame* object also extends a PTPlot object, *PlotLive*, which gives it the functionality to plot points in real-time and sets up a window frame with some useful GUI features. *PlotFrames* are new windows that are created on the fly every time the “Plot It” button is pressed. The NetPlot application manages a list of references to these *PlotFrames* and registers each new *PlotFrame* with the *MessageReceiver* so that it can receive data to plot. When a *PlotFrame* window is closed, the NetPlot application unregisters the *PlotFrame* from the *MessageReceiver* and removes the reference to that window from its list.

The next elements of the GUI, next to the “Plot It” button, control the loading and saving of XML formatted configuration files for NetPlot. The “Save Plots” and “Load Plots” buttons both bring up a file dialog box to choose the file to save or load. Saving the configuration entails saving all of the connection configuration information setup in the NetPlot main window and the configuration, formatting, size, and location of each *PlotFrame* window. Loading the configuration file returns NetPlot to the state that it was in when the configuration was saved. This reopens all of the *PlotFrame* windows, starts their plotting processes and even sizes and positions their layout according to the same scheme that had been previously set up. The “Reconnect” check box to the side of the “Load Plots” button is to specify the option (by default, the option is selected) of disconnecting the current *MessageReceiver* connection to the middleware and reconnecting with the configuration information from the file. If unselected, the program does not reconnect and simply uses the previously establish connection to the middleware.

The feature to load and save configuration files allows NetPlot to be executed and fully configured from the command line. The command line execution of NetPlot allows the user to specify arguments for zero or more configuration files for NetPlot to automatically load on launch. This allows a standardized, pre-configured NetPlot to be started from a script. Figure 22 shows a block diagram of the structure of a typical NetPlot configuration file. The XML schema for the NetPlot configuration files can be found in Appendix B and an example configuration file can be found in Appendix C.

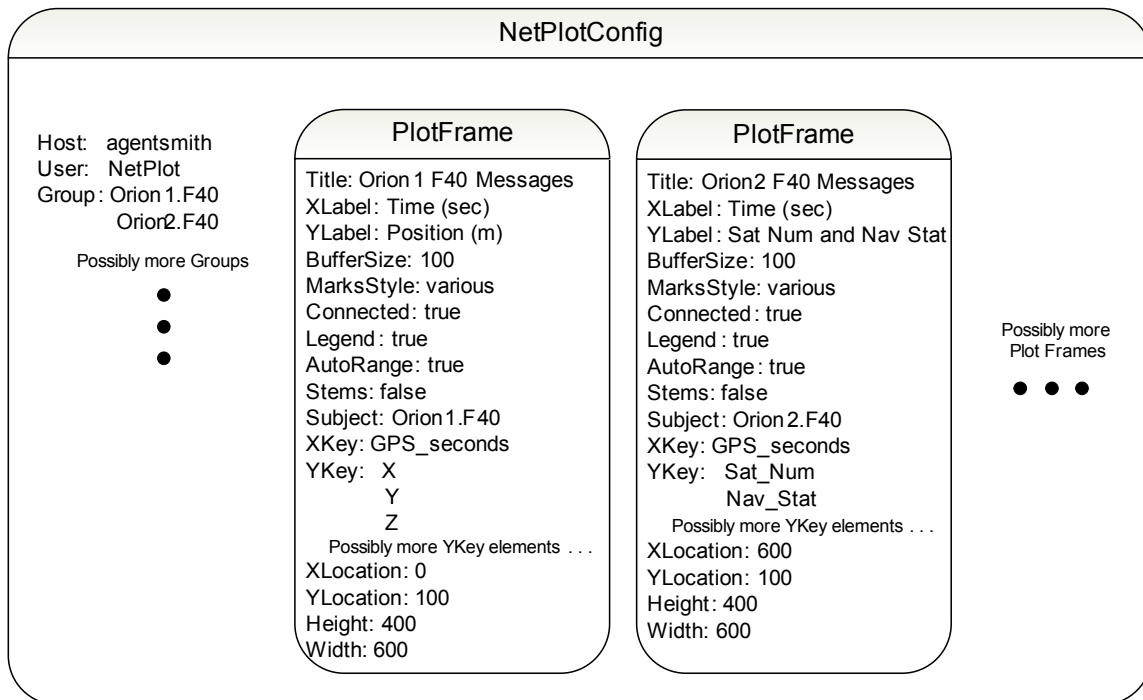


Figure 22: Structure of the NetPlot configuration file

5.5.3 Plot Frame Design

As we mentioned above, the *PlotFrame* extends the *PTPlot* object *PlotLive*. The *PlotLive* object supplies its own window frame and a useful GUI for the plot is already built. The *PlotLive* object left the “add points” method unimplemented, which we had to fill in with our code of how the points would be plotted. The *PlotLive* object runs in a thread where the “add points” method is called in a loop. *PlotFrame* also implements the *MessageHandler* object from the *SpreadXML3* package and is registered with the instance of the *MessageReceiver* held by the main application object. To implement *MessageHandler*, the unimplemented method “handle message” was defined to process the message, store the required data in global variables in the *PlotFrame* object, and set a flag to notify the “add points” method that new data can be added to the plot. The *MessageReceiver* passes received message to the *PlotFrame* object through a call to the “handle message” method. The received XML formatted message is interpreted with the help of the *MessageWrapper* class from *SpreadXML3*.

Figure 23 below shows a screenshot of the *PlotFrame* window. The *PlotFrame* window is very intuitive and behaves like a typical window. It is minimize-able, maximize-able, resizable, and closable. The plot graph reacts to changes in size and shape of the window and fits itself accordingly. The plot graph also allows click-and-drag zooming; a downward drag creates a box that shows where the plot graph will zoom in on, and an upward drag creates a box around another box that shows the relative ratios for the current viewed section and the size of the section after zooming out. Also, there are various tool-tip help texts that pop-up when hover the mouse over a button.

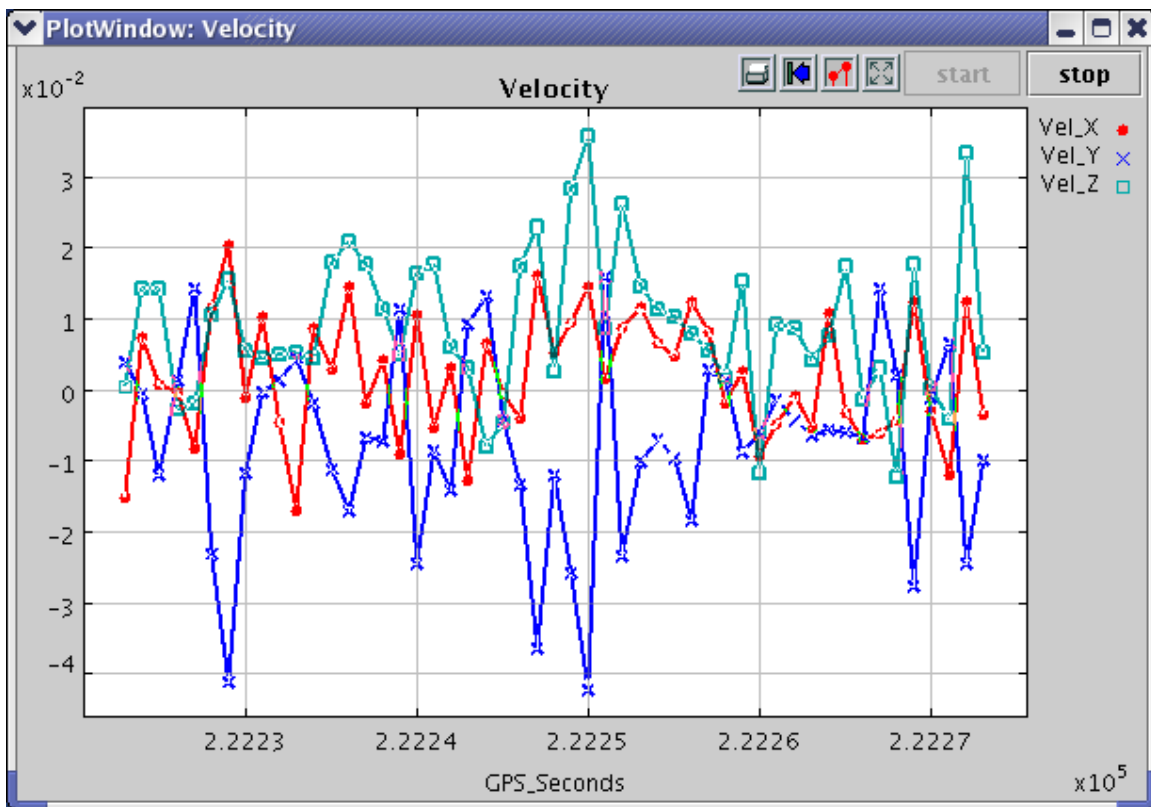



Figure 23: Screenshot of a Plot Frame window

In the above example, the *PlotFrame* is plotting the “Vel_X”, “Vel_Y”, and “Vel_Z” variables against the “GPS_Seconds” variable. The *PlotFrame* has been set up with various customizable data representation options, such as connecting the points with lines, showing data sets with the “various” dot style, showing the legend, auto-ranging the plot axes, etc. These options were set when the *PlotFrame* was created with the NetPlot main application or when NetPlot loaded the configuration information of the *PlotFrame* from a file. These data representation options can be changed even after the *PlotFrame* is created, by clicking on the format button, . This button brings up a pop-up window that allows the user to customize the *PlotFrame*. A screenshot of this pop-up window is shown below in Figure 24.

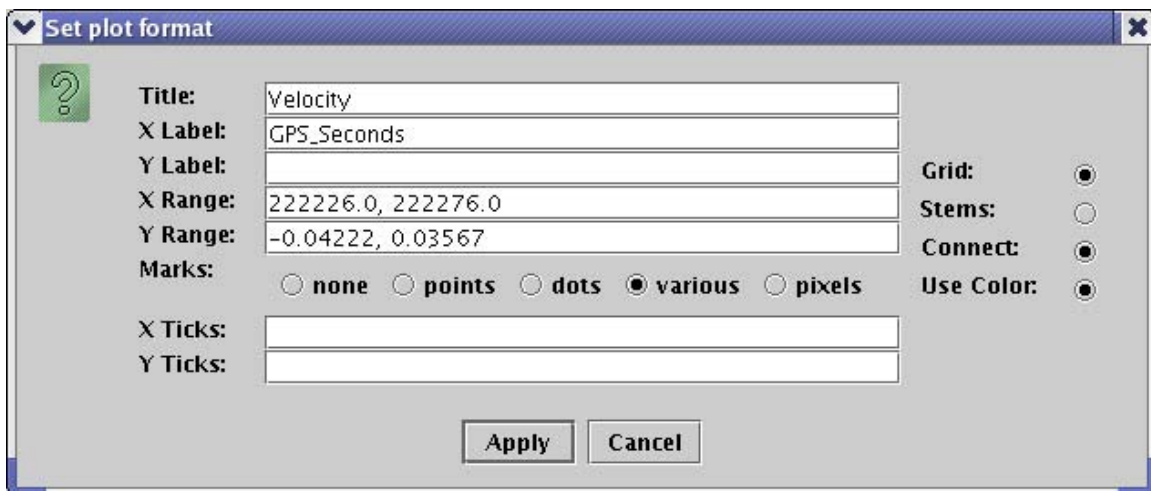





Figure 24: Screenshot of the plot format window

The other buttons on the *PlotFrame* window also have useful functions. The print button, , brings up the system print dialog and allows the user to print the plot either to the printer or to a file. The restore button, , restores the X and Y range on the axes to the original values. The fit button, , will resize the plot axes to fit the data; this option is done automatically with every point added if the auto-range was selected when configuring the *PlotFrame*. The “stop” and “start” buttons are used to pause or resume the addition of data points as data is received.

5.6 Data Logger Utility

The data logger utility is a simple application that we created to log data that came across the middleware. We developed this application at the end of our project time at NASA because the applications that used to the middleware needed some method for archiving the data. This application works as a plug-in to the middleware. Because of the design of the middleware, we were able to add data logging support to all data communicated across the network without modifying any of the programs sending or receiving the data. The application simply connects to the middleware and subscribes to the subjects that it is specified to log. A user manual for the application can be found in Appendix A.

5.6.1 Data Logger Design Requirements

The design requirements for Data Logger were:

- 1) Log all data on specified subject(s) from the middleware
- 2) Logs files in a simple, accessible format
- 3) Time stamp data
- 4) Simple, intuitive GUI
- 5) Command line executable (for scripting)
- 6) Save/load configuration files

We determined the above design requirements through discussions with our mentor. An important objective was that the data logger be able to log all data sent over a specified subject on the middleware. Loss of messages would not be acceptable for an important data log. It was also important that the data be logged in a light, but easily accessible format. The requirement to time stamp the data with the local system time and the time since the last message was received is required in order to assure the viewer of the log that none of the messages (which usually come at one-a-second) were dropped. A simple, intuitive GUI made it easier for anyone unfamiliar with the Data Logger, or even the middleware concept, to use the application. The command line executable requirement was essential for the ability to script the Data Logger. Saving and loading the configuration of the Data Logger would allow the application to be completely scripted. Ideally, the logger would be started, load the files automatically, and function smoothly, all from a script used to start a simulation run on the FFTB.

5.6.2 Data Logger User Interface Design

The user interface for Data Logger is very simple and intuitive. It is a single frame interface with only buttons, text fields, and lists for interaction with the application. All of the text fields, buttons, and

lists on the interface are marked with concise labels that explain their purpose. In addition, there are tool-tip text pop-ups that appear if you hover the mouse over a text field, button, or list. The tool-tip text gives an explanation (a short sentence) of what the component in the GUI is used for. We also added a status bar at the bottom that displays messages as the program performs certain actions or if any errors occur. Below in Figure 25 is a screenshot of the Data Logger.

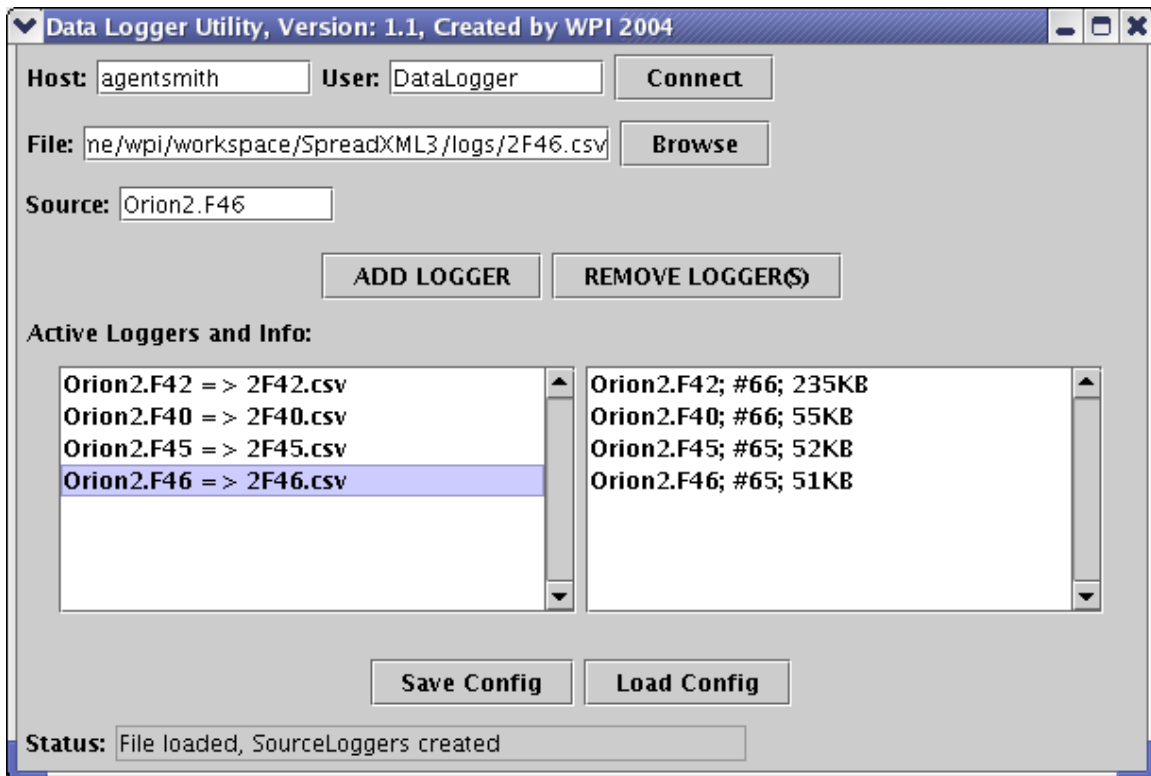


Figure 25: Screenshot of Data Logger

The top row of the GUI has text fields for specifying the host machine running the Spread middleware, what user name *MessageReceiver* in Data Logger should connect with, and a button labeled “Connect” to connect to the Spread middleware. Being able to specify the location of the middleware is important because the location of the middleware may be moved. Being able to specify the username is also necessary because Data Logger may be run on several different computers and each instance requires its own unique username.

The next two rows of the GUI allow are for specifying information needed to log a data source. The file text field is for specifying the path and filename of the log file, which can be either typed in or found using a file dialog that pops up by clicking the “Browse” button. The source text field is for specifying the subject to record data from. With these fields specified the user can start the logger threading by clicking on the “Add Logger” button. For each logger “added” a threaded object, named *SourceHandler*, is started that receives messages from the specified source and writes them to the specified file. The operation of the *SourceHandler* object is described in the next subsection.

In the middle of the application, two lists display the active *SourceHandler* objects. The left list displays the *SourceHandlers* with the convention of “[subject] => [log file]”. The right list displays the same *SourceHandlers* with the convention of “[subject]; [# of received messages]; [log file size]”.

Corresponding rows in each list correspond to the same *SourceHandler*. The user can use the left list to select one or multiple *SourceHandlers* by clicking. The “Remove Logger” button will terminate and remove the selected items.

In the second to last row of the GUI, there are two buttons for saving and loading Data Logger configurations. The save button will save the configuration of the application including the subject and log file path information for each logger in an XML formatted file. The load button will load the configuration out of an XML formatted file. Loading the configuration will initialize the connection, start the loggers on the specified subjects, and log them to the specified files. Data Logger can be run from the command line with none or more arguments that specify configuration files that automatically load on launch. This key design feature allows Data Logger to be scripted.

5.6.3 Data Logger Operation Details

The Data Logger application consists mainly of two objects, *Data Logger* and *SourceHandler*. The main Data Logger object consists of main application and GUI window. The main Data Logger object also handles the instance of *MessageReceiver*. The manner in which the Data Logger application handles the created *SourceHandlers* is similar to the manner that NetPlot uses. The *SourceHandlers* are stored in a list that can be modified by the “add” and “remove” functionality of the GUI. As the *SourceHandlers* are created and added, they are also registered with the *MessageReceiver*. As the *SourceHandlers* are removed and deleted, they are also unregistered with the *MessageReceiver*.

The *SourceHandler* object implements the *MessageHandler* interface from the SpreadXML3 package, by defining the “handle message” method to process the message received using the *MessageWrapper* class and then write the message data to the log file specified in that instance of *SourceHandler*. The *SourceHandler* object writes to the log file one line at a time, delimiting columns with commas. The first line that it prints is the header line that titles each column. This format of data logging is a standard called “Comma Separated Values” (file ends with “*.CSV”) and is readable by many spreadsheet and data analysis programs such as Microsoft Excel.

The Data Logger can save its configuration to an XML formatted file. This file includes the configuration information for connecting to the Spread middleware and the configuration details for each *SourceHandler*. The structure of a typical Data Logger configuration file is shown in a block diagram in Figure 26 below. The XML schema for the Data Logger configuration files can be found in Appendix B and an example configuration file can be found in Appendix C.

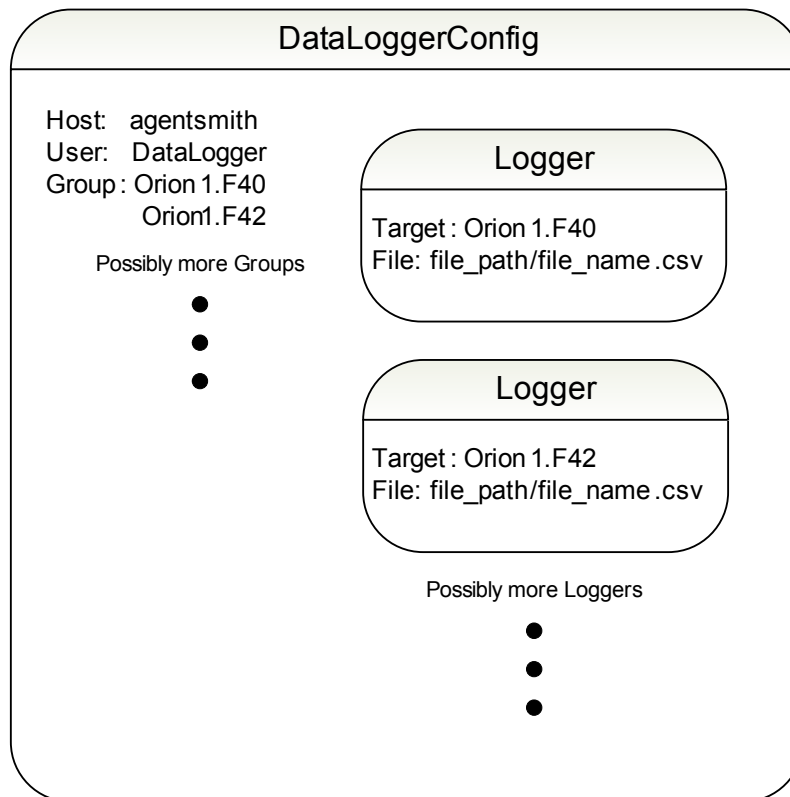


Figure 26: Structure of the Data Logger configuration file

5.7 Testing and Evaluation Design

Along with developing the software to accomplish our objectives, we also designed test applications to evaluate the functionality of our software. We designed both a simple GUI and data sender to verify the performance of our Java classes that provided the real-time plotting and middleware capabilities. The test applications allowed us to easily debug problems with our main Java classes since we could control and monitor the network communications.

5.7.1 TestGUI Design

Our first test application, called TestGUI shown in Figure 27, contained several buttons and text fields to demonstrate and configure the middleware communications. The two large text areas at the top of the application represented the send and receive areas, which were used to represent a sender and receiver of the data. The data from the sender field was sent to the Spread middleware on the specified subject and the receiver would subscribe to the group in order to receive the data. The settings for the Spread connection are handled by the text fields denoted “Host”, “RecUser”, “SendUser”, “Subject”, and “Group”. The “Host” refers to the central computer that Spread is running. “RecUser” and “SendUser” are the unique usernames for the receiving and sending applications, respectively. The “Subject” is the channel that sent messages are broadcast to, while “Group” is the channel that receivers subscribe to receive messages. TestGUI also has buttons associated with the “Subject” and “Group” text fields in order to join, leave, or leave all subjects or groups. Each of these fields also has an unchangeable text field to monitor the current subject and group subscription lists.

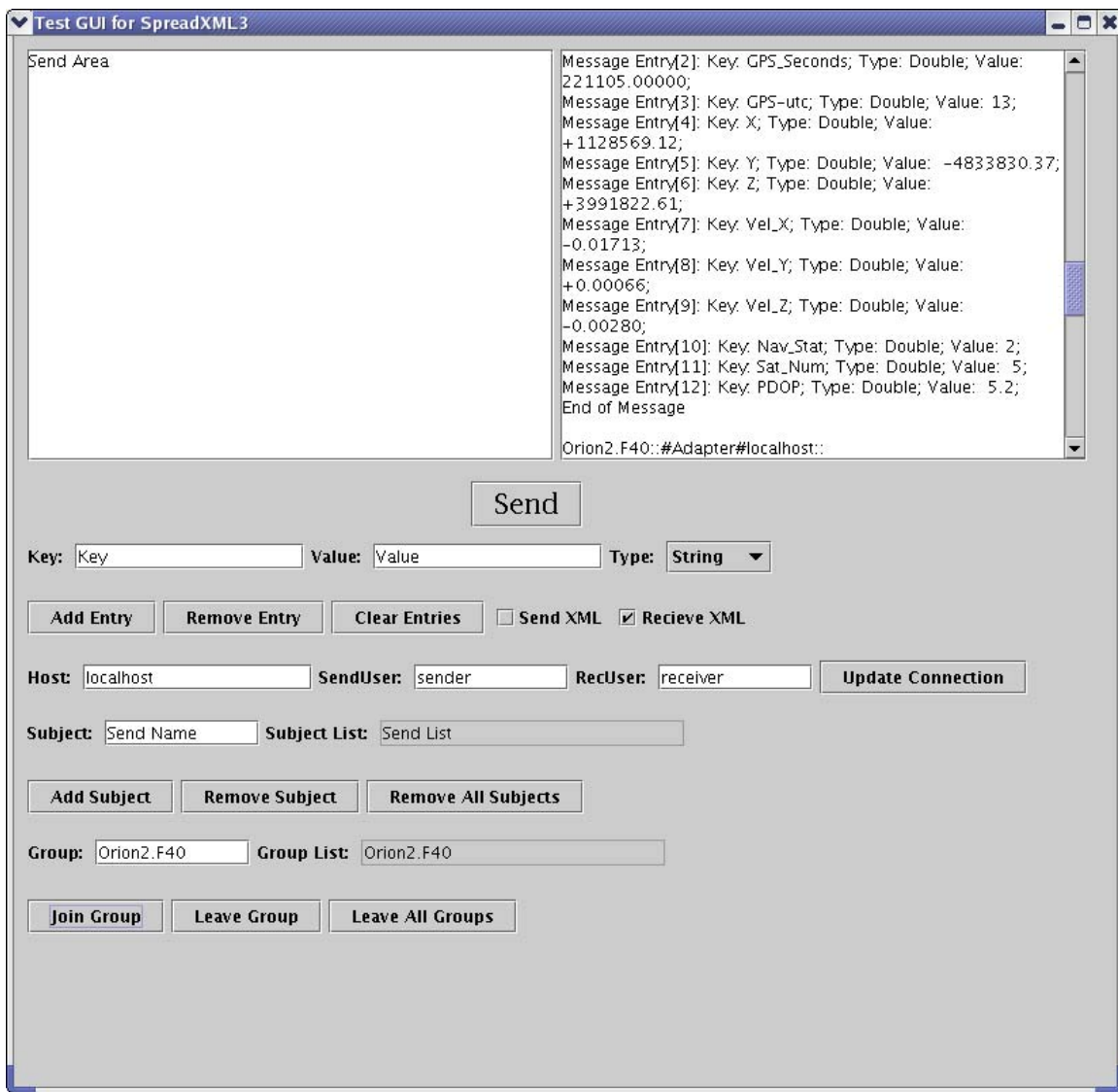


Figure 27: Screenshot of TestGUI

In addition to testing the Spread connection and middleware functionality, we also included components within TestGUI to verify the use of XML formatted text being transmitted over the middleware. The text fields and combo-box denoted by “Key”, “Type”, and “Value” are used to represent a MessageEntry as described within the message schema. These message entries could be added in the correct XML format using the “Add Entry” button. By adding entries, TestGUI would create an XML formatted string that followed the message schema. This feature of TestGUI allowed us to test the ability to send and receive XML formatted text utilizing the schema-derived classes. There was also the option to remove single entries or clear all of them. Two other features of TestGUI relating to XML are the “Send XML” and “Receive XML” checkboxes. If “Send XML” is selected then the message sender will send an XML formatted string using the actual data objects within the software representing each message entry. On the other hand, if the checkbox is not selected the sender will send the XML formatted text that was echoed into the send text area. If “Receive XML” is selected then the message receiver will parse the XML formatted string into each message, message entry, key, value, and type and display the parsed data in the receive text area. Otherwise, the data will be displayed in the exact same format that it is sent.

5.7.2 TestData Design

We designed a second test application, called TestData seen in Figure 28, with the sole purpose of sending data to various subjects. The TestData window was very basic having only the host and username text fields for the Spread connection as well as a “Start” button to begin sending the different data series. Within TestData we included several test data series to broadcast over the middleware including variations of sines, cosines, parabolas, random noise (between 0 and 1, and 0 to 20), and tangents. Once the test application sent the data in the correct XML format, which we tested using TestGUI, we were ready to test NetPlot. The main reason for TestData was to test and evaluate NetPlots’ ability to use the Spread middleware to receive data as well as parse and plot different data series. Once we demonstrated the potential to plot data, we then evaluated various features associated with the plot windows.



Figure 28: Screenshot of TestData

5.8 Summary

Our design and development path led us to create various applications and software packages in order to work toward our project goal of improving FFTBPlot, the FFTB real-time plotting application, and FFTB component communication. These goals led us to develop SpreadXML3, a software package used to implement a middleware communication scheme for the FFTB computer network. We used this software package to re-implement the network communication functionality in the Adapter/Client software, created by the 2003 WPI project team. Our attempts at working with the FFTBPlot software led us to making the major design change of abandoning FFTBPlot for a simpler, more efficient solution. This solution was NetPlot, which utilized our SpreadXML3 package and a different plotting engine, to plot data being transmitted over the middleware. We also created a Data Logger utility that logs the data transmitted over the middleware in a comma separated value file. Other software that we created consisted of simple testing applications for the middleware.

6 Implementation

The implementation of our design required the use of various tools and implementation techniques. Some of the techniques included creating interface objects, employing multi-threaded processes, and utilizing complex data objects. These techniques are explained in this chapter. This chapter also describes the programming tools and software packages that we utilized for the advanced functionality of software, such as XML interpreting and network communications. We also discuss our source code documentation standards.

6.1 Java Development

For general Java development, we utilized the Eclipse Java integrated development environment from IBM and the Java Standard Developer's Toolkit (SDK) from Sun Microsystems. We used Eclipse version 3.0, which can be found at <http://www.eclipse.org>. Eclipse is open-source software, available for both Linux and Windows operating systems. We used the Java SDK version 1.4.2, which can be found at <http://java.sun.com>. The Java SDK, which is available for several operating systems including Linux and Windows, includes a Java runtime environment and provides the functionality of the Java Application Programming Interface. We developed and compiled our software on both Red Hat Linux machines and Windows XP machines.

6.2 Uncommon Implementation Techniques

Advanced or unusual techniques not seen in many simpler applications are described first below.

6.2.1 Message Handlers and Message Receiver

The *MessageHandler* object in SpreadXML3 is a Java interface. An interface is only partially defined and must be implemented by another object in order to be instantiated. *MessageHandler* is completely empty except for a partially defined method "handle message". For this method, the return type is defined as null and the parameters are defined as a byte array ("data" – the raw form of the data), a string array ("subjects" – a list of the subjects that message was sent on), and a string ("sender" – the name of the sender of the message). The body of the function is left empty to be defined by the object that implements this interface. This allows the operation of *MessageHandler* to be completely customizable by the developer utilizing the SpreadXML3 package. Objects such as *PlotFrame* of the NetPlot package and *SourceHandler* of the Data Logger package are implementations of *MessageHandler*.

The *MessageReceiver* object of the SpreadXML3 package holds a list of objects that have implemented *MessageHandler*. However, each object can still be referred to as *MessageHandler* objects because of the Java class hierarchy. The *MessageReceiver* object runs in a loop where it listens for message on the connection to the middleware. When *MessageReceiver* receives a message, the message and its supporting information is passed to each *MessageHandler* object. The *MessageReceiver* object does this by calling the "handle message" method, supplying the appropriate values as parameters, of each of the *MessageHandlers* in its list.

6.2.2 Multi-threading and Synchronization

Multi-threading is a technique necessary when there is a requirement for multiple processes to be run at the same time. Without multi-threading, a program simply runs on single system processing thread and can only perform one action at a time. For example, if a program needs to manage its animated GUI and process data at the same time, these two tasks need to be run on separate threads or the execution of one of these tasks would cause the other task to not be executed at all. Multi-threading in Java can be performed by creating thread objects. A more advanced way of implementing multi-threading involves using the `SwingWorker` class, which is provided by the Java Swing utilities package (`javax.swing.SwingUtilities`).

Without multi-threading, we could not have created the software that we designed. It was essential for our programs to be able to perform multiple operations (such as listening for messages and responding to GUI interaction) at the same time. However, the use of threads involves understanding and carefully planning how a program operates. If multiple threads perform operations on the same object at the same time, there is the danger of the two threads interfering with each other and causing unexpected errors in an application. Synchronization is a Java concept that allows the programmer to restrict access to certain objects or methods to only one thread at a time. Synchronization acts as a safeguard to prevent multiple threads from interfering with each other, however a program that overuses synchronization can create inefficiencies or even lock up.

Our implementations of SpreadXML3, NetPlot and Data Logger employ multi-threading techniques. The *MessageReceiver* object of SpreadXML3 runs on its own separate thread to poll the Spread connection, receive messages, and distribute messages to the *MessageHandlers*. NetPlot and Data Logger are inherently utilizing multi-threading as well, because they utilize the *MessageReceiver* object. PTPlot, the plotting engine that is utilized by NetPlot also employs multi-threading in order to plot data in real-time. The DataLogger application utilizes the `SwingWorker` class in order to refresh the GUI objects in a separate thread. `SwingWorker` is a class designed for the purpose of utilizing multi-threading for GUI object updating.

NetPlot was the only application we developed that required synchronization. The *PlotFrame* object of NetPlot was associated with two threads, since *PlotFrame* extended the *PlotLive* object of PTPlot and implemented the *MessageHandler* of SpreadXML3. The *PlotLive* object is a threaded object that calls the “add points” method continuously in a loop. The *MessageReceiver* also calls the “handle message” method whenever a message is received. The “handle message” method is implemented to take the data and place it into a global object within the *PlotFrame* object. The “add points” method is implemented to take data out of that global object and plot it. These threads would interfere and cause serious errors when they both attempted to access this global object at the same time. In order to fix this problem, we used synchronization to restrict access of the global object to only one of these methods at any time.

6.2.3 Array Lists and Iterators

Array lists and iterators are special Java data types provided by the Java utilities (`java.util`) package. The array list provides an easy way to manage array objects. An array list holds a list of objects

and provides useful methods to add, remove, and retrieve objects in the list. They provide a clean method for handling a list of objects without the hassle of writing code to handle arrays. An array list can also return an iterator object to provide an even simpler method of accessing its objects. Iterators are objects that provide methods to traverse through a collection of objects very efficiently. Retrieving objects with an iterator involves calling the “has next” method to check if there are more objects to retrieve and then calling the “next” method to get that object. Calls to these methods are usually put in a loop along with the code that processes the objects retrieved.

We utilize array lists and iterators in SpreadXML3, NetPlot, and the DataLogger. In SpreadXML3, the *MessageReceiver* handles its list of references to *MessageHandlers* with an array list. The main application object of NetPlot handles its list of references to *PlotFrame* objects with an array list. The main application object of DataLogger handles its list of references to *SourceHandler* objects with an array list. By employing these useful array list and iterator objects, we sped up our development time and simplified our source code.

6.3 Middleware

To implement the middleware concept for the FFTB, discussed previously in Chapter 5, we utilized the Spread toolkit version 3.17.2 found at <http://www.spread.org>. This toolkit provided the network communications libraries and the central Spread application necessary to achieve the data communications aspect of the middleware. A major benefit of using Spread rather than other similar software is that it is open source and available in multiple languages including Java, C, C++, Perl, Python, and Ruby. Since all of our software development was done using Java we chose to use the Java Spread libraries. The toolkit also provides several programs to test and understand how the network communication libraries are implemented and utilized for different applications. In addition, the main website includes thorough documentation pages that explain the Spread toolkit libraries. Our SpreadXML3 package employs the Spread toolkit to provide an easier, more functional interface to the middleware.

6.4 XML Processing

There were many portions of our software that utilized XML formatting to either create configuration files or standardize communicated data. In order to define the formatting structure of a specific XML file type, we created an XML schema for each XML file. We wrote the XML schemas using an XML editing program called Altova XMLSpy, version 2004 release 4, found at <http://www.altova.com>. The XML schemas for our configuration files and messages are located in Appendix B. The schemas were then used to generate schema-derived interfaces and classes that represent the schema. The process of generating these interfaces and classes was accomplished using the JAXB libraries, provided in the Java Web Services Developer Pack version 1.4, to bind the schema. Our SpreadXML3 package used the auto-generated interfaces to perform different XML processing tasks such as marshalling and unmarshalling. These tasks were used to package data into a proper formatted message and then parse the data from the message into individual data objects, according to the schema.

6.5 Plotting Engines

As we worked toward accomplishing our goal of improving the simulation plotting package, FFTBPlot, we had to understand the implementation of the plotting engine, JFreeChart. Due to problems discussed previously in Chapter 5, we moved towards designing our new plotting software, NetPlot, which was implemented using a different plotting engine, PTPlot version 5.3. This plotting engine is open source, available in Java and located at <http://ptolemy.eecs.berkeley.edu/java/ptplot>. The PTPlot website also includes documentation pages that explain all of the PTPlot libraries. PTPlot includes many plotting options, but we mainly utilized the PlotLive class, which provided the ability to plot in real-time. The manner in which this package plots data in real-time is also more efficient than JFreeChart. The PTPlot plotting engine was implemented within NetPlot to provide the essential libraries for the plot windows.

6.6 Java Coding Standards

While developing our source code, we followed general Java coding standards and the NASA coding standards that we were aware of. We used the general Java standard of choosing clearly understandable multi-word names, where the first letter of each word is capitalized, for the names of our class and interface definitions (ex: “MessageReceiver”, “SourceHandler”). We also used the general Java standard to choose the names of instances of objects, methods, and variables. This standard is the same as for the classes except the first letter is never capitalized (ex: “plotFrameList”, “removeMessageHandler”, “subjectList”).

Documentation within the source code was also an important aspect of the coding standards. For each source code file, we added a header that gives the purpose of the file, authorship information, and the revision version. We provided clear documentation for each class, method, and member variable that required explanation. We also placed comments throughout complicated blocks of code to help explain the algorithm.

6.6.1 JavaDoc Documentation Standard

In addition to using the Java coding standards, we followed the JavaDoc documentation standard for documenting our software. This included adding documentation to the classes, methods, and variables in the format recognized by JavaDoc. Using the Eclipse IDE, we were able to generate a series of HTML files from the documentation in our software. The HTML files created using the JavaDoc standard, enabled us to provide a webpage describing all of the classes, methods, variables, and structure associated with SpreadXML3, NetPlot and Data Logger. The link to the JavaDoc for our software is <http://www.wpi.edu/~codyh/MQP/doc/index.html> and is also included on the CD with all of our project files.

6.7 Summary

Overall, implementing our design involved the utilization of various development tools, third-party libraries and advanced implementation techniques. These tools, libraries and techniques allowed us to implement many advanced functions into our software.

7 Testing and Evaluation

Our project had several objectives each with a set of design requirements to meet. In order to verify that our software met these requirements we had to test and evaluate software components as we developed and integrated them throughout the development process. Below we describe the testing and evaluation procedures we used for the Middleware, Adapter/Client, FFTBPlot, NetPlot and Data Logger software components.

7.1 *Middleware*

The Spread middleware concept began as simply an idea, but after we demonstrated the functionality of the Spread toolkit, the first steps towards implementing the concept were realized. We tested Spread using several steps of development, detailed previously in Chapter 5, which eventually led to the SpreadXML3 package. The basic test application we developed, TestGUI, allowed us to analyze the Spread functionality throughout different steps of developing the middleware communications package. TestGUI had to constantly be modified as more functionality, such as transmitting messages over the middleware and XML marshalling and unmarshalling, was added to our communications software.

A major test of the middleware was the reliability. We tested this by monitoring the time between data messages being transmitted across the middleware. The test ran for approximately one hour to log GPS receiver data that was being sent at a rate of 1 Hz (1 data message per second). We logged the time between receiving messages to verify that the frequency of messages was one per second. Figure 29 below shows the graph of time between messages in milliseconds versus test time. Throughout the entire test no messages were dropped and we also calculated the average time between messages to be 999.96 ms, which is sufficiently close to one second. The maximum time between messages was 1090 ms and the minimum was 939 ms, therefore all of the messages were received at nearly the same rate they were being sent.

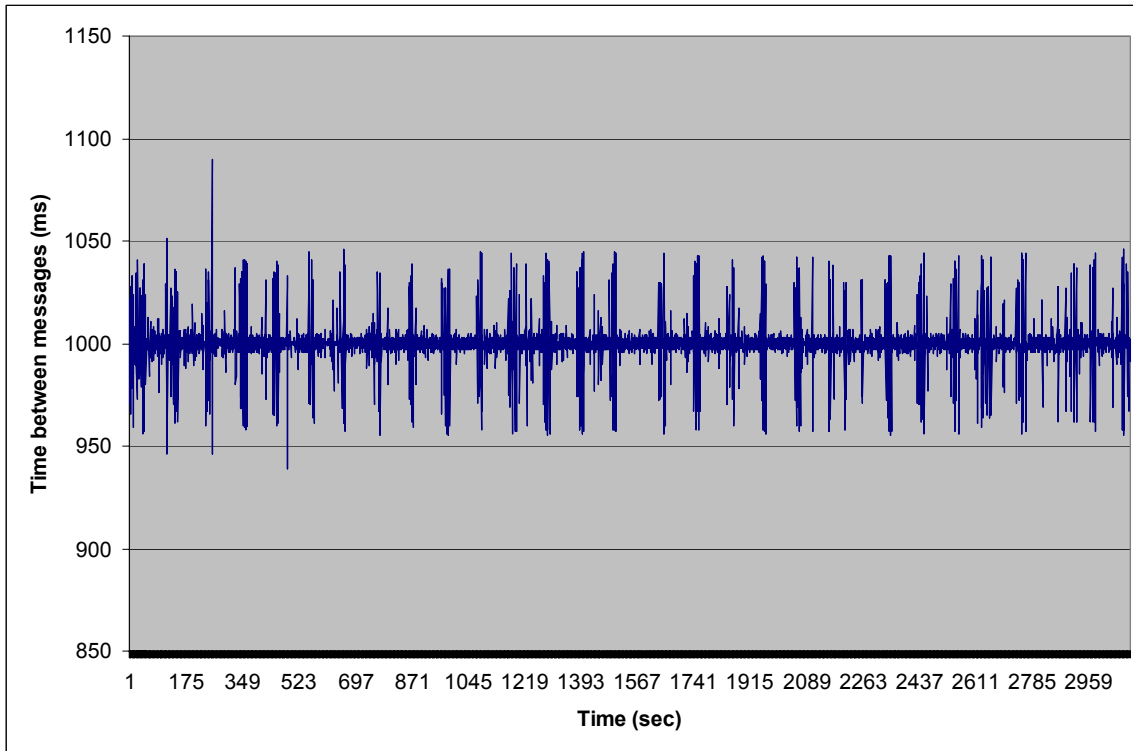


Figure 29: Message time test graph

For our source of network communications, the SpreadXML3 package was developed and met our requirements to replace point-to-point connections, send and receive specified messages, use a standard data format (XML), and provide reliability (no dropped messages). The middleware was also the basis for all network communications in our other software. Evaluations were made as different applications utilized the function of the middleware. Overall, the Spread middleware has proven reliable, efficient and flexible.

7.2 Adapter/Client

After establishing SpreadXML3 as our communications package we were able to begin modifying the Adapter/Client software. The first step in our modifications was to integrate the SpreadXML3 package into both the Adapter and Client. We removed the old point-to-point connections and replaced them with instances of our SpreadXML3 classes to handle communications through the middleware. To verify we met the design requirement of using the Spread middleware for communication, we ran a test on the Adapter/Client by initializing the Orion GPS receivers and then ran the Adapter and Client. After working out a few bugs, the Adapter displayed the GPS data properly and the Client accurately received and displayed the data being sent through the middleware by the Adapter.

Another requirement for the Adapter/Client was to allow support for Pivot GPS receivers and have flexible serial communications to be able to interchange between the Pivot and Orion GPS receivers. Since the Pivot receivers transmitted similar data with minor formatting changes compared to the Orion data, minimal modifications to the Adapter had to be made. The Adapter was able to recognize the difference between the GPS receivers and handled the messages accordingly. We were able to test both

the Adapter and Client by connecting one Orion receiver and one Pivot receiver to the serial ports of the Adapter/Client computer. Next, we switched which serial port each was connected to in order to verify each serial port and respective Client display could handle both types of GPS receiver.

To confirm that the Adapter/Client handled the different receivers messages properly we examined the data being displayed on the Client as well as the raw data (which included the subject the message was sent to) being transmitted using our TestGUI application. In conclusion, we were able to add the support for the Adapter/Client to handle the Orion and Pivot GPS receivers interchangeably and without having to worry about different setups for each specific receiver. This benefits the test bed such that if either type of receiver is connected, the Adapter broadcasts Orion and Pivot data through the middleware and the Client “subscribes” to and displays both types of data without any problems.

7.3 FFTBPlot

While attempting to improve the simulation plotting package, FFTBPlot, we were able to meet the requirement of adding the network communications using Spread. However, after testing the ability to plot data received through the middleware we concluded that the plotting engine within FFTBPlot was insufficient for performing real-time plotting. In addition, we also evaluated the overall performance of FFTBPlot and realized that developing a new plotting package from scratch would save the trouble of resolving bugs and replacing the plotting engine for FFTBPlot. For this reason, we began the development of our new plotting software, NetPlot, which utilized another plotting engine, PTPlot, that was able to handle real-time plotting.

7.4 NetPlot

In our development of Netplot we had several requirements to meet that were similar to the original objectives for improving FFTBPlot. The first step in meeting our requirements was to create a simple and intuitive GUI that we tested by viewing the layout and seeing how easy all the features were to understand and use. Once we had developed the software to receive and plot data using the middleware, we began testing the PTPlot plotting engine and the data communications. We used TestGUI to monitor the data being transmitted over the network to verify it was correct. To test the plotting capabilities we ran various tests, which included “subscribing” to numerous “subjects”, plotting several combinations of data, creating plots with different available formatting options, and analyzing the updating of plotted data points. We found that the plotting engine was able to handle displaying data accurately and at the same rate as messages were being received without lagging. However, our original successful tests were performed on a Linux machine. When we tested the software on a Windows-based machine we noticed lagging if multiple plot windows were open. We discussed this issue with our mentors and concluded that it must be related to either the Windows Java runtime environment or possibly a Windows threading issue. The actual source of the problem is still unidentified, but it does not pose a concern because a majority of the machines in the lab are Linux-based and NetPlot ran smoothly through the tests on those computers. Since we isolated the cause of the problem to Window machines, our mentor stated that NetPlot could be run solely on Linux machines.

After testing the main objective for NetPlot to communicate using the middleware and efficiently plot in real-time, we moved towards the requirement proposed by our mentors of command line execution

and using XML files to save and load plotting configurations. We first tested the saving and loading of configuration files since the command line execution would use these files as arguments to load on start. The testing of configuration files consisted of using the NetPlot configuration schema to store all of the information regarding the Spread connection, plotted data, plot formatting options, and plot window. We used recommendations by our mentors and other test bed engineers to determine the final information that needed to be stored by the configuration file. Once we created the schema, we tested the configurations by setting up a demonstration NetPlot and then saving the configuration. We then analyzed the saved file to ensure that all of the necessary information was stored properly. The final test was to verify that the configuration could load correctly into NetPlot. The loaded configuration proved to work by opening all of the saved windows in the same location and size, setting up the middleware connection, and receiving and plotting the appropriate data.

Upon validating the capability to save and load configuration files, we approached testing the command line script files. This was straightforward since we had to make sure we provided the proper locations for classpaths and the executed program within the script file. The script allowed for the configuration files to be passed in as arguments for the run command, enabling NetPlot to startup and automatically load the given configuration file(s). We tested the script file by using the command with or without arguments to ensure it worked either way. The command line execution met our expectations and provided a simple one-line command to run NetPlot with or without a specified configuration.

7.5 Data Logger

Similar to NetPlot, Data Logger was tested and evaluated in the same manner for the design requirements of saving and loading configuration files and command line execution. Before we tested those aspects, we had to meet other requirements for Data Logger including logging data in a simple and accessible format for specified subjects and time stamping the data. We chose to save the logged data in the comma-separated-values format, which proves to be simple and accessible because the data can easily be opened and separated using Microsoft Excel or similar spreadsheet software. We ran several tests using Data Logger to log data from various subjects including the different messages from both the Orion and Pivot GPS receivers. The log files were then analyzed to make sure the data was formatted correctly and could be easily opened in Excel. One feature we added to the main window of Data Logger was the ability for the user to view the active logger file information. We added the information for each active logger to display the number of received messages and log file size so the user could notice any errors regarding the logger file.

7.6 Summary

Throughout the design and implementation process we tested and evaluated our software to ensure the proper functionality and operation to the design requirements that were established. With the exception of FFTBPlot, we were able to meet the design requirements for the middleware, Adapter/Client, NetPlot and Data Logger. Since the Spread middleware still remains in the first stage of being implemented for the communications within the formation flying test bed, further testing and evaluation are necessary.

8 Conclusions

This chapter presents our project conclusions that analyze how well we accomplished our goals and also presents recommendations for future work.

8.1 *Accomplishment of Our Goals and Objectives*

We accomplished a significant amount of work that benefited the Formation Flying Test Bed. The individual software applications met their respective design requirements as documented in the Chapters 5 and 7. However, we did not completely adhere to our original goals and objectives. Our first original goal to enhance the FFTBPlot software led us to develop the SpreadXML3 package. Instead of modifying FFTBPlot, we created a new plotting application called NetPlot. NetPlot accomplished the same design requirements that were specified for the FFTBPlot software modifications. We did not attempt our secondary goal of working on the post-processing tool suite, because we decided to work on other pieces of software, such as the Data Logger and the 2003 WPI Adapter/Client.

8.2 *Recommendations*

The vision of the middleware design concept and the impact of our project work extend beyond the brief ten weeks that we had at the Goddard Space Flight Center. Our project is not a standalone product but only a piece of the larger picture of the FFTB. The actions taken by others to utilize, integrate, and improve our software determine the benefits that the FFTB will receive from our work. Therefore, we have given many recommendations that specify some of these methods.

8.2.1 **SpreadXML3 Recommendations**

The development of the SpreadXML3 package is the first step toward the complete re-implementation of the communications with the Formation Flying Test Bed. The design concept of the middleware will eventually replace the various point-to-point connections between components. SpreadXML3 is fully functional, however for it to assume the role as the application programming interface to the middleware it must be even better in terms of optimization, intuitiveness, extensibility, and range of functionality. Therefore, we have some recommendations for future work to improve the SpreadXML3 package.

- **Develop a C++ version of SpreadXML3 that provides a similar, if not identical, interface to the middleware**

Developing a C++ version of SpreadXML3 would allow applications, written in the very common C++ programming language, to use the middleware for communications. There are FFTB components written in C++ and there is always the possibility that a new component being added is written in C++. This C++ version should also be similar, if not identical, in object structure and the location of functionality in the various objects. Similarity of the two versions will prevent confusion and reduce the learning curve when a developer utilizes the C++ SpreadXML3 after having worked with the

Java SpreadXML3. At the time of the authoring of this paper, an FFTB software developer is working on this task.

- **Redesign and optimize the message structure to reduce the message size**

The XML messages sent over the middleware are unnecessarily bulky. An example an XML message can be found in Section 5.2.3 of this report. A possible way to reduce the size of the message would be to reduce the size of the tags (ex: “Message” can be replaced with “M” and “MessageEntry” can be replaced with “E”). At the time of the authoring of this paper, an FFTB software developer created a new message structure and is working on adding it into SpreadXML3.

- **Define a naming standard for the subjects used for communications across the middleware**

A naming scheme for the various subject names that the data is sent over should be devised. This recommendation does not require any source code modifications to SpreadXML3; instead it is only a recommendation about how it should be utilized. In the Adapter software we used the naming scheme of “[GPS receiver type][receiver port number].[GPS message type]” (ex: “Orion1.F40”). A more appropriate naming scheme might utilize subject names that look like “[simulation name].[spacescraft id].[data message type]”.

- **Create a standard set of “key” names used to format the data sent across the middleware**

Setting a standard set of “key” names for the various pieces of data for each specific message would also be beneficial. This would mean that for each subject, a master list of the possible keys used in messages in that subject would be created. This master list would be a standard that all senders to that subject would have to follow. There would be different methods of implementing this scheme. A simple method would be to have a paper list that programmers consulted before they set up an application to send or receive messages on a subject. A more complex method could involve creating a message-validating piece of software that messages must be passed through before broadcast over the middleware.

- **Test the extremes of the middleware performance**

The extremes of the middleware performance have never been thoroughly tested. However, reaching these extreme limits of performance could be difficult even with a carefully set up experiment. The Spread middleware is a very robust application and theoretically designed to handle an unbounded number of clients. Also, the FFTB lab has a 1-gigabit network. A test designed to push the limits of the network and the Spread middleware could offer compelling proof of its capabilities.

8.2.2 NetPlot Recommendations

NetPlot sufficiently meets its design requirements. There are only a few recommendations for NetPlot that we can devise. These recommendations are for the FFTB software developers or future WPI project teams.

- **Investigate the performance lag exclusive to Windows machines**

As we mention in the Testing and Evaluation chapter, the NetPlot application has performance issues when running on the Microsoft Windows operating system. A severe performance lag occurs when multiple plot windows are open. This issue does not occur when run on the Linux operating system, which is where the program will be used exclusively. It might be worthwhile to investigate and fix this performance issue. Some software developers at the FFTB lab suggested that this problem might be caused by the inefficiency of the Windows version of Java Virtual Machine or the method in which Windows handles multi-threading. Both of these possibilities should also be investigated along with analyzing the source code.

- **Improve the GUI**

Although the GUI is easy-to-use and intuitive, we believe that improvements can always be made. Some examples would be to redesign the layout or add support for hot-keys to speed up the activation of commonly performed actions. However, changing the layout could cause confusion to users familiar with the original application. Also, an overly complex GUI design should be avoided. The GUI should only be changed in a manner that makes it more intuitive and easier to use.

8.2.3 DataLogger Recommendations

The Data Logger is a very simple application with the most basic functionality. This is because we developed this utility application near the end of the project and did not have the time to add all of the features that we would have liked. This application adequately performs its task, but it can still be improved to accomplish even more. The follow recommendations are either for the FFTB software developers or future WPI project teams.

- **Enable the feature of selective data logging as opposed to logging all of the data**

The Data Logger simply logs all of the data being sent over a specified subject, including sender information and a timestamp. The logger writes the data in an efficient format, but the files can still become quite large. A useful feature for the data logger would be the option to filter out data variables that do not need to be logged or filter messages based on the source of the message. An option to log only once in a specified interval of time would also be useful.

- **Enable the ability to choose the log file format**

The Data Logger by default uses the comma-separated-values format, where each value is separated with commas and each message is put on a separate line. Allowing the user to choose the file format of the log file would be a useful feature.

- **Create the feature to read a log file and rebroadcast the logged data**

Reversing the function of the Data Logger to read a log file and rebroadcast the data could be a useful feature as well. This feature could be used to setup a demonstration or test the functionality of another part of the FFTB connected to the middleware. Useful options for this feature could include

setting user defined pauses between sent messages or utilizing the timestamps to send the messages at the same rate that they were logged at.

- **Improve the GUI**

Similar to the recommendations for the NetPlot application, we believe that the GUI can always be improved. The layout is pretty simple but could be restructured. Also, hot-keys to perform common tasks could also be supported.

8.2.4 General Recommendations

We also have some general recommendations that do not directly fall under any of the above applications. These recommendations specify general future work or general objectives that relate to our project and could be beneficial to the FFTB. These recommendations are for the FFTB software developers or future WPI projects teams.

- **Create a top-level application to facilitate the start up of the multiple pieces of software used in an FFTB simulation run**

This top-level would control and automate the launch of all of the various components and software used in an FFTB simulation run along with the NetPlot and DataLogger software that we created. This would be of great benefit to the FFTB and improve their productivity. NetPlot and DataLogger can both be launched from the command line along with arguments that specify the configuration files to use. Therefore, the automated launch of the software that we created should be relatively simple. However, devising a method to automate all of the FFTB components, which are situated on different computers and pieces of hardware, may be challenging. Our mentor suggested that this task alone could be the subject for a future WPI project team.

- **Develop a real-time data manipulation application**

The concept of a real-time data manipulation utility application was discussed with our mentors, however we did not have time to approach this task. Such an application would have the functionality to receive messages on a specified subject, modify values as specified and then rebroadcast to the middleware on another specified subject. The design of this application would have to be carefully thought out in order for it to be flexible for many situations. This application could replace the need for data manipulation functionality in many other components of the FFTB.

- **Document all new source code with the JavaDoc standard**

We employed the common practice of documenting our code with the JavaDoc standard. This allows useful help pages for your source code to be auto-generated by the Eclipse IDE. We strongly suggest that all WPI project teams document their Java source code with the JavaDoc standard. Using JavaDoc and creating the help pages will make source code much easily understood for future development.

8.3 Summary

The software developed within the context of this project met the requirements that were expressed for the design and included additional features that could be used to improve the software when the design process is continued. Overall, our project experience at the Goddard FFTB has been challenging and beneficial to developing several skills such as software development, technical writing, group working, and presentation skills.

9 References

1. Ahalt, D. A., *About GSFC – In Depth – Contributions*, <http://www.gsfc.nasa.gov/indepth/about_contributions.html> (viewed on April 5, 2004).
2. Ahalt, D. A., *Goddard's Mission*, <<http://pao.gsfc.nasa.gov/gsfcc/welcome/mission/mission.htm>> (viewed on April 5, 2004).
3. Andren, M., Castiglione, J., Woodhull, J., *GPS Formation Flying Testbed GUI and Translator*. Worcester; Worcester Polytechnic Institute Major Qualifying Project Report.
4. Armstrong, Eric (December 2001), *A Quick Introduction to XML* <http://java.sun.com/xml/jaxp/dist/1.1/docs/tutorial/overview/1_xml.html> (viewed on August 16, 2004).
5. Beard, R. W., Ren, W. (June 2003). "A Decentralized Scheme for Spacecraft Formation Flying via the Virtual Structure Approach". *AIAA Journal of Guidance, Control, & Dynamics* <http://www.ee.byu.edu/grad1/users/beard/www_docs/papers/preprints/RenBeard03c.pdf> (viewed on April 19, 2004).
6. Diaz, Alphonso V., *GSFC – About Goddard*, <<http://www.gsfc.nasa.gov/about.html>> (viewed on April 6, 2004).
7. Eckel, B., (2000) *Thinking In Java: The Definitive Introduction To Object-Oriented Programming In The Language Of The World Wide Web*, 2nd Ed. New Jersey; Prentice Hall.
8. Ferguson, P. (September 24,2001). *Agent Based Software for the Autonomous Control of Formation Flying Spacecraft* <<http://hills33.mit.edu/OA/>> (viewed on April 21, 2004).
9. Garber, S. J. and Launius, R. D., *NASA History Fact Sheet*, <<http://www.hq.nasa.gov/office/pao/History/factsheet.htm>> (viewed on April 5, 2004).
10. GSFC, *Implementing NASA's Strategies for the 21st Century*, <<http://www.gsfc.nasa.gov/indepth/gsfccplan.pdf>> (viewed on April 5, 2004).
11. Hardee, M., Smaragdis, M, et al., *What Is Java Technology?* <<http://java.sun.com/java2/whatis/>> (viewed on April 18, 2004).
12. Leitner, J., F. Bauer, D. Folta, M. Moreau, R. Carpenter, J. How (February 1, 2002). "Formation Flying in Space", *GPS World*, <<http://www.gpsworld.com/gpsworld/article/articleDetail.jsp?id=9518>> (viewed on April 19, 2004).
13. Lynn, J., *About GSFC – In Depth – Dr. Goddard*, <http://www.gsfc.nasa.gov/indepth/about_drgoddard.html> (viewed on April 5, 2004).
14. Kane-Hager, Lisa. *GTMO About* <<http://gsfctechnology.gsfc.nasa.gov/about/index.html>> (viewed on April 6, 2004).
15. Kane-Hager, Lisa. *Distributed Space Systems* <<http://gsfctechnology.gsfc.nasa.gov/focusAreas/dss.htm>> (viewed on April 6, 2004).

16. MathWorks Inc., *Matlab 6.5.1 Product Description*
<<http://www.mathworks.com/products/matlab/description1.html>> (viewed on April 20, 2004).
17. McInnes, C., McQuade, F. *Spacecraft Formation-Flying Control*
<<http://www.aero.gla.ac.uk/Research/Ss/form.htm>> (viewed on April 21, 2004).
18. NASA – GSFC. (June 4, 2001). News Release: “Satellite formation flying concept becoming a reality”. <<http://spaceflightnow.com/news/n0007/04satformation/>> (viewed on April 6, 2004).
19. NASA, *NASA History In Brief* <<http://history.nasa.gov/brief.html>> (viewed on April 5, 2004).
20. Ort, Ed and Bhakti Mehta (March 2003), *Java Architecture for XML Binding (JAXB)*
<<http://java.sun.com/developer/technicalArticles/WebServices/jaxb/index.html>> (viewed on August 16, 2004).
21. Saraf, S., A. Knoll, F. Pelletier, M. Tafazoli. (2002). “Investigating Formation Flying and COTS in an Integrated Simulation Environment”. <<http://www.spaceops2002.org/papers/SpaceOps02-P-T2-16.pdf>> (viewed on April 19, 2004).
22. Smithsonian Institution, *How Does GPS Work?* <<http://www.nasm.si.edu/galleries/gps/work.html>> (viewed on April 21, 2004).
23. Spaceflight Now. *Birds inspire formation-flying satellites.*
<<http://spaceflightnow.com/news/n0007/04satformation/>> (viewed on April 6, 2004).
24. Tillerson, M., L. Breger and J. P. How (June 2003) “Distributed Coordination and Control of Formation Flying Spacecraft” <http://hills33.mit.edu/papers/ACC_finalsubmission.pdf> (viewed on April 17, 2004).
25. Trimble, *All About GPS Tutorial* <<http://www.trimble.com/gps/>> (viewed on April 20, 2004).
26. Turner, Dave, *Global Positioning System Current Status and Modernization Efforts*
<<http://www.oosa.unvienna.org/SAP/act2002/gnss1/presentations/session01/speaker01/>> (viewed on April 21, 2004).
27. World Wide Web Consortium (December 1997), *Extensible Markup Language (XML)*
<<http://www.w3.org/TR/PR-xml-971208>> (viewed on August 16, 2004).
28. Zandonella, C., “Follow My Leader”, *New Scientist*. (Sept. 25, 1999)

Appendix A: WPI 2004 Software User Manual

Software Description

The software that this manual deals with is an application for real-time plotting of data outputted to the Spread middleware and a data-logging program to monitor the data. The software we have created utilizes a common network communications library called the Spread Toolkit. This library provides the middleware for which the data is sent and received between different applications.

The real-time plotting software is called NetPlot. This application receives messages, specified by the user, sent over the middleware and determines what data series can be plotted. The user then chooses the data to plot and the format for each plot window

The data logging application is simply called Data Logger. The program is basically used to monitor data that is sent through the middleware. The user specifies which data to log and the application will log the data to a file as well as keep track of important file information.

In order to use the software developed by the WPI 2004 project team there is required software that needs to be installed. The required software and details of using NetPlot and DataLogger are discussed later in this manual.

Requirements

To run the applications developed for plotting and logging data several essential software packages must be installed. The mandatory software includes the following:

- The Java 2 Standard Edition (J2SE) Software Development Kit (SDK) version 1.4.2.x or higher, located at:
<http://java.sun.com/j2se/1.4.2/download.html>
- The Java Web Services Developer Pack (JWSDP) version 1.4, located at:
<http://java.sun.com/webservices/downloads/webservicespack.html>
- The basic Spread Toolkit, located at:
<http://www.spread.org/download.html>
- Ptpplot 5.3 – Java Plotter, located at:
<http://ptolemy.eecs.berkeley.edu/java/ptplot5.3/ptolemy/plot/doc/index.htm>

The Java 2 SDK is required since it provides a complete environment for applications development in the Java programming language. The package also includes necessary tools such as a compiler, debugger, and the Java 2 Runtime Environment (JRE). The JWSDP is needed to utilize the Java Architecture for XML Binding (JAXB) libraries as well as the general JWSDP shared libraries. The Spread Toolkit and Ptpplot also obtain necessary libraries for our applications.

Spread Middleware

The main source of network communication between our applications and the Formation Flying Test Bed (FFTB) components is provided by the Spread Toolkit which acts as the middleware for sending and receiving data. To begin using Spread the user must first set up the configuration file, named `spread.conf`, located in the `spread-bin-3.17.2/linux` directory. In this file the user needs to specify the host

computer that spread is running (currently ‘agentsmith’) as well as the port number (currently 4803). The configuration file only has to be changed if spread is run on a computer other than ‘agentsmith’. Once the configuration is complete the spread.exe file, also located in the spread-bin-3.17.2/linux folder, must be run to start the spread middleware.

NetPlot Plotting Software

NetPlot is located on ‘agentsmith’ in the /home/wpi/workspace/java_suite/ directory. We created a script file to allow NetPlot to be easily run from the command line. From the command line the user must be in the java_suite directory and enter the following:

Run_NetPlot.sh [configuration file1] [configuration file2] ...

The “[configuration file[n]]” argument(s) is a feature that allows the user to specify one or more configurations to be loaded on the start of NetPlot. The configuration files are in XML format and created by the user while NetPlot is running. The current default directory for these XML files is located at java_suite/samples. The Run_NetPlot.sh script is capable of starting NetPlot with or without the configuration file arguments.

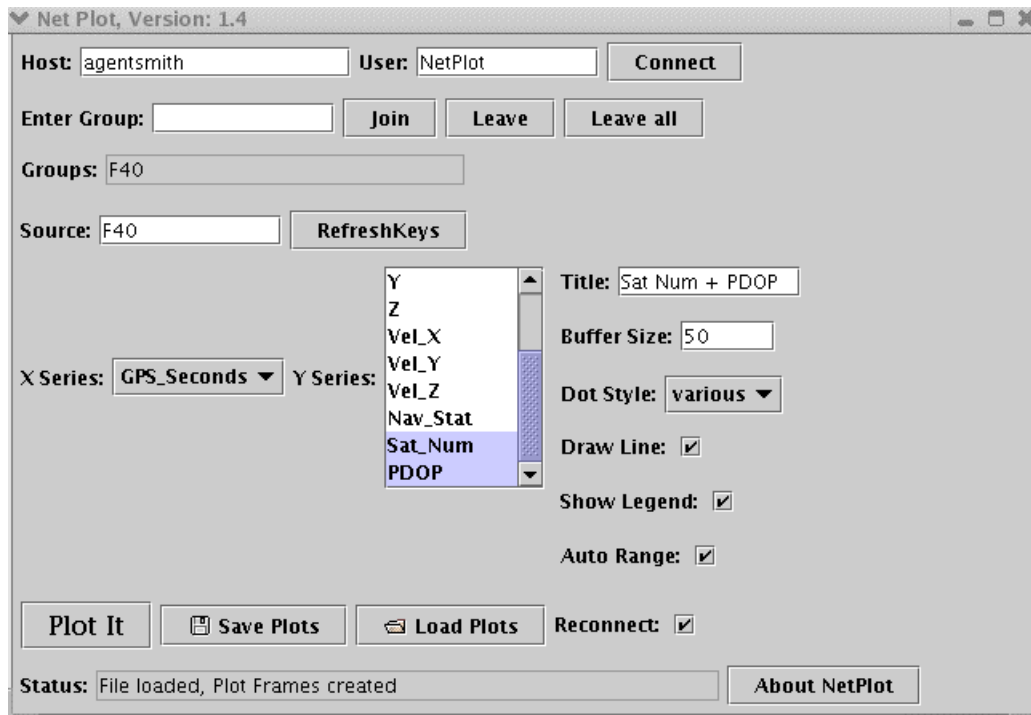


Figure 1: Main NetPlot GUI window

After starting the NetPlot application, shown above in Figure 1, there are various buttons and fields that are used to establish the spread connection, configure the plot windows, save/load configurations, and monitor the status of the application. All of the buttons and fields reveal an associated tool tip when the mouse is hovered over them to describe the use of each. The top three lines of the graphical user interface (GUI) represent the necessary parameters for establishing the connection to the spread middleware. The first step is to specify the Host, which refers to the host computer (default is ‘agentsmith’) where the spread middleware is running. Then the user must create a unique username for the NetPlot application to register on the spread middleware. If another instance of NetPlot is running the

user must change the username to something other than the default of “NetPlot” or a spread error will occur because two applications are attempting to register under the same username. To connect NetPlot to the spread middleware the user must next click the “Connect” button.

The second line from the top is where the user can subscribe to different groups (the subjects that messages are sent to). The buttons to the right of the “Enter Group” text field allow for joining and leaving one or all groups. Below this line of the GUI is an unchangeable field to display the current NetPlot group subscriptions.

The text field denoted “Source” is where the user enters the group they want to receive and plot data from. Then by clicking the “Refresh Keys” button, NetPlot will automatically receive and analyze a message from this group and update the X and Y series with the data series that are capable of being plotted.

Once the data series have been loaded the user can then select one X series and one or more Y series to plot. To plot multiple Y series hold the Ctrl key or Shift key and select the desired series. To the right of the X and Y series are several plot window formatting options including plot title, buffer size, dot style, draw line, show legend, and auto range. The title corresponds to the title for individual plot window and buffer size relates to the number of points to store in the buffer. If the user would like to plot all incoming data without losing any previous points then a ‘0’ must be entered in this field to represent an infinite buffer. The dot style option allows the user to choose which style they would like associated with each data point. Draw line specifies whether or not each consecutive data point is connected with a line. The show legend and auto range checkboxes give the user the option to provide a legend and to automatically scale the plot to the incoming data. If the user chooses not to auto range, then they have the capability of using zoom features and setting their own X and Y ranges within the individual plot window.

Upon establishing the connection to the spread middleware and setting the different options for the plot window, the user must then click the “Plot It” button to create the individual plot window. Multiple plot windows can be created by changing the X and Y series, the data sources, and different formatting options and then clicking the “Plot It” button again. An example plot window is depicted in Figure 2 below.

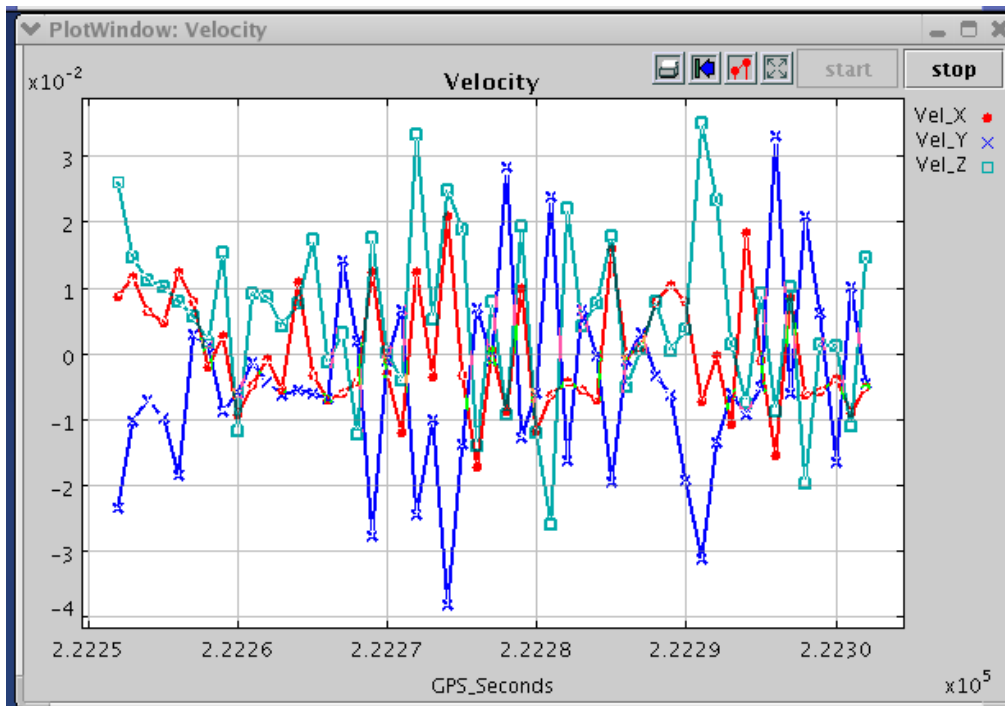






Figure 2: Example plot window

Next to the “Plot It” button are two buttons for saving and loading the configuration. Clicking the “Save Plots” button will create an XML configuration file, stored and named to the user specifications, that represents the current configuration of the spread connection and all open plot windows. For each plot window the configuration saves the X and Y series, all of the plot formatting options, and the size and position of the window. Clicking the “Load Plots” button will open the chosen configuration file and create all of the plot windows exactly how they were stored. If the “Reconnect” checkbox is selected NetPlot will also reconnect to the spread middleware. When running NetPlot from the command line these XML configuration files are the configuration file arguments that can be added to the Run_Netplot.sh command to automatically load a configuration on startup.

At the bottom of the main NetPlot GUI is a status box that displays important action and error messages to notify the user of NetPlots’ status. There are several messages that are displayed and each corresponds to a specific action or error, each of which is explained in the later section Status Box Messages. Next to the status box is also an “About NetPlot” button. This pops up a window that describes the application, any known issues, and a few usage notes.

Now that the use of the main NetPlot window has been described there are features within the individual plot window that also need to be discussed. In the top right of the plot window, as seen above in Figure 2, are six buttons that are part of each plot window. The two buttons, Start and Stop, are self explanatory such that they start and stop the plotting of data. The four smaller icon buttons relate to different options of the plot. The first on the left, , allows the user to print the plot and contains all of the different options for printing (color, print to file, size, layout, etc.). The second button from the left, , allows the user to resize the plot to its original X and Y ranges. The third button, , opens a window to change several of the plot format options including title, X and Y labels and ranges, dot style,

turning on the grid, stems, color, and line connect, and creating custom ticks for the X and Y axes. As a note, the X and Y ranges can only be changed if auto ranging was deselected, otherwise the axis ranges will continuously be modified according to incoming data. The fourth button, , provides the ability to auto-scale the plot to fit the current data. This auto-scale option is not necessary when the plot is already auto ranging. Another feature of zooming in and out within the plot window is only useful when auto ranging is turned off. This feature is done by clicking on the plot and dragging up to zoom out and down to zoom in.

Data Logger Utility

The Data Logger utility and its' required files are located on 'agentsmith' in the /home/wpi/workspace/java_suite/SpreadXML3/util directory. We created a script file to allow Data Logger to be easily run from the command line. From the command line the user must be in the java_suite directory and enter the following:

Run_Logger.sh [configuration file1] [configuration file2] ...

The "[configuration file[n]]" argument(s) is a feature that allows the user to specify one or more configurations to be loaded on the start of Data Logger. The configuration files are in XML format and created by the user while Data Logger is running. The current default directory for these XML files is located at java_suite/xml. The Run_Logger.sh script is capable of starting Data Logger with or without the configuration file arguments.

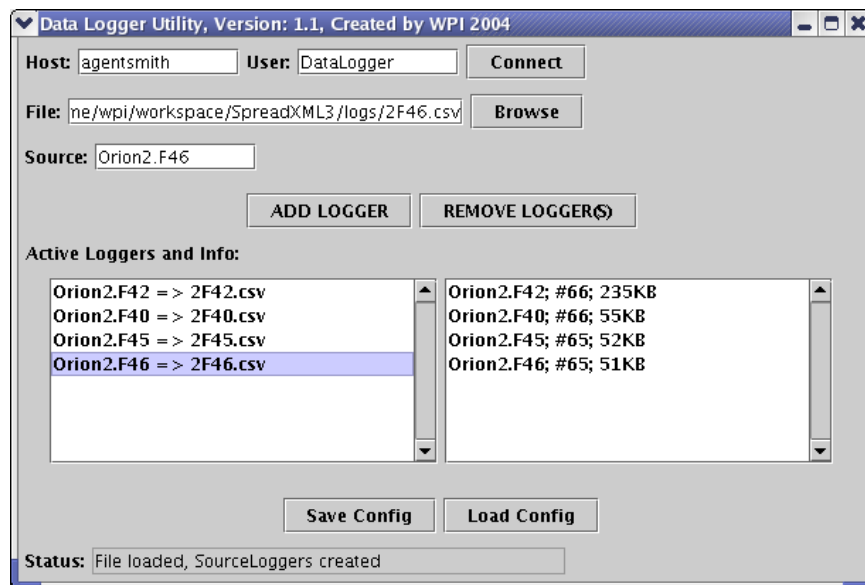


Figure 3: Main Data Logger GUI window

Once the Data Logger application is started the window shown above in Figure 3 will appear. Similar to NetPlot there are many buttons and fields to establish the spread connection, set the location for the log file, identify the source to log, view the loggers and file information, save and load configurations, and monitor the applications status. Each button and field has an associated tool tip to describe its' function. The top line of the Data Logger GUI is the same as NetPlot such that the user specifies the host

computer running spread, a unique username for the Data Logger application, and then clicks the “Connect” button to establish the connection to the spread middleware.

The text field labeled “File” is where the user identifies the location and file name to store the logged data. The file type should be a comma separated values file (.csv) since the data logger stores the data using the comma delimiter between values. Microsoft Excel can then open the .csv file easily and separate the values into columns by their message entry keys. The file can be stored in another format but .csv is the most useful since the data is comma delimited. Next to the text field is the “Browse” button to assist the user in saving the log file to the proper location.

After setting the log file name and location the next step is to choose the source (spread group) to receive data from. The source is the subject to which messages are sent to the spread middleware. The given source can then be added to the list of loggers by clicking the “Add Logger” button. Each logger is then placed in the active loggers list with the label “Source => file name”. Next to the active logger list is also information regarding the log file. The label in the file information list is in the format “Source; # of messages; file size in KB”. To remove a logger the user can select one or multiple loggers from the active loggers list and simply click the “Remove Logger(s)” button.

Below the active loggers and file information lists are two buttons for saving and loading Data Logger configurations. They work similar to the configurations in NetPlot in the sense that they store the spread connection and information regarding each active logger in an XML configuration file. Each logger also has the associated source, file and path name stored along with them. To save the configuration the user simply clicks the “Save Config” button to save all the current loggers. The current default directory where the log files are saved is the ‘logs’ folder within the main ‘java_suite’ directory. The configurations can also be loaded using the “Load Config” button or by adding the XML file as an argument when Run_Logger.sh from the command line.

Another similarity to NetPlot is the status box at the bottom of the window. The status box displays action and error messages to let the user know what is happening within the application. These status box messages are described in the next section of this manual.

Status Box Messages

- **NetPlot**

- Connection Established** – Displayed when the connection to the middleware is established using either the settings loaded from the configuration file or manually setting up the connection on the main NetPlot GUI
- Data from File was null** – Occurs when loading a NetPlot configuration file that is empty
- Done updating source keys** – Displayed after NetPlot updates the data series from the user specified source without error
- Error joining groups specified in File** – Occurs when unable to join spread groups saved in the NetPlot configuration file
- Error joining source** – Occurs when unable to join the specified source (spread group)
- Error rejoining groups after reconnect** – Occurs when loading a NetPlot configuration file and using the reconnect option and unable to join the saved groups in the file
- File loaded, Plot Frames created** – Displayed after loading the NetPlot configuration file and creating the plot frames stored in the file
- File Not Found** – Occurs when the specified file to load is not found
- File Not Opened: Cancelled by user** – Occurs when the user closes the File dialog before opening a NetPlot configuration file
- File Not Saved: Cancelled by user** – Occurs when the user closes the File dialog before saving a NetPlot configuration file
- joinGroup button error** – Occurs when unable to join the specified group in the “Enter Group” text field
- leaveAllGroups Error** – Occurs when unable to leave all groups that NetPlot is subscribed to using the “Leave all” button
- leaveGroup button error** – Occurs when unable to leave the specified group in the “Enter Group” text field
- Saved file at: “File path”);** - Displayed to let the user know where the NetPlot configuration file has been saved
- Spread Error: Connecting** – Occurs when unable to connect to the spread middleware using the main NetPlot GUI, most likely because spread is not running or the specified host is incorrect
- Spread Error: Connecting from File settings** – Occurs when unable to connect to the spread middleware using the NetPlot configuration file settings, most likely because spread is not running or the settings are incorrect
- Unknown host** – Occurs when the specified host for the spread middleware is incorrect in the “Host” text field
- Unknown host in loaded File** – Occurs when the specified host for the spread middleware is incorrect in the loaded NetPlot configuration file
- Unmarshalling Error** – Occurs when loading a configuration file that is invalid according to the NetPlot configuration schema
- Updating source keys...** – Displayed while NetPlot is updating the data series and after receiving a message for the specified source from either the main NetPlot GUI or a loaded configuration file
- Waiting for message to update keys...** – Displayed when NetPlot is waiting for a message from the specified source to update the data series

XML Parse Error – Occurs when loading an XML configuration file and NetPlot is unable to parse the XML file, most likely because the XML file doesn't follow the message schema

- **Data Logger**

Data from file was null – Occurs when the Data Logger configuration file is empty

Error joining group for logger from File – Occurs when unable to join spread group for Data Logger application saved in the Data Logger configuration file

Error joining groups from File - Occurs when unable to join spread group(s) for individual log files saved in the Data Logger configuration file

Error rejoining groups after reconnect – Occurs when loading a Data Logger configuration file and unable to join the groups in the file

File loaded, SourceLoggers created – Displayed when a Data Logger configuration file is loaded and the loggers are created without error

File Not Opened: Cancelled by user - Occurs when the user closes the File dialog before opening a Data Logger configuration file

File Not Saved: Cancelled by user – Occurs when the user closes the File dialog before saving a Data Logger configuration file or a Data Logger file.

Saved file at: "File Path" - Displayed to let the user know where the Data Logger configuration file has been saved

Spread Connection Error from File - Occurs when unable to connect to the spread middleware using the Data Logger configuration file settings, most likely because spread is not running or the settings are incorrect

Unknown Host in File - Occurs when the specified host for the spread middleware is incorrect in the loaded Data Logger configuration file

Unmarshalling Error - Occurs when loading a configuration file that is invalid according to the Data Logger configuration schema

Required Java Libraries

Spread Toolkit: spread-3.17.0.jar

JWSDP-shared: activation.jar
commons-beanutils.jar
commons-collections.jar
commons-digester.jar
commons-logging.jar
jaas.jar
jax-qname.jar
jta-spec1_0_1.jar
mail.jar
namespace.jar
relaxngDatatype.jar
xsdlib.jar

JWSDP-JAXB: jaxb-api.jar
jaxb-impl.jar
jaxb-libs.jar
jaxb-xjc.jar

PTPlot: plot.jar
gui.jar

Appendix B: XML Schemas

- **Data Logger Configuration Schema**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="DataLoggerConfig"
    type="DataLoggerConfigDefinition"/>
    <xs:complexType name="DataLoggerConfigDefinition">
      <xs:sequence>
        <xs:element name="Host" type="xs:string"/>
        <xs:element name="User" type="xs:string"/>
        <xs:sequence>
          <xs:element name="Group" type="xs:string"
            minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:sequence>
          <xs:element name="Logger"
            type="SourceLoggerDefinition" minOccurs="0"
            maxOccurs="unbounded"/>
        </xs:sequence>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="SourceLoggerDefinition">
      <xs:sequence>
        <xs:element name="Target" type="xs:string"/>
        <xs:element name="File" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:schema>
```

- **Message Schema**

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Message">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="MessageEntry"
          type="MessageEntryType" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:complexType name="MessageEntryType">
    <xs:sequence>
      <xs:element name="key" type="xs:string"/>
      <xs:element name="type" type="xs:string"/>
      <xs:element name="value" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

- **NetPlot Configuration Schema**

```

<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified"
  attributeFormDefault="unqualified"
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="NetPlotConfig"
    type="NetPlotConfigDefinition"/>
    <xs:complexType name="NetPlotConfigDefinition">
      <xs:sequence>
        <xs:element name="Host" type="xs:string"
          default="agentsmith"/>
        <xs:element name="User" type="xs:string"
          default="NetPlot"/>
      <xs:sequence>
        <xs:element name="Group" type="xs:string"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:sequence>
        <xs:element name="PlotFrame"
          type="PlotFrameDefinition" minOccurs="0"
          maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PlotFrameDefinition">
      <xs:sequence>
        <xs:element name="Title" type="xs:string"/>
        <xs:element name="XLabel" type="xs:string"/>
        <xs:element name="YLabel" type="xs:string"/>
        <xs:element name="BufferSize" type="xs:int"/>
        <xs:element name="MarksStyle" type="xs:string"/>
        <xs:element name="Connected" type="xs:boolean"/>
        <xs:element name="Legend" type="xs:boolean"/>
        <xs:element name="AutoRange" type="xs:boolean"/>
        <xs:element name="Stems" type="xs:boolean"/>
        <xs:element name="Subject" type="xs:string"/>
        <xs:element name="XKey" type="xs:string"/>
      <xs:sequence>
        <xs:element name="YKey" type="xs:string"
          minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:element name="XLocation" type="xs:int"/>
      <xs:element name="YLocation" type="xs:int"/>
      <xs:element name="Height" type="xs:int"/>
      <xs:element name="Width" type="xs:int"/>
    </xs:sequence>
  </xs:complexType>
</xs:schema>

```

Appendix C: Sample XML Configuration Files

- **Data Logger Configuration**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<DataLoggerConfig>
  <Host>agentsmith</Host>
  <User>DataLogger</User>
  <Group>Orion1.F40</Group>
  <Group>Orion2.F42</Group>
  <Group>Orion2.F40</Group>
  <Group>Orion2.F45</Group>
  <Group>Orion2.F46</Group>
  <Group>Orion1.F42</Group>
  <Group>Orion1.F45</Group>
  <Group>Orion1.F46</Group>
  <Logger>
    <Target>Orion1.F40</Target>
    <File>logs/1F40.csv</File>
  </Logger>
  <Logger>
    <Target>Orion2.F42</Target>
    <File>/home/wpi/workspace/SpreadXML3/logs/2F42.csv</File>
  </Logger>
  <Logger>
    <Target>Orion2.F40</Target>
    <File>/home/wpi/workspace/SpreadXML3/logs/2F40.csv</File>
  </Logger>
  <Logger>
    <Target>Orion2.F45</Target>
    <File>/home/wpi/workspace/SpreadXML3/logs/2F45.csv</File>
  </Logger>
  <Logger>
    <Target>Orion2.F46</Target>
    <File>/home/wpi/workspace/SpreadXML3/logs/2F46.csv</File>
  </Logger>
  <Logger>
    <Target>Orion1.F42</Target>
    <File>/home/wpi/workspace/SpreadXML3/logs/1F42.csv</File>
  </Logger>
  <Logger>
    <Target>Orion1.F45</Target>
    <File>/home/wpi/workspace/SpreadXML3/logs/1F45.csv</File>
  </Logger>
  <Logger>
    <Target>Orion1.F46</Target>
    <File>/home/wpi/workspace/SpreadXML3/logs/1F46.csv</File>
  </Logger>
</DataLoggerConfig>
```

- **NetPlot Configuration**

```

<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<NetPlotConfig>
  <Host>agentsmith</Host>
  <User>NetPlot</User>
  <Group>CH1</Group>
  <Group>CH2</Group>
<PlotFrame>
  <Title>Trigs from File</Title>
  <XLabel>Time</XLabel>
  <YLabel />
  <BufferSize>100</BufferSize>
  <MarksStyle>various</MarksStyle>
  <Connected>true</Connected>
  <Legend>true</Legend>
  <AutoRange>true</AutoRange>
  <Stems>false</Stems>
  <Subject>CH1</Subject>
  <XKey>Time</XKey>
  <YKey>Sine</YKey>
  <YKey>Cosine</YKey>
  <YKey>Sine2</YKey>
  <YKey>Cosine2</YKey>
  <YKey>Sine3</YKey>
  <YKey>Cosine3</YKey>
  <XLocation>600</XLocation>
  <YLocation>0</YLocation>
  <Height>400</Height>
  <Width>600</Width>
</PlotFrame>
<PlotFrame>
  <Title>Noise from File</Title>
  <XLabel>Time</XLabel>
  <YLabel />
  <BufferSize>100</BufferSize>
  <MarksStyle>dots</MarksStyle>
  <Connected>false</Connected>
  <Legend>true</Legend>
  <AutoRange>true</AutoRange>
  <Stems>false</Stems>
  <Subject>CH2</Subject>
  <XKey>Time</XKey>
  <YKey>Noise</YKey>
  <YKey>IntNoise</YKey>
  <XLocation>0</XLocation>
  <YLocation>0</YLocation>
  <Height>400</Height>
  <Width>600</Width>
</PlotFrame>
</NetPlotConfig>

```

SOFTWARE DESIGN FOR THE FFTB PLOTTING UTILITY

Appendix D: FFTBPlot Software Design

Author: Victor Lu

Graphical User Interface (GUI)

The GUI design for the FFTBPlot is based on a hierarchy of chooser panels. A chooser panel is as shown in Figure 1:

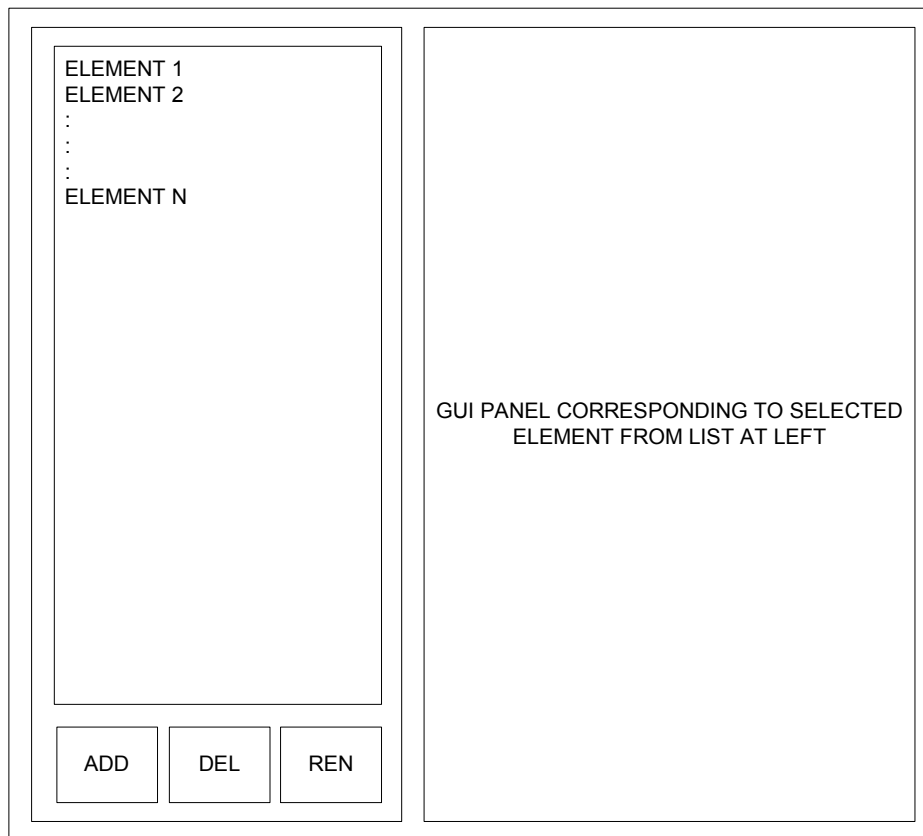


Figure 1: Chooser Frame

Each chooser panel allows the user to add, delete, or rename elements in the list portion through the ADD, DEL, and REN buttons. When the user selects an element in the list, its GUI panel that allows the user to customize its parameters appears in the panel on the right.

The class names for the distinct chooser panels are PlotWindowGUI (for selecting plot windows) and PlotChooserGUI (for selecting individual plots within each plot window). There is another class, the PlotGUI, which represents the GUI for configuring items within each plot.

Parallel to this are distinct chooser panels such as DataSourceGUI (for selecting data sources) and DataSeriesChooserGUI (for selecting individual data series within each data source). There is another class, the DataSeriesGUI, which represents the GUI for configuring items within each data series.

The hierarchy and data flow for these elements are shown in Figure 2:

SOFTWARE DESIGN FOR THE FFTB PLOTTING UTILITY

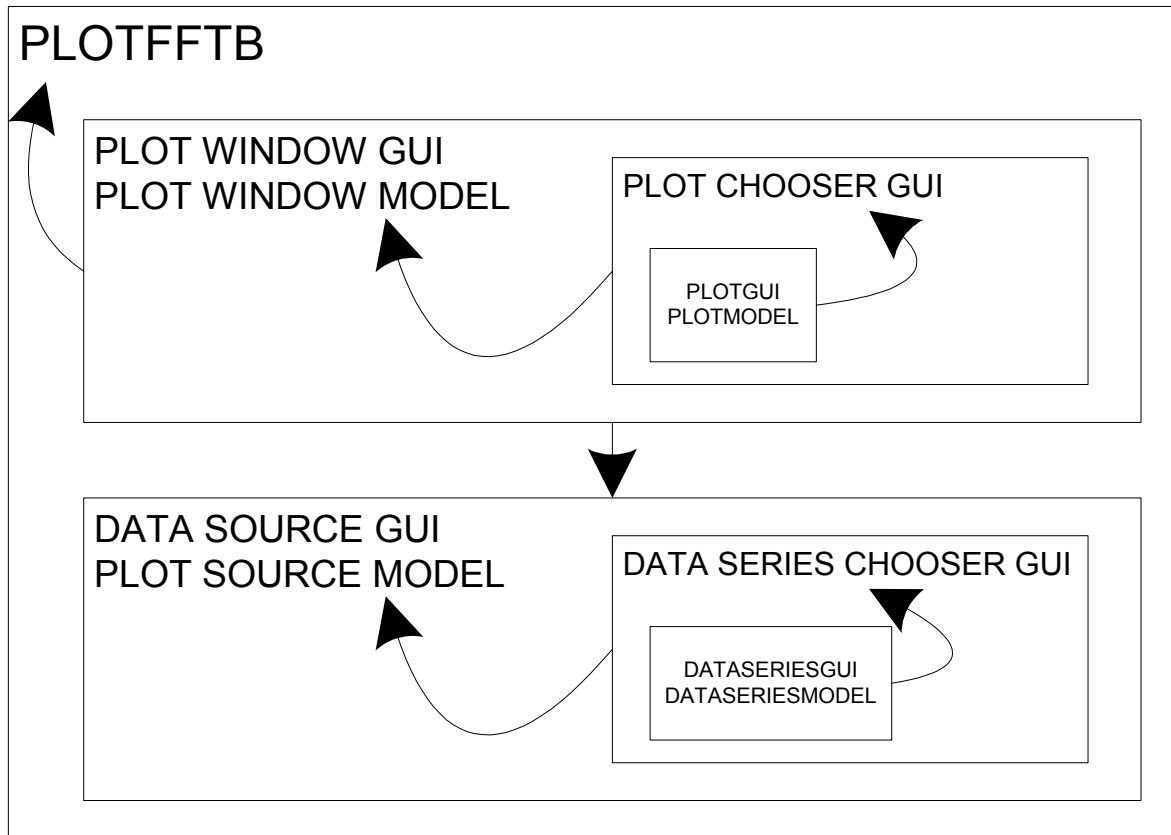


Figure 2: Data Flow and Hierarchy for Major Classes

As shown in Figure 2, the main class for the entire application is PlotFFTB. This houses all the elements in the application.

When the user starts the application, the PlotWindowGUI is brought up. By default, it is empty and no element is selected. Therefore, a default panel appears that instructs the user to select an element. After an element is added and/or selected, the PlotChooserGUI panel for that selected Plot Window appears in the panel on the right. Then, the same applies with the list of plots for each plot window. By default, a default panel appears, asking the user to select an element from the list of plots. After an element is selected, then the appropriate PlotGUI appears on the right hand panel.

In the PlotWindowGUI, there is a button for Configure Data Sources. When this button is pressed, the DataSourceGUI is brought up. Again, if there are no data sources, then a default panel appears, instructing the user to select an element from the list. And as with the plot choosers, the user selects an element in the list which brings up the appropriate DataSeriesChooserGUI. Then, the user selects an element in the Data Series list to bring up the appropriate DataSeriesGUI.

SOFTWARE DESIGN FOR THE FFTB PLOTTING UTILITY

Data Handling

Data in FFTBPlot is handled in much the same hierarchy as the GUI is.

The application gets data from two types of configuration files, both in XML format. In the code, these are referred to with a Model suffix (i.e. PlotWindowModel, DataSeriesModel).

Without the configuration files, all the lists are blank and the user needs to add elements.

The plotting XML file has the configuration parameters for the plot windows and the individual plots within each plot. Listing 1 shows a sample plotting XML file with the correct XML tag names.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<PlotRequest>
  <PlotWindow PlotWindowDataSource="GEONS SV FILE"
PlotWindowName="NewPlotWindow_1" PlotWindowXAxis="MJD Sec" PlotBuffer="1000"
PlotPointFrequency="10" PlotWindowIncludeLegend="true"
PlotWindowIncludeZero="false">
    <Plot PlotTitle="NewPlot_1">
      <PlotYAxes>
        <PlotYAxis>FDIR_1</PlotYAxis>
        <PlotYAxis>FDIR_4</PlotYAxis>
      </PlotYAxes>
      <PlotYAxisTitle>No Title Provided</PlotYAxisTitle>
    </Plot>
  </PlotWindow>
  <PlotWindow PlotWindowDataSource="GEONS SV FILE"
PlotWindowName="NewPlotWindow_2" PlotWindowXAxis="MJD Sec" PlotBuffer="1000"
PlotPointFrequency="10" PlotWindowIncludeLegend="true"
PlotWindowIncludeZero="false">
    <Plot PlotTitle="NewPlot_1">
      <PlotYAxes>
        <PlotYAxis>X Position (meters)</PlotYAxis>
        <PlotYAxis>Y Position (meters)</PlotYAxis>
        <PlotYAxis>Z Position (meters)</PlotYAxis>
      </PlotYAxes>
      <PlotYAxisTitle>No Title Provided</PlotYAxisTitle>
    </Plot>
  </PlotWindow>
</PlotRequest>
```

Listing 1. Plotting Configuration XML File

As Listing 1 shows, the highest level XML tag is PlotRequest. PlotRequest contains the lower-level tags PlotWindow. PlotWindow contains the lower level tags Plot. Plot contains the lowest level tags PlotYAxis.

The PlotWindow tag contains the most attributes such as PlotWindowName and PlotBuffer.

Plotting XML files can be found in the /sample directory after the user has installed the software.

Listing 2 shows a sample of the data source configuration XML file.

SOFTWARE DESIGN FOR THE FFTB PLOTTING UTILITY

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<DataSource DataSourceName="GEONS COV FILE">
  <DataSourceType>FILE</DataSourceType>
  <DataSourceFileName>/home/nfshome/live_data/out/Case-2/Case-
2_covariance.txt</DataSourceFileName>
  <DataSourceHeaderLines>4</DataSourceHeaderLines>
  <DataSourceRefreshRate>1</DataSourceRefreshRate>
  <DataSourceColumnDelimiter>SPACE</DataSourceColumnDelimiter>
  <DataSeries DataSeriesTitle="Time">
    <DataType>INTEGER</DataType>
    <DataSeriesHasSubColumns>>false</DataSeriesHasSubColumns>
    <DataSeriesNumberSubColumns>4</DataSeriesNumberSubColumns>
    <DataSeriesSubColumnDelimiter>SPACE</DataSeriesSubColumnDelimiter>
    <DataSeriesSubColumnType>BOOLEAN</DataSeriesSubColumnType>
  </DataSeries>
  <DataSeries DataSeriesTitle="MJD Sec">
    <DataType>FLOATING_POINT</DataType>
    <DataSeriesHasSubColumns>>false</DataSeriesHasSubColumns>
    <DataSeriesNumberSubColumns>4</DataSeriesNumberSubColumns>
    <DataSeriesSubColumnDelimiter>SPACE</DataSeriesSubColumnDelimiter>
    <DataSeriesSubColumnType>BOOLEAN</DataSeriesSubColumnType>
  </DataSeries>
</DataSource>
```

Listing 2. Data Source Configuration XLM File

The Data Source XML file contains the highest level tag DataSource, DataSource contains lower level tags such as DataSourceType, DataSourceHeaderLines, DataSourceRefreshRate, DataSourceColumnDelimiter, and DataSeries. DataSeries contains lower level tags such as DataType, DataSeriesHasSubColumns, DataSeriesNumberSubColumns, DataSeriesSubColumnDelimiter, and DataSeriesSubColumnType.

Data source configuration XML files can be found in the /cfg directory after the user installs the software.

Currently, the main class, PlotFFTB, houses all the implementation for the File menu to load and save data to XML configuration files.

GUI-Data Interaction

When the user loads or saves configuration data, the appropriate XML files are read or written to. Upon loading a configuration file, the GUI panels are updated appropriately. Upon saving a configuration file, the GUI elements are read from and the data are written to their corresponding tag locations in the XML files.

When the software executes (i.e. when the user hits a Plot! Or Apply button), the software will access what is in the GUI, NOT the configuration files, to determine how to plot the data.

There IS a difference between the PlotWindowGUI and the DataSourceGUI. In the PlotWindowGUI, the configuration for all the elements (i.e. the PlotWindows and their

SOFTWARE DESIGN FOR THE FFTB PLOTTING UTILITY

constituent Plots) is stored in ONE XML file under ONE PlotRequest tag. In the DataSourceGUI, the configuration for EACH data source is stored in a SEPARATE XML file.

Adding or removing parameters to the application will involve modifying the following:

1. The GUI component corresponding to the parameter must be either removed or implemented if new.
2. The Schema for the XML file must be changed appropriately. Schema can be found in the .xsd files. Once the schema is changed, then the Java Web Services Developer Pack Tool must be re-run to re-compile the code which accesses the data from the XML files.
3. The GUI component that uses the new parameter must be implemented in all the proper places to access the automatically generated functions from the Java Web Services Developer Pack Tool.