GFP0609

EMC² SYMAPI Phase II

A Major Qualifying Project Report:
Submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science

Sponsoring Agency: EMC²

Submitted by:

_____

Joshua Anderson

_____

Chris Cooper

Approved:

_____

Professor Gary F. Pollice, Major Advisor

Date: April 26, 2007

Abstract

This report details the EMC² SYMAPI project, focusing on Phase II, a Major Qualifying Project completed from B06-D07 by Joshua Anderson and Chris Cooper. The MQP is an extension of the EMC² On-the-Fly Configuration Java-based application developed by Andrew Ralich in B05. EMC Corporation's need for this software is a result of the growing costs of creating and maintaining on-site labs (Ralich 1). The program provides the company with a graphical drag-and-drop interface with which they can design and configure labs and simulate them via software without incurring heavy costs.

The primary goals of Phase II of the project involved expanding and enhancing the program's graphical user interface and increasing the program's functionality. Specifically, the main two objectives involved implementing an "auto layout" mechanism to clearly display labs while minimizing connection intersections and also enhancing the GUI to allow for more detailed connection placement based on specific ports of devices. Secondary project goals included other GUI developments, increasing extensibility by allowing hard-coded data to be specified via XML, code testing, and some minor bug fixes.

Acknowledgements

Special thanks to Peter Kushner, Andrew Ralich, Jon Krasner, and EMC^2 for allowing this project opportunity. Thanks to Professor Gary Pollice for advising this project and offering his help and expertise throughout the course of the project. Also thanks to Professor David Brown, Professor Stanley Selkow, and Professor Dan Dougherty for aiding in our research efforts.

Table of Contents

List of Figures

List of Tables

Introduction

Our project was sponsored by EMC Corporation, based in Hopkinton, MA. EMC² is one of the world's leading information technology companies, specializing in data storage and management solutions. A large portion of the products that the company sells are data storage devices (About EMC).

However, with constantly increasing hardware costs, a problem has arisen. In order for EMC² to effectively sell their products, they are required to loan out test products to potential clients. This allows the prospective clients to be able to test out the hardware and decide exactly what they need and how they want it configured. This is a costly process, and at some points, EMC² might have up to $15 million in equipment loaned out at one time, which is a huge insurance expense (Ralich 5).

In order to help to alleviate these costs, EMC² decided to begin the development of a software product that will enable their clients to virtually set up a laboratory environment with the products that they would like to purchase. This product will allow them to market their products without the incurred costs of loaning out actual products. However, developing a software program able to fully capture the capability of simulating a laboratory environment with actual EMC² products requires a lot of thought process and design (Ralich 5).

Last year, EMC² decided to enlist the services of WPI student Andrew Ralich to begin developing this application. Andrew created the EMC² On-the-Fly Configuration Java-based program to fulfill the company's goals. He provided them with a drag-and-drop style graphical user interface, developed with SWT, in which realistic EMC² lab setups could be visualized and configured (Ralich 8-19).

However, software projects are long, ongoing processes, and EMC² continued coming up with ideas on how to extend and enhance the software, making it more useable, detailed, and realistic. This year, they brought us in from WPI to handle these duties. The details of this portion of the project will be described throughout the remaining sections of this report.

The remaining sections of this document are the Background section, the Methodology section, the Analysis and Results section, and the Future Work and Conclusions section. In the Background section, the sponsor is described more thoroughly, and more information about the need for this software is discussed. The Background section also contains information about Computer Science concepts put to use throughout the project. In the Methodology section, we go into more detail describing how we used some of these concepts to solve specific problems, and we also discuss any other interesting details about the implementation. In the Analysis and Results section, we describe various ways in which we measured how well this project succeeded, as well as what we learned from the project as a whole. Finally, we conclude with the Future Work and Conclusions section, where we describe possible future endeavors related to this application.

Background

Sponsor Background

As previously mentioned, EMC Corporation sponsored our project, and their headquarters are located in Hopkinton, MA. EMC² is one of the leading companies in information technology and sells a wide variety of products, the bulk of which are related to data storage and management solutions (About EMC).

EMC² creates various hardware devices to handle data storage, and they typically sell these products all over the world to businesses in need of large storage arrays. The Storage Platform Enablers and Applications (SPEA) department at EMC² builds software applications used to manage their Symmetrix line of storage devices. Currently, over 100 programmers are employed in this department (Ralich 6).

The employees in SPEA also use other programs to help them develop the Symmetrix software. Frequently, the developers will need to have some sort of representation of a Symmetrix laboratory environment for use with these programs. Typically, this lab representation is obtained from data in a "debug log," which can be created by a host connected to the appropriate Symmetrix network (Ralich 6).

The generation of these debug logs require that the Symmetrix system is active, and EMC² has set up lab environments in various different configurations. However, SPEA believes that creating their own software applications in which virtually *any* setup can be represented is highly beneficial to the company. Having this type of program would enable SPEA developers to test lab environment configurations more completely and more cost-effectively, without requiring active devices (Ralich 6-7).

Application Background

Starting work on this project required us to gain familiarity with Andrew Ralich's existing code base. We quickly dove into the program and familiarized ourselves with the graphical user interface as well as the underlying code and the class structure. Andrew further aided us in acclimatizing to the new project by providing class diagrams and explaining some of the key features of the code. He also had us do a quick "getting to know the program" exercise and asked us to go through a list of small tasks, identifying which classes would have to be modified to accomplish each one.

The graphical user interface is developed using Java SWT graphics libraries and provides a user-friendly drag-and-drop environment to create EMC² lab setups. The window is essentially broken up into two major components: the palette and the draw area. The palette, on the left-hand side of the GUI, supplies the user with a collapsible menu of devices to select from, each with its own thumbnail to aid in visualization. Various types of Symmetrix and CLARiiON storage arrays, host machines, and connections are made available. The draw area comprises the majority of the window and provides a scrollable area in which the user can place and connect devices and visualize lab environments.

The basic code structure is comprised of separate packages and classes that separate and encapsulate code related to graphics and code related to data. The most vital classes are the ViewManager and MainControl classes. The ViewManager manages all the graphical aspects of the application, and the MainControl provides the ViewManager with a way to access the actual data stored by the program.

A large amount of information is contained in the various data storage classes. The EditPart class is an abstract class that represents a device in the virtual lab, and it is extended into subclasses with specific information for Symmetrix, CLARiiON, and Host machines. Each EditPart is also dependent on two other abstract classes. The ViewBase class stores information about the SWT images to use for each device, and the ModelBase class stores hard data pertaining to each device. Both of these classes are extended into subclasses for information specific to a certain type of device (Symmetrix, CLARiiON, Host). Connections between the devices are represented by a Connection class.

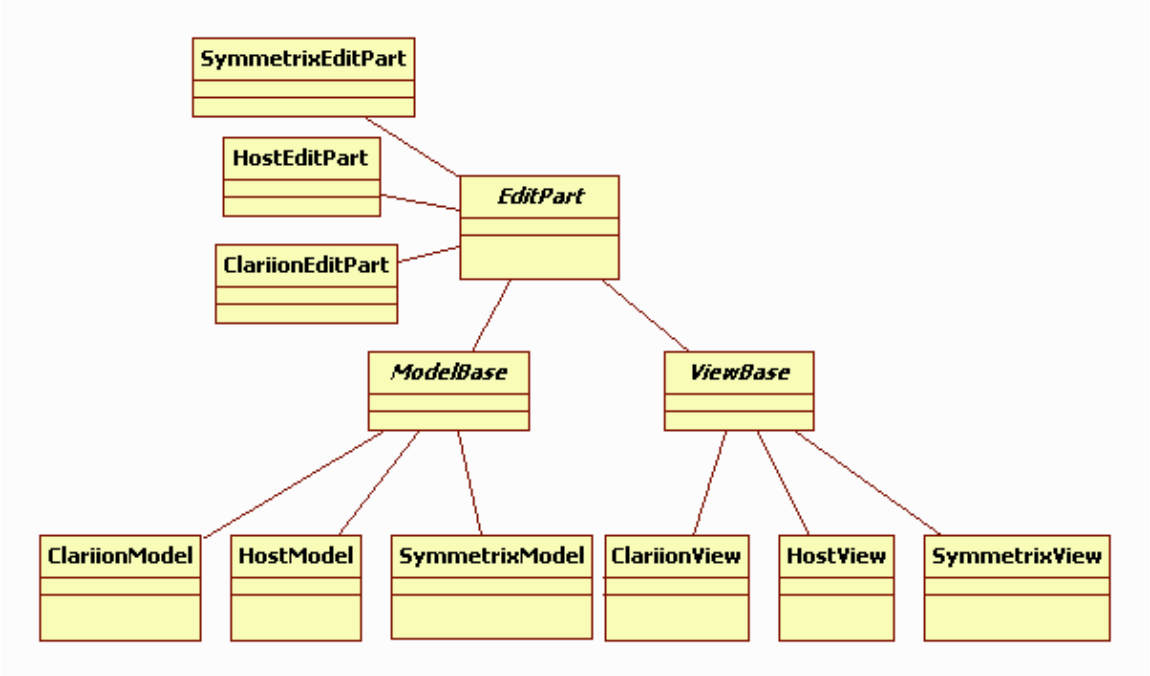

Figure 1: Part of Existing Class Structure (Ralich 14)

Methodology

Project Overview

This project was a two term project over terms B06 and C07, and the work was broken up into two portions, one for each term. The first portion of the project involved acquiring knowledge of the code base through fixing minor bugs and implementing new minor features. The most important and major project goals were fulfilled during the second part of the project. The tasks during this term were essentially split into two paths of work upon which EMC^2 had decided. One track was mainly algorithmic and required designing an "auto layout" feature to neatly arrange labs in the draw area of the GUI. The other track consisted of mostly graphical requirements and focused on providing the user with an interface to view the backs of devices and connect specific ports dynamically. A final important goal during the second part of the project was to implement the ability to save and load lab configurations via XML.

Part One

During the first portion of the project, a number of minor bugs were fixed, and several more minor features were implemented. On the first day, we started learning the SWT library, with which neither of us had much experience previously. We started work on the properties dialog windows that can be brought up to change some of the properties for each type of device. We fixed the layout of the buttons in the dialogs and also implemented a mechanism to keep the windows from popping up partially off the right

side of the screen. Later during the term, we also fixed a bug in which some properties in the properties windows were not being saved correctly.

The majority of the larger goals that we completed during this term were GUI enhancements, or HCI enhancements. When we began work on the project, the user essentially had no way of knowing if they were in "connection placement" mode after clicking on the first of the two devices to connect. To adhere more to user expectations, EMC^2 wanted us to implement "rubber banding" during connection creation. To do this, we altered the GUI code so that after clicking the first device to connect, a line of the correct color for that connection type extends from the center of the device, following the mouse cursor until either a second device is selected or the connection is cancelled. Connection cancellation was also a feature we added during this term, allowing the user to right click to stop making a connection after selecting one device.

During this term, we also implemented keyboard accelerators to make it easier for the user to perform the functions of the mouse simply with keyboard input. We edited palette and draw area code to allow the user to scroll through the palette selections with arrow keys and select a palette item with the enter key. We also wrote code to allow the user to shift focus between the palette and the draw area using tab and ctrl+tab and also to allow users to move devices in the draw area with arrow keys. To make the keyboard accelerators more user-friendly, we also implemented a way to select a device already placed on the draw area by drawing a black box around the "currently selected" device and then allowing the user to cycle through the devices using tab and shift+tab.

Also, when we began work on this project, the program was in a state such that the user needed to reselect the same item in the palette to place more than one in the draw

area. We modified the GUI code to allow for "multi-device drop," which lets the user simply continue clicking in the draw area to create more of the selected device without having to move over to the palette and select it again. We also made more minor GUI enhancements and bug fixes throughout the term.

Aside from the new features implemented during this portion of the project, EMC2 also wanted us to develop a test suite to ensure that all the code was doing what was intended. We created JUnit tests, using JUnit 3.8.1, to test all code not related to the SWT graphics. The testing is discussed later, in the Results section of the report.

Finally, we also performed some refactoring during this stage of the project. Some of Andrew's code was redundant in places, particularly in the device properties dialog windows, so we applied some object-oriented design principles to make the code better and more adaptable to change, not only for us, but for developers who work on this project in the future. Refactoring and object-oriented design concepts are mentioned again later in this section.

Part Two

During the second term, as previously mentioned, tasks were broken up into two main paths. The more graphical-oriented goals involved devising a way for users to visualize the back planes of devices and select specific ports to connect. The more algorithmic requirements were to implement an auto layout feature to automatically display a lab environment neatly on the screen with few connection intersections. Chris chose to work on the graphical track, while Josh worked on the algorithmic track. Towards the end of the term, one other important goal surfaced, and EMC^2 asked us to

finish the XML saving and loading of lab setups, which had been started by Andrew previously. The main goals of the entire project were fulfilled throughout this stage of the project.

The Graphical Track

The goals of the graphical track involved allowing for more advanced connections to be made by the user. A method needed to be implemented to allow the user to make more advanced connections. This eventually involved the creation of new dialog boxes, the gathering of new images of back planes of devices, and also the creation of some new classes. This provided a challenge, as the new classes created had to work well with the existing classes that had already been implemented for the rest of the system.

During this portion of the project, one of the most important goals was to allow for multiple connections between the same two devices. In the original implementation of the program, only the ability to create single connections between devices existed. However, in reality, the storage arrays can contain many different ports, and numerous connections can be made between different ports of the same devices. In order to meet this requirement, the entire connection creation process and the connection establishment classes were rewritten. All of the code involving connections was modified to use a new connection class, in which the "sides" of the connection were no longer entire devices, but specific ports of devices.

Originally, EMC^2 wanted to have a full animation involved in the new connection creating process. The animation they had in mind involved having the device spin around to show its back plane, where the connections are made. We were unsure that this

was the best idea for implementing this feature. The implementation seemed to be very expensive time-wise. The fact that the devices are quite small on the screen also seemed to pose a problem with being able to make connections effectively. Chris contacted Professor Brown, an HCI expert at WPI, for help with this issue and submitted a number of ideas to him to get his thoughts on the situation.

After consulting Professor Brown, the idea of creating a new dialog box to pop up during the connection making process seemed to be more intuitive. EMC^2 agreed to this new idea. We were eventually able to obtain actual back images of the storage array devices, with some difficulty, since only a few people have permission to take photos in the labs. Chris then created a graphical representation of a board and put all the images to use for each device's new connection placement dialog.

In this implementation, when a user clicks on a storage array device to make a connection, a window pops up, displaying the proper back plane. The user can then click on a port to select it, and the port highlights in the color of the connection type being created. Once the desired port is chosen, the user clicks a "Done" button to complete the process.

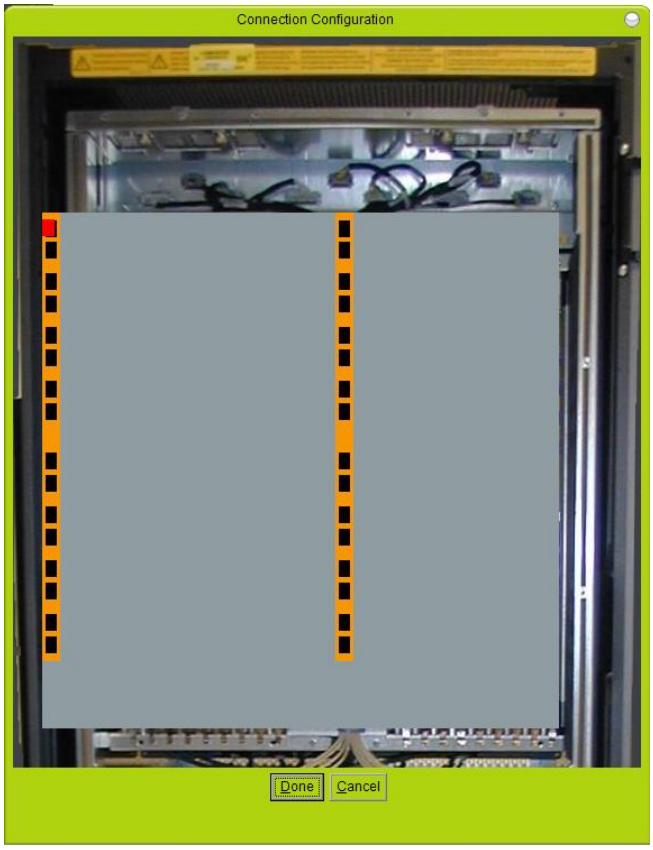

Figure 2: Connection Creation Dialog

Chris also created another dialog window as part of the graphical track. He expanded the connection properties window to actually display the back planes of both devices connected via the selected connection. The window shows connections between the two devices as lines between ports, and other external connections are represented by colored ports.

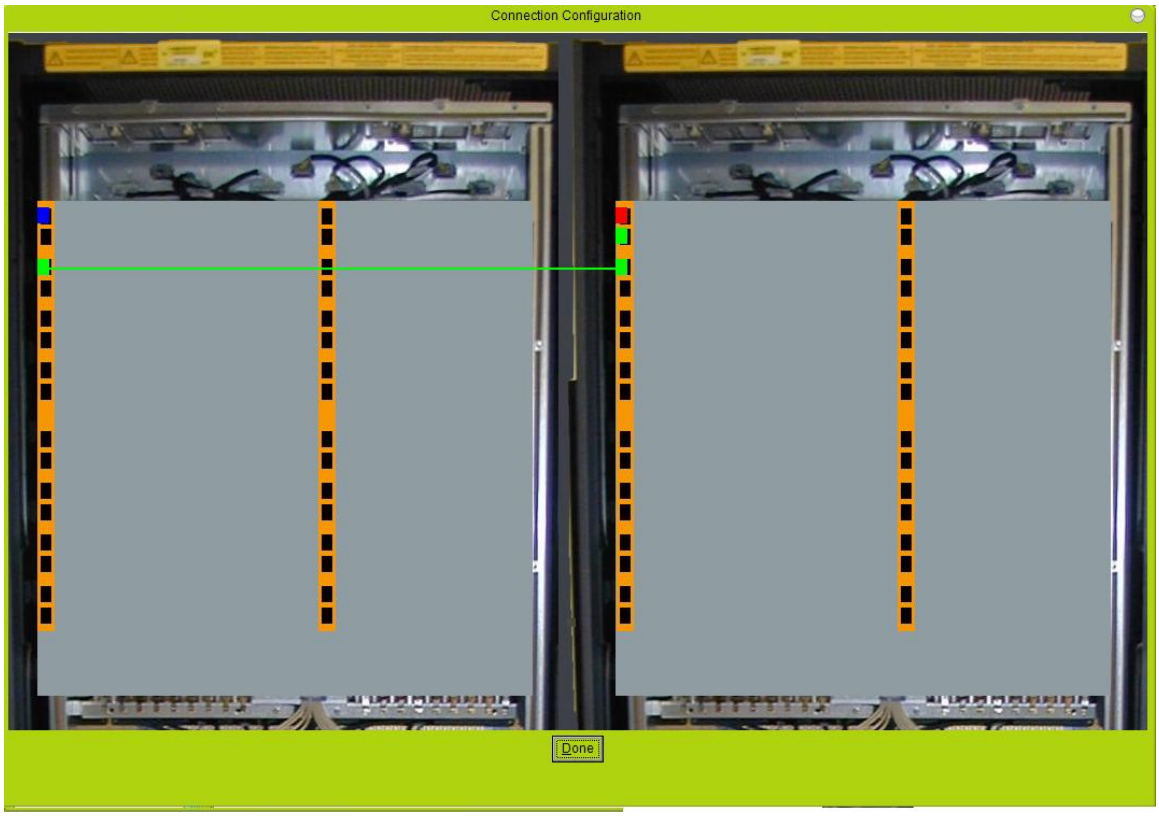

Figure 3: Connection Properties Dialog

The Algorithmic Track

As aforementioned, the main goal of the algorithmic track was to implement an auto layout mechanism to allow the user to easily arrange a lab setup neatly in the draw area with a minimum amount of connection intersections. The main use for this feature was to automatically layout a lab configuration loaded from an XML file.

Josh began work on this portion of the project by researching graph drawing algorithms. He talked to Professor Stanley Selkow, an algorithms expert at WPI, and obtained some literature on the subject from Professor Dan Dougherty. After studying

several graph drawing methods, Josh chose a form of radial graph drawing as the most suitable algorithm to use for the auto layout.

The radial graph drawing algorithm draws the nodes of a graph in concentric circles around a center node. After the center node is drawn, all nodes connected to it are drawn in a circle around it. Increasingly large circles of nodes are drawn outward, each containing nodes connected to nodes in the previous circle (Di Batista, Eades and Tamassia 52-55).

The adapted algorithm in the application takes advantage of Dijkstra's algorithm to place the devices in the appropriate circles. Dijkstra's algorithm is used for calculating shortest paths from one node of a weighted graph to all other nodes (Cormen, Leiserson and Rivest 595-601). In this case, all edge weights were the same.

The implementation of the auto layout starts by finding the object with the most connections and places that in the center of the viewing area. The algorithm then lays out the other devices it is connected to radially around it, with devices placed in the concentric circles based on the number of hops they are from the device in the center. This algorithm allows for the neatest possible layout of the objects by minimizing intersections as much as possible.

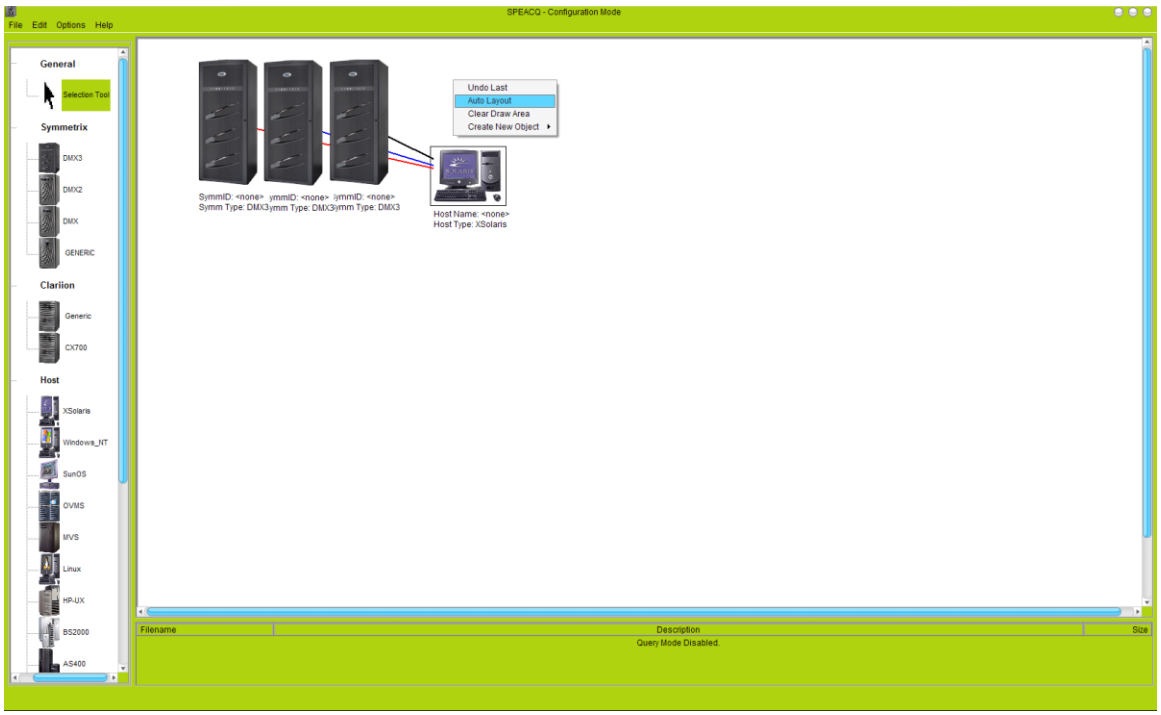

Figure 4: Pre-Auto Layout

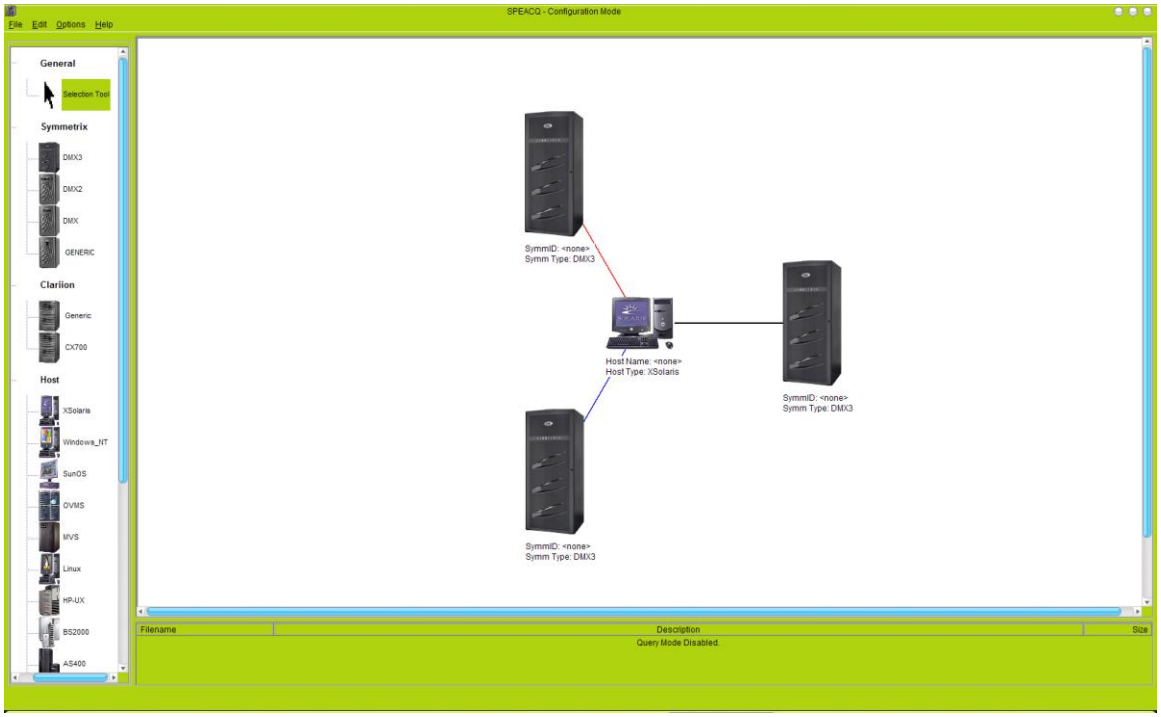

Figure 5: Post-Auto Layout

Saving and Loading Lab Configurations

The last goal of this portion of the project was to implement the saving and loading of laboratory setups from XML files. This provided a challenge, as we were required to use schemas that were existing formats used by other software that EMC^2 develops. The XML file that saved the configurations had to conform to this schema, and, upon loading, we had to figure out how to parse it properly using SAX and recreate the laboratory appropriately.

In order to represent connections between devices in the XML, we added new elements to the XML file upon saving to store board and port information as well as connection type data. Currently, the XML saving stores information about Symmetrix devices locally attached to Hosts.

```
  </device_set>
  <model>DMX3</model>
- <connection>
    <attachment>local</attachment>
    <board>0</board>
    <director>0</director>
    <port>0</port>
    <type>SCSI</type>
  </connection>
</Symmetrix>
```

Figure 6: XML Connection Data

eXtreme Programming Concepts

The eXtreme Programming approach to software engineering is an idea utilized by many software development companies around the world. Its focus is on a team

atmosphere, and it is rooted in a number of core practices based on this team environment and development in stages, or iterations (Jeffries). We used several of these practices throughout the course of our project.

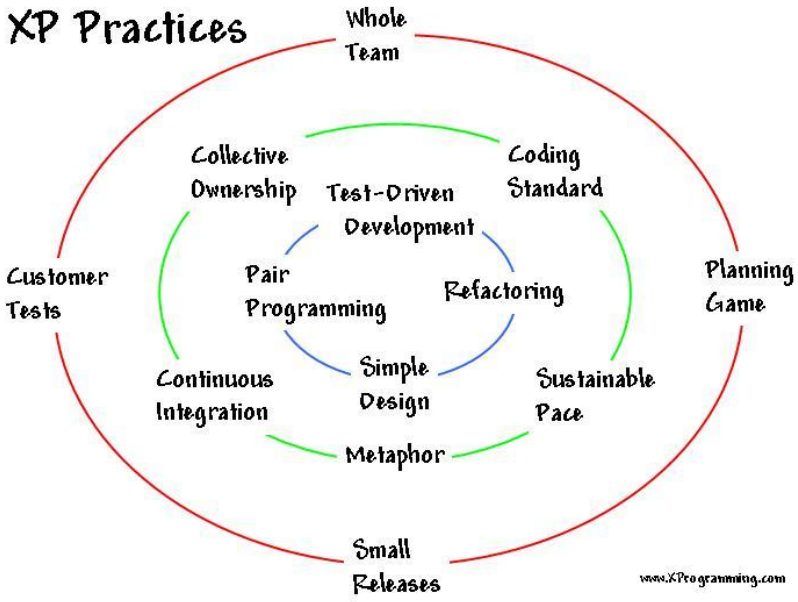

Figure 7: eXtreme Programming Core Practices (Jeffries)

We put the "Whole Team" eXtreme Programming concept to use during the entire development process. The essential element of this practice is the "customer," for whom the software is being developed. The customer basically drives the project by coming up with prioritized requirements for the developers to fulfill (Jeffries). In this case, quite apparently, our sponsors, EMC², filled the role of the customer. Each week, representatives of the company, namely Andrew Ralich and Peter Kushner, communicated with us and let us know which project goals took precedence over others at that time.

The "Pair Programming" practice of eXtreme Programming involves coding in pairs, with one developer at the keyboard coding and the other at their side monitoring their partner's work, catching errors and communicating ideas (Jeffries). We used this method of programming many times, particularly during the first term of the project, which was extremely beneficial in gaining familiarity with the code and implementing minor requirements. We were able to effectively combine our efforts and focus on the same tasks. While pair programming, we also made use of the eXtreme Programming "spike" to quickly research Java libraries we weren't familiar with.

The "Refactoring" eXtreme Programming concept refers to cleaning up code and making it easier to work with without changing functionality (Jeffries). During the first portion of the project, as mentioned above, we refactored some of Andrew's code to make it easier to adapt to changes made down the line. The device properties dialog classes contained redundant code, so we condensed all the information into one class. Andrew's type classes, which contained information about device types, also contained a lot of superfluous code. We created an abstract class to store all the generalized type data in a single class and then subclassed for type-specific data.

Object-Oriented Design Concepts

Object-Oriented Design is a software development practice based on code modularization, abstraction, and delegation of responsibilities. Design patterns are a major component of object-oriented design, and provide problem-solving methods that can be adapted to solve common problems in many different design situations. Examples

of design patterns used in this project are the Observer, Model-View-Controller, and Factory (Shalloway and Trott).

The Observer pattern is used when one class needs to update based on a change in the status of another class. One class is "observed" by another class, and the Observer is notified of the Observable class' changes and updates accordingly (Shalloway and Trott 315-327). We used this design pattern in order to update and reform connections after a moving a device in the draw area. We made MainControl an Observer of ViewManager to accomplish this task.

Model-View-Controller is a pattern usually applied to GUI development. The main idea is that responsibilities are allocated to specific classes, with the "View" containing graphical elements, the "Model" containing data, and the "Controller" serving as an access point to the data (Ralich 12-13). This pattern was already put into use by Andrew, with the ui package managing the GUI components, the model package containing the data, and MainControl acting as the Controller.

The Factory pattern is used to delegate the responsibility of creating instances of a certain kind of object to a specific class (Shalloway and Trott 385-390). Chris employed this pattern to control the creation of boards, directors, and ports associated with each device. Each device contained its own board Factory.

Results and Analysis

The best way to assess the success of this project was to listen to feedback from the sponsors. We periodically gave project demonstrations on-site at EMC^2, and this gave us a good indication of whether or not the project satisfied their requirements and meeting their goals. Each meeting with them gave us a better understanding of what we needed to focus on and helped us prioritize requirements to some degree.

We also developed a JUnit test suite for the non-graphical elements of the code base, which was something that was not previously done. We wrote test cases for individual classes and methods to assure that our code and the code written previously by Andrew did what was intended and was free of errors. We installed the Eclipse Coverlipse plug-in to give us some values with which we determined how thorough our testing was. After we made a lot of implementation changes, particularly with how connections were implemented, and after we developed new tests, the Coverlipse plug-in listed the following code coverage values for the non-graphical packages:

Package	**Coverage %**
logging	93%
model	98%
property	100%
type.types	94%

type.types.special	100%
type.typesets	98%

Table 1: Coverlipse Code Coverage for Non-Graphical Packages

We obtained another measure of how good the program was by using Visual Paradigm and interfacing it with Eclipse. This allowed us to view the class and package structures and was helpful during the refactoring stages of the development. From Visual Paradigm, we were able to visualize cohesion and coupling, and this aided us in trying to achieve a balance between the two. Cohesion refers to how well elements of a single class function together to achieve a desired purpose. Coupling refers to how dependent a class is on other classes. Typically, high cohesion and low coupling is desired; however, high cohesion usually increases coupling, so finding a balance is necessary. During the refactoring of the code, we were able to work towards this result by eliminating redundant classes and doing some delegation of responsibilities.

Overall, the project accomplished many different goals. In B-Term, we jumped in quickly and established a familiarity with a fairly sizeable code base, thoroughly tested the code, applied object-oriented design concepts to refactor code, and added functionality desired by the sponsors to the application. We were able to provide a way for a user to visualize when they are in "connection placement" mode by implementing rubber banding, and we also devised a way to provide users with the option of using keyboards to manipulate many aspects of the GUI. We also satisfied many other minor requirements during this term.

In C-Term, we fulfilled the main goals of the project set by the sponsors. We developed an auto layout mechanism to neatly display labs, a graphical means of connecting specific ports of devices, and added the ability to save and reload lab configurations using XML.

After several project demonstrations for EMC^2, we believe that most of their initial goals set at the beginning of the year were met. We achieved all of the most important goals, and many minor requirements as well, and the sponsors are satisfied with the project.

Future Work and Conclusions

Future Work

In the future, this software will become more detailed, and a large number of rules not currently implemented will be specified via XML, providing for more realistic lab setups and simulations. Most of these rules have to do with limiting port-specific connections in various ways and more specific device properties. There will also be some rules added to make boards more device-specific.

The GUI will also be further enhanced to allow users to dynamically add and remove boards from devices. CLARiiON back images will also be added, with horizontally placed boards, and their own device-specific boards and ports.

The XML saving and loading of lab configurations could also be extended to include Symmetrix devices remotely connected to hosts, as well as CLARiiONs. All of these ideas have been mentioned by EMC^2 as future goals for the project.

Conclusions

As a result of doing this project, we gained knowledge of Java libraries that we were not previously familiar with, specifically SWT and SAX, and we also acquired experience in a real life software development environment. We learned the importance of time management, object-oriented design principles, and maintaining good communication with the customers.

If we were able to do this project again, we would do several things differently to make it go more smoothly. We would spend less time on minor functionality and part one of the project. We would also communicate better with the customer regarding time constraints and estimations of what work can be accomplished within those constraints. Finally, we would also do the project as a one-term project to eliminate scheduling issues and the need to balance course work, project work, and extracurricular activities. This was probably the biggest issue with regard to how well the project ended up going.

Works Cited

About EMC. 2007. 25 April 2007 <http://www.emc.com/about/>.

Cormen, Thomas H., et al. Introduction to Algorithms. MIT Press and McGraw Hill, 2001.

Di Batista, Giuseppe, et al. Graph Drawing: Algorithms For the Visualization of Graphs. Upper Saddle River, New Jersey: Simon and Schuster, 1999.

Jeffries, Ronald E. What is Extreme Programming? 1999-2007. 25 April 2007 <http://www.xprogramming.com/xpmag/whatisxp.htm>.

Ralich, Andrew. "EMC² On-the-Fly Configurations." MQP Report. 2006.

Shalloway, Alan and Trott, James R. Design Patterns Explained: A New Perspective on Object-Oriented Design. Boston, Massachusetts: Pearson Education, 2005.