Project Number: GFP0802

Wonderland Task Viewer

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

by:

Matthew Duval

James Johnson

Ryan Moak

Date: March 17, 2009

Approved:

Professor Gary F. Pollice, Major Advisor

Abstract

Our objective for this project was to implement a productivity tool for a virtual team project room using the Sun Microsystems Project Wonderland software. More specifically, we created an application that would allow for teams using the Project Wonderland software to discuss, manage and view task data from within Project Wonderland itself. In order to do this we had to create a Graphical User Interface (GUI) within Project Wonderland that a user could interact with, as well as implement a system that could communicate with an external task management application such as SourceForge®.

Acknowledgements

First, we would like to thank Sun Microsystems for developing the Project Wonderland Virtual Environment because without it we would have been unable to undertake this project. In addition, we'd also like to acknowledge the work of the developers of Pivotal Tracker who have created a free, feature-full online task management application which was essential in implementing our application's repository backend. Finally, we would like to especially thank Professor Gary Pollice for taking the time out of his schedule to advise our team and for consistently providing relevant guidance and feedback throughout the life of our project.

Table of Contents

List of Illustrations

1. Introduction

Productivity is a topic widely discussed in software development and in recent years, productivity tools have grown tremendously in usefulness and popularity. One avenue being explored is the idea of virtual environments as a tool for ease of communication as well as a means of increasing productivity between team members located in different regions around the world. As a result, many virtual environments have been created in recent years such as the popular Second Life® as well as the relatively unknown Project Wonderland. Although, quite different in terms of their intended uses they do share much in common as Virtual Worlds. This project will deal specifically with creating a productivity tool to manage tasks for a software engineering team within Project Wonderland.

The goal of this project was to create a productivity tool in the Project Wonderland virtual world which would allow a user or users to maintain a list of projects and tasks. Project Wonderland is an open source virtual environment developed by Sun Microsystems that aims to provide an experience that is secure, flexible, and stable enough to be used for real business[12]. Organizations would be able to design their own worlds in Project Wonderland to suit their groups' needs and would be able to import tools easily from the Project Wonderland developers' network.

The project we worked on was interesting for a couple of reasons. First, Virtual Worlds have recently come under increased focus for their usefulness as virtual meeting rooms. They enable people who are working on projects in different areas of the world to take part in traditional conference room meetings. Although conference calling and webcams do allow for interaction between remote parties these solutions require additional technology to be purchased and are often unable to be customized to fit a specific company's needs. Virtual Environments on the other hand can allow participants to simulate a real conference room which could potentially foster a greater feeling of connectivity between users. Although this idea is unproven, it is one of the goals of Project Wonderland to explore this possibility. Another benefit of Virtual Environments is that the only technology required to attend a meeting is a computer and an Internet connection which is often readily available in the modern workplace.

Second, the program we designed is meant to be extensible which is very useful in the modern software environment. Many applications exist to manage tasks, and as a result we did not want our application to limit our users by forcing them to use a specific repository or task management backend. This would allow people to use our program without forcing them to recreate all the Projects and Tasks that they already have stored in another repository. We designed an abstract backend to support this increased flexibility.

Creating a productivity tool for a virtual team project room using the Project Wonderland software was the project's initial vision, but there are several parts to what was intended in that vision. The concept of the application depended heavily upon an abstracted backend for accessing an external task management system. This backend would supply all functionality for controlling that system to the software, which would allow for an easy transition to a different external task management system, if such a transition were desired. The backend would also provide a generalized definition and organization of a task, which would provide a concrete basis for how the UI would work and flow.

Design for the user interface focused on three primary tasks: logging in and out of the task management system, exploring a set of projects and the tasks within, and editing and updating a task. These tasks would be accomplished by a series of panels which would be controlled by a common cell application object. The cell application object is a part of the Project Wonderland Application Programming Interface (API) and will be discussed in greater detail later on in this paper. Java's Swing and Abstract Windowing Tool (AWT) libraries were used in the plans for how the UI would be accomplished.

The next sections of the paper go into more detail about the project. The background section delves into the current state of the art in the areas of virtual worlds, task management, and Project Wonderland. Methodology focuses on the design that we chose to implement for our Task Manager. More specifically it discusses the package structure of Project Wonderland, the design of the graphics and controller packages for the project, and finally the Extensible Markup Language (XML) parsing software that was used. The results and conclusion section discusses how the project turned out in comparison to our original design, and also outlines future work that could be undertaken to improve our application.

2. Background

Virtual Worlds

While Project Wonderland may be fairly new, the concept of virtual worlds has been around for quite some time. In 1997, Active Worlds, Inc. released Active Worlds©, its 3d virtual reality platform that allows its users to explore and modify 3d virtual environments. In 2003, Linden Research, Inc. released Second Life, a virtual world that users interacted with through the Second Life Viewer, a free client program. All three of these virtual worlds are very different, but they all have certain commonalities.

The virtual worlds that these programs reside in are very different from each other. Active Worlds is designed for their users to pick from a huge list of worlds to become a part of and modify. All of these worlds are different. Within these worlds, players can claim a piece of land as their own. Second Life, on the other hand, has one massive world in which everyone resides. Land must be "rented" in Second Life in order to construct a building. Users of Project Wonderland do not own any of the world in which they reside, and cannot modify it from within.

All three virtual worlds have some mechanism to allow users to create extensions for the world. Active Worlds and Second life both do this with a tool inside the world, as well as externally through a Software Development Kit (SDK). Project Wonderland, on the other hand, can only be extended through a SDK. The extension tools for Active Worlds and Second Life are similar, in that their in-world tool allows for creation of simple 3D objects with simple scripts, and the SDK allows for more complicated scripting[7]. The SDK for Project Wonderland, on the other hand, focuses more on creating unique cell types, so that the user can add new functionality to their world.

In order for users in a virtual world to interact, they must have a way to communicate. All three virtual worlds allow for communication between users, through text and Voice over Internet Protocol (VoIP). In Project Wonderland, private conversations can be initiated between two people, or conversation can be open. VoIP volume is determined by distance from a person's avatar, so when having a conversation in the virtual world, two avatars must be fairly close. This is similar to Active Worlds, in which you can only see the chat of people within 200 meters of your avatar. Second Life does not have this problem, instituting a global chat.

Project Wonderland differs from the other virtual worlds in its target audience. Both Second Life and Active Worlds advertise their product to both the entertainment and to the business sector. For example, a study was done to determine the feasibility of Active Worlds for distance learning[3]. By teaching a 3D modeling course in Active Worlds, they showed that it worked well as a teaching tool because the teacher is able to modify the environment of their classroom to tailor it to their needs. The teacher was also able to show the students the actual models from different angles.

Project Wonderland's target audience is specifically the professional one. Their goal is to create an environment stable and robust enough to conduct real business in it[12]. Through use of collaboration tools through Project Wonderland, a team should be able to work together on a project in the virtual world without need to actually meet.

Task Management

Today software is ubiquitous. Cellular phones, automobiles, and even some everyday household appliances often contain a computer chip and millions of lines of code. As a result of this, software and Information Technology (IT) projects have become one of the most heavily budgeted areas in many businesses. Unfortunately what should be a profitable business can also have disastrous financial repercussions if not planned accordingly. The Institute of Electrical and Electronic Engineers (IEEE) reports that 5 – 15% of software related projects will be abandoned before or shortly after delivery[2]. In addition, most projects that do end up being completed often come in way over schedule and end up costing the sponsoring company millions as a result. In response to the rising costs and repercussions of failed software projects, many new development styles have been created in order to attempt to ensure project success. One of the most popular today is the Agile approach. Agile is a very broad term that focuses on the customer’s ability to direct the project and for the team to respond rapidly to change while delivering value to the customer.

Extreme Programming (XP) is an example of the Agile approach. By dividing the customers’ expectations into a series of software user stories, it becomes possible to then create various software implementation tasks based on exactly what the customer is asking for. The idea behind extreme programming is to tackle the most important tasks in the eyes of the customer in order to create periodic functional releases that can then be reviewed to make sure they meet the requirements of the project. If they do then you continue, if something doesn't work the way the customer wants then revisions are made until it does. By keeping the customer involved in the project, the most important features should be able to be delivered on time whilst the less important ones will be left off until the end or even not included.

The popularity of XP has led to the rise of a number of project management software applications that allow a project leader to store functional requirements, assign them to members of their team, and monitor the progress of the project as a whole. It was these types of applications such as SourceForge, Jira, Bugzilla, and so on that provided the basis for our project.

Task management is a crucial aspect of project planning and lies at the heart of all project management styles. Often the discussion of tasks and functional requirements is a central point to focus on during software development meetings in XP. In these meetings it is important to be able to access the tasks of a given project for a number of reasons. Viewing tasks allows you to see what work is left on a project, seeing how often tasks are completed is useful in gauging the progress of a team, and being able to modify the data stored in those tasks allows for the flexibility required by customers of XP. Although many software solutions exist that allow a group to manage a repository of tasks for a project there are a couple of specific concerns that arise.

First, many software projects these days are grandiose in scope and are often worked on simultaneously by different people, potentially in different areas of the world. In a global development environment such as this, a white-board or locally installed application to keep track of tasks will not suffice the majority of the time. Thankfully, there are Web applications that provide the same functionality and allow remote users to all have access to the same wealth of data. This allows everyone to access the same data but since every web application is not synchronized, people could potentially not all be viewing the same data. This can cause much confusion in software meetings and leads to the second issue.

The second problem with current task management practices that can be remedied by using a virtual world is the issue of concurrency. In these modern remote meetings often conference calling software, webcams, and chat rooms are utilized to discuss the current projects iteration. This requires interfacing with many different technologies at once in order to just attend the meeting which can be cumbersome. This difficulty is compounded once the meeting actually begins, since everyone is split up. In addition, project members who are attending the meeting via these remote methods could potentially miss important information and therefore might not be on the same page as other members. As a result, the important part of the meeting, i.e. the discussion of the project and current tasks, could potentially fall to the wayside.

Our goal in creating a virtual meeting environment with an embedded task manager is to allow everyone to view the same data concurrently and also create the illusion that they are all meeting in the same room. By creating a synchronous task manager that could interface with Sun's Project Wonderland Virtual Meeting Room it would be possible to discuss current functional requirements with all the members of your team regardless of location, whilst also ensuring that everyone is viewing the same information at the same time. In addition, the goal of our project was to have our task manager application be extensible enough that any project management software with an API could be used as the backend of our program. That way regardless of a project's current choice for task management, it would still be feasible to integrate it into our application and does not limit our software's users to a single flavor of project management software.

Project Wonderland: How It Works

Project Wonderland is a server-client program for displaying 3-Dimensional “volumes”, or cells. The project is broken down into a server package, a client package, and a common package. Servers are created with the server application, and client applications can access it via networking. The common package contains the functionality for communication between client and server, which works through a messaging system. An instance of the Project Wonderland application is comprised of many cells working together.

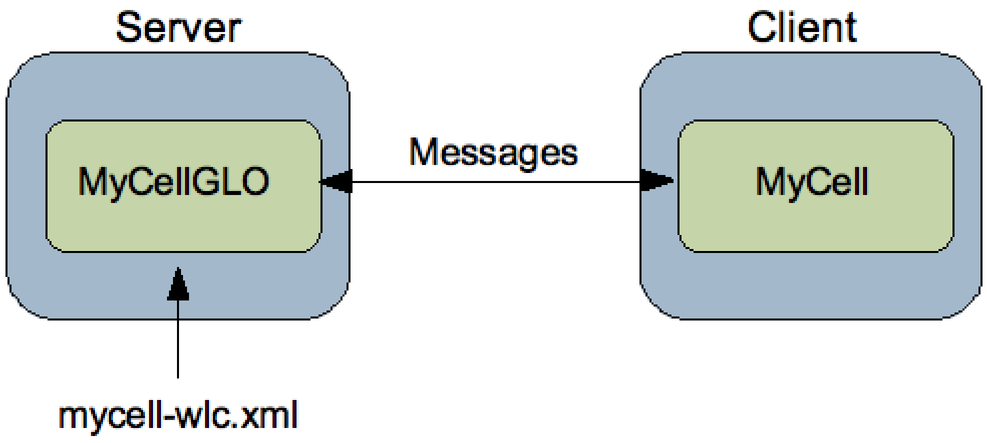


Illustration 1: Project Wonderland Client-Server Cell Design

A cell in Project Wonderland, defined as a “3D volume,” is actually composed of two cells. The server cell contains the data and data-accessing controls, which communicates with the client cell, which accesses and uses the data. The server cell resides in the server package and the client cell resides in the client package, the two becoming a part of the server and client applications respectively.

Cell messages are created for communication between specific cells, and those are kept in the common package, as both client and server cells need to create and use them. The messaging system for each cell follows a design similar to the proxy pattern, which involves an object sending messages through a medium to a proxy, which tells another object to act on the first. The first object in this case being the client cell, which utilizes the common package to send messages to the server package, which holds the proxy that reads the messages and tells the server what to do. The server in turn sends messages to the client.

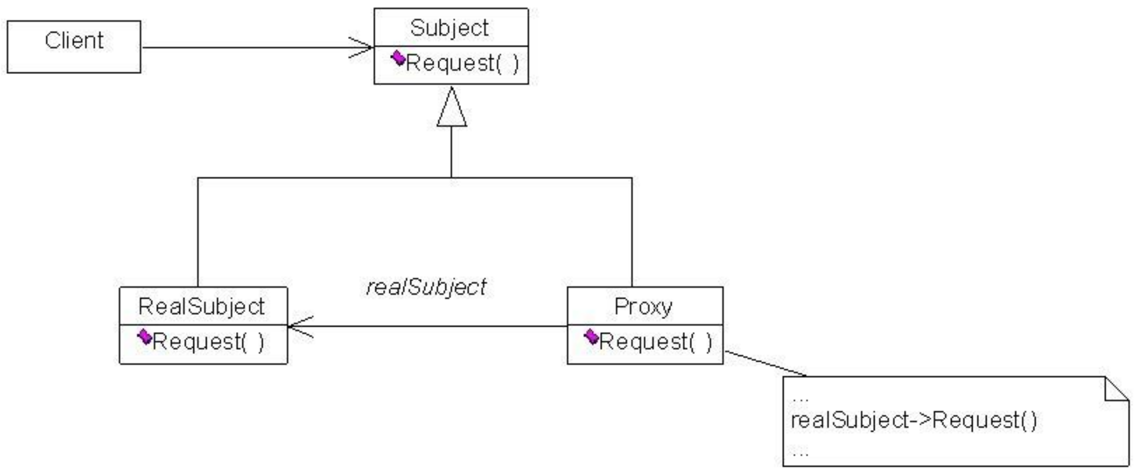


Illustration 2: The Proxy Design Pattern

When the Project Wonderland application is launched, the base cell of a server is accessed when a user logs in. The contents of that base cell are composed of cells from the Project Wonderland Modules Project, which is an Ant project composed of many various cell projects. Ant is a build tool for java-based applications[1]. It's very similar to Makefiles for creating a C or C++ application, except that it allows for more of a freeform project because of Java's portability. Each module cell project is integrated into the Wonderland Project when it is compiled, and launched at runtime when the base cell is being accessed and viewed.

Extending Project Wonderland, which is one of the main goals of this MQP, is done by creating a new extended cell project in the Wonderland Modules Project. An extended cell project utilizes the same structure as Project Wonderland, server/common/client, as the cell project will be integrated into the whole project when compiled. To create a 3D application, which runs within Wonderland, a server side cell i.e. a `StationaryCellGLO` or `StationaryCellAppGLO` is required. Corresponding to the server side cell, a client side cell needs to be written, which is composed of a `StationaryCellApp` and a `SharedApp2DImageCell`. Each user who logs into Project Wonderland will have a client cell generated for each server cell contained in the base server cell. Code is written for a user interface within the client cell, and data control within the server cell. Messages are created within the common package that both cells use. In addition, a derived instance of `CellSetup` is created which the cell framework is designed to utilize during the creation of the client and server cells.

Adding a cell to the base cell in a server is done by adding it to a specific file system of XML files that get accessed when building the cell. These XML files will tell the base cell what cells from the Wonderland Modules Project to create within itself. After adding the extended cell to the XML and putting it in the appropriate place, you should be able to run a server application with that cell running and access it from a client application.

Building the server and client applications is done through running Ant build files. The Wonderland Project and Wonderland Modules Project are both Java projects that use Ant build files, which allows for Project Wonderland's cell-based architecture to build the Wonderland Project with specific cell projects from the Wonderland Modules Project integrated into it. This is important to testing for this project because we can test to see if an error is generated from within our code or Project Wonderland by selecting a build path that does not include our code and compare it to one that does.

3. Methodology

As it was created by Sun Microsystems, Project Wonderland is designed to be developed in the NetBeans[8] Integrated Development Environment (IDE). The Project Wonderland file system is a NetBeans free-form Ant project that is composed of even more free-form ant projects. When we began our project, however, we were unable to get Project Wonderland to run using the Netbeans IDE. We were unable to pinpoint exactly why Netbeans was failing but after many hours of work we still had not succeeded in launching Project Wonderland without errors. In order to overcome this roadblock we chose to transition our work to a more familiar IDE, Eclipse. Although Project Wonderland was not designed specifically for use within Eclipse, we were still able to get it up and running relatively hassle free which was a vast improvement from the results we achieved through the use of Netbeans.

The Eclipse IDE is not designed to take in the Netbeans free-form Ant project. Because of this, the entire Project Wonderland file structure had to be loaded as the project. Ant still worked the same way, but the Java compiler didn't work in every file, so the majority of our debugging had to be done by compiling the code with Ant and checking the output for syntax errors. Ant worked perfectly in Eclipse after the build path was configured, so that was the IDE of choice for the rest of the project.

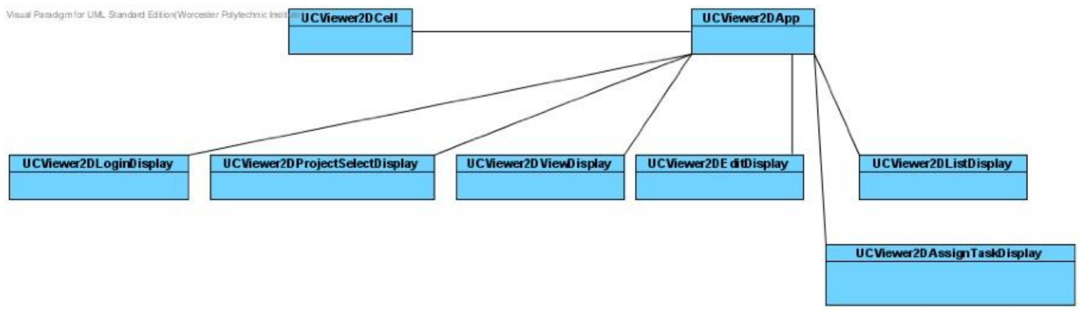
The Web-based administration features of Project Wonderland allow for easy dissemination of the Project Wonderland client to users[11]. With a specific Ant target, the user can build a WAR file from their Project Wonderland source directory (or the default can be downloaded online). The WAR file contains a Java Web start client which will launch Project Wonderland, a Wonderland World Builder which is a simple interface to let users quickly build worlds for use in Project Wonderland, and a Wonderland Art Upload that allows the user to upload art to the Project Wonderland server to add new content.

In order to use this WAR file, it has to be stored in a Web container that supports Java Servlets. The Project Wonderland website suggested the use of Jetty, and it worked out fairly well for us. Jetty was very easy to set up and configure. One configuration file had to be modified, the WAR file had to be copied into a folder, and then a JAR file was run. Jetty did have one problem, though. On occasion when trying to download the Web start client, the Jetty server would give a security exception and crash. The error seemed to happen at random and only on one computer, so the problem was never solved.

Besides the one error with Jetty, it turned out to be a very effective way to demo the project. After building the WAR file and starting the Jetty server, anyone with a Web browser and Java 5 or 6 can go into your version of Project Wonderland and see what you have done. This was a perfect way to demonstrate our tool as we added more and more functionality.

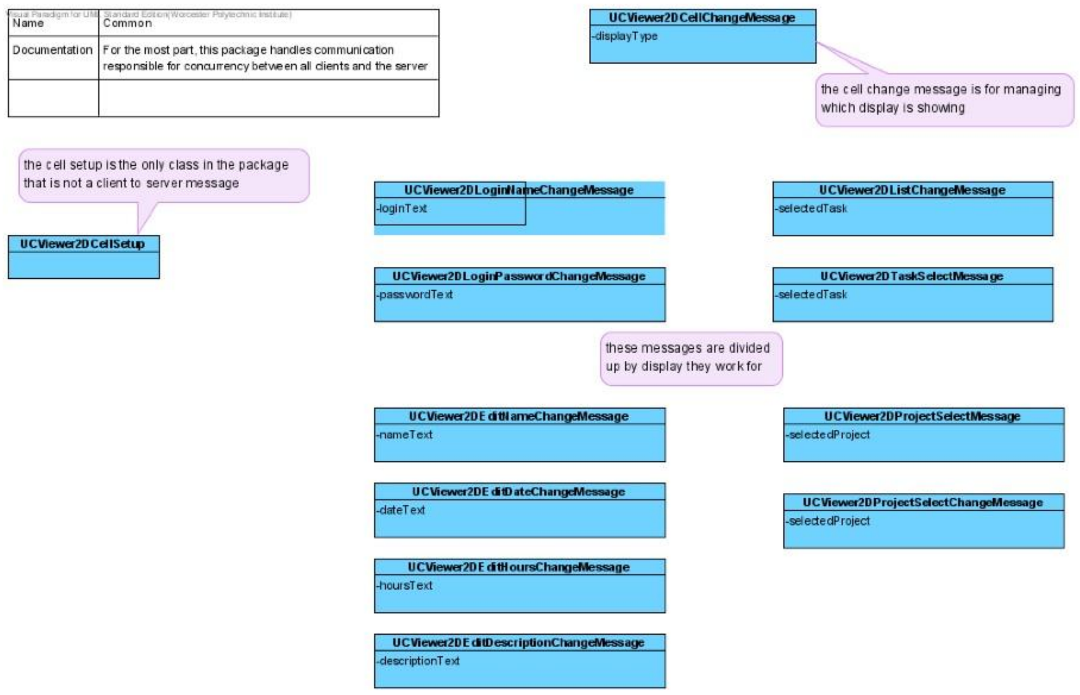
Class Diagram

Client


Visual Paradigm for UML Standard Edition(Worcester Polytechnic Institute)
UCViewer2DCell
UCViewer2DApp
UCViewer2DLoginDisplay
UCViewer2DProjectSelectDisplay
UCViewer2DViewDisplay
UCViewer2DEditDisplay
UCViewer2DListDisplay
UCViewer2DAssignTaskDisplay

Class Diagram

Common



Name	Common
Documentation	For the most part, this package handles communication responsible for concurrency between all clients and the server

the cell change message is for managing which display is showing

the cell setup is the only class in the package that is not a client to server message

these messages are divided up by display they work for

Visual Paradigm for UML Standard Edition(Worcester Polytechnic Institute)
UCViewer2DCellChangeMessage
-displayType
UCViewer2DCellSetup
UCViewer2DLoginNameChangeMessage
-loginText
UCViewer2DListChangeMessage
-selectedTask
UCViewer2DLoginPasswordChangeMessage
-passwordText
UCViewer2DTaskSelectMessage
-selectedTask
UCViewer2DEditNameChangeMessage
-nameText
UCViewer2DProjectSelectMessage
-selectedProject
UCViewer2DEditDateChangeMessage
-dateText
UCViewer2DProjectSelectChangeMessage
-selectedProject
UCViewer2DEditHoursChangeMessage
-hoursText
UCViewer2DEditDescriptionChangeMessage
-descriptionText

Illustration 3: UML diagrams of Client and Common packages

Design

The first step taken in designing the application was to determine what exactly a task manager accomplished. A set of attributes was agreed upon for a task and for a project of tasks. In order to discern which attributes would be important, several task managers were researched, like SourceForge and the Mylyn plug-in for Eclipse, and common attributes shared between their definitions of a task were found. Next, a backend abstract framework was designed for interfacing with the UI, which would later be extended to access and control an external task manager. The idea being that the framework could then easily be tailored to interface with various task managers.

After the framework was designed, the user interface had to be outlined. We came to the conclusion that access to a list of projects or a list of tasks from a project was needed, as well as the ability to edit those tasks. Allowing multiple users and login capabilities was also needed. From these needs it was decided to have a central window that corresponded to the application, which could manipulate multiple views that it could switch between, i.e. one for the project list, one for the task list, and so on. This type of windowing system is easily provided using the AWT package in the Java library. The UI was designed to work similarly to a `JWindow` with several `JPanels` to switch between.

The last step in the design phase was applying it to the client-server setup that Project Wonderland employs. The UI functionality was placed within the client package because the server side cells had no need to access the UI. In addition, the code for the Server was placed in the server package to mirror the client cell. Finally, a system of messages was added to the common package to talk between the client and server cells. These messages were simple in design, but how the client and server interpret each is separate and distinct. The messages would often be sent to the server when something was selected through the UI, or a variable field was changed. When a message was received by the server, each client would then be notified and updated accordingly. This allowed us to ensure concurrency between multiple client applications since each client application was kept up to date as long as the server was kept up to date.

Graphics

AWT is a windowing toolkit that is primarily used for the creation and manipulation of windows and panels, as well as some simple graphics, i.e. shapes, lines, colors. Unfortunately, AWT is not a package devoted to creating graphical user interfaces. Swing is a project within the Java Foundation Classes with the purpose of creating GUI components that can be integrated easily together with a common look and feel. The Swing project contained a simple component for all of the UI capabilities required for the project: List Boxes, Buttons, Text Boxes, Labels, and so on. Swing components were decided upon for the UI in the design.

After the design was complete, work started with implementing the abstract framework into the UI. The ease of use that Swing provides with its components and framework made the UI a quick task and it was ready to test within days. This excess of time seemed like a blessing until it was discovered that Project Wonderland v0.4, the build being used, does not support Swing. According to a release from the Project Wonderland team at Sun[15], Project Wonderland v0.5 is to have Swing embedded into its architecture. This was exciting, but seeing as the project was scheduled to be completed by then, we were forced to rethink our design of the UI.

With the Swing project a non-option, another avenue was sought for our UI: AWT. As AWT is merely a windowing toolkit, with limited graphics capabilities, the components we were planning on taking advantage of through Swing had to be imitated. AWT allows for mouse and key listeners to work with each UI panel that we implemented to enact control over their components. Mouse listeners took care of focus and buttons, and key listeners filled in text at the current focus, if applicable. Each component had to be drawn and have scaling handled, which isn't very costly, but did result in a very inefficient amount of code. The initial premise of solving the Swing problem was to abstract all the UI so it could easily be replaced later, which was not done initially, but was accomplished in the final version of the project.

Controller

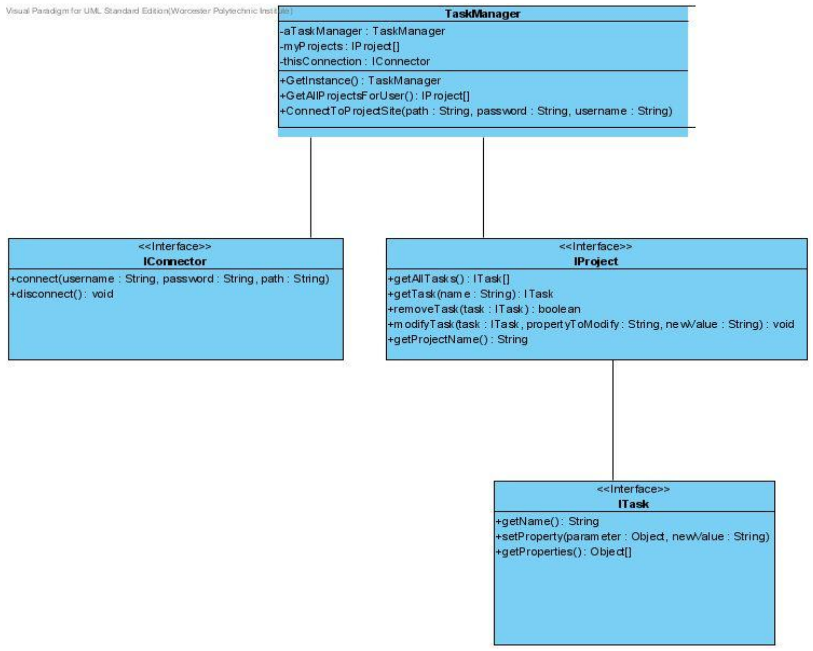
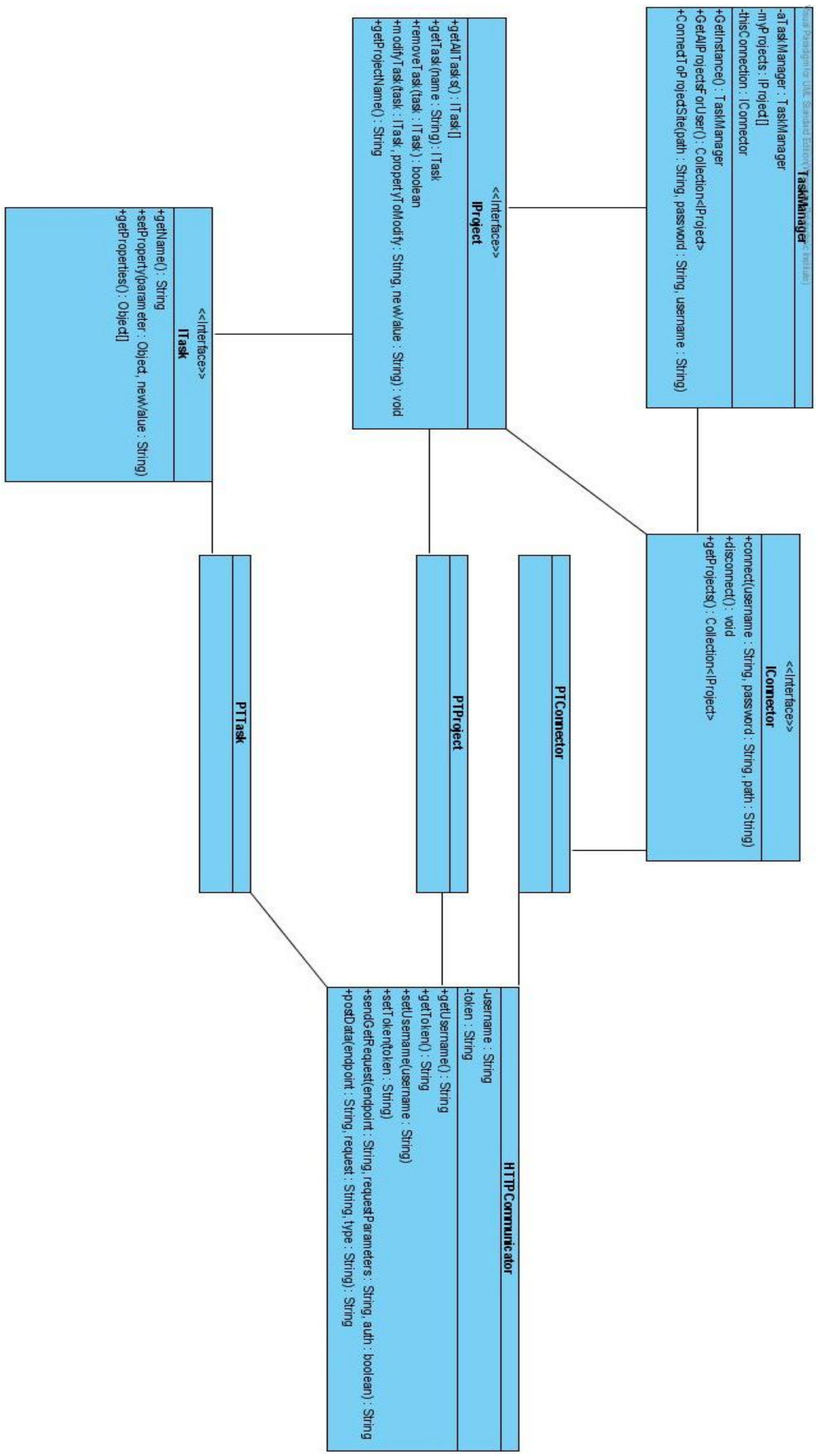

Visual Paradigm for UML Standard Edition(Worcester Polytechnic Institute)
TaskManager
-aTaskManager : TaskManager
-myProjects : IProject[]
-thisConnection : IConnector
+GetInstance() : TaskManager
+GetAllProjectsForUser() : IProject[]
+ConnectToProjectSite(path : String, password : String, username : String)
<<Interface>>
IConnector
+connect(username : String, password : String, path : String)
+disconnect() : void
<<Interface>>
IProject
+getAllTasks() : ITask[]
+getTask(name : String) : ITask
+removeTask(task : ITask) : boolean
+modifyTask(task : ITask, propertyToModify : String, newValue : String) : void
+getProjectName() : String
<<Interface>>
ITask
+getName() : String
+setProperty(parameter : Object, newValue : String)
+getProperties() : Object[]

Illustration 4: The Initial Design of the Controller Package


TaskManager
-aTaskManager : TaskManager
-myProjects : IProject[]
-thisConnection : IConnector
+GetInstance() : TaskManager
+GetAllProjectsForUser() : Collection<IProject>
+ConnectToProjectSite(path : String, password : String, username : String)
<<Interface>>
IProject
+getAllTasks() : ITask[]
+getTask(name : String) : ITask
+removeTask(task : ITask) : boolean
+modifyTask(task : ITask, propertyToModify : String, newValue : String) : void
+getProjectName() : String
<<Interface>>
ITask
+getName() : String
+setProperty(parameter : Object, newValue : String)
+getProperties() : Object[]
<<Interface>>
IConnector
+connect(username : String, password : String, path : String)
+disconnect() : void
+getProjects() : Collection<IProject>
PTTask
PTProject
PTConnector
HTTPCommunicator
-username : String
-token : String
+getUsername() : String
+getToken() : String
+setUsername(username : String)
+setToken(token : String)
+sendGetRequest(endpoint : String, requestParameters : String, auth : boolean) : String
+postData(endpoint : String, request : String, type : String) : String
Illustration 5: UML Diagram of Final Controller Package

In order for the UI to function, it became necessary to create a controller class that would allow for the UI to communicate with some external task manager. The main requirements for the controller package were that it be able to manage a list of projects, and in addition manage the list of tasks within each of these projects. The ability to add and remove tasks, modify the information held in a task, and view task information was required for the UI to function correctly. In order to implement this functionality we chose to design three interfaces which would define the required methods such as getTask, editTask, etc.

These three interfaces were the `IController`, `IProject`, and `ITask`. The responsibilities for the `IController` were to allow a user to login, and also get the list of tasks that the user has access to. These three interfaces were all managed by a `TaskManager` class which was designed to be a Singleton since there should only be one `TaskManager` instance created for each person using our application. In addition, in order to make our application more extensible the only change required in the program should be to change the type of `IController` created in the `TaskManager` to the implementation that functions with the backend the programmer wishes to use. Next, the `IProject` interface defined the functionality that is required of a project that is to be used in our application. It is required that each project has a name, and that the project manages a list of `ITasks` that contain the actual information relevant to specific tasks. Classes that implement `IProject` are required to be able to modify the data in each of the tasks, as well as add them and remove them to and from the system, respectively. Finally, the `ITask` interface specified the methods required of a basic task in our system.

This included getting the properties of the data held in the task, as well as being able to change the properties within the task. In order to implement these functions some type of backend was required to store the task data so that we could retrieve it. We attempted a number of solutions including SourceForge, XPlanner, and ultimately, Pivotal Tracker.

We tried to make our program as extensible as possible so that it would not be difficult to allow our application to run using various backends. It was important that we did this since it became necessary for us to experiment with many different applications. Our initial plan was to use Sourceforge but problems arose with the API that could not be corrected and as a result we were forced to research other options.

Since SourceForge was a Simple Object Access Protocol (SOAP) Web application we tried to find others that were similar since that is what we were most familiar with at the time. This led us to the XPlanner application which was popular for its ease of use and simple Web API. Installing XPlanner proved to be fairly reasonable and we quickly got it running on a local computer. Unfortunately, the SOAP version of the Web API for XPlanner was built to work with the Java API for XML Based Remote Procedure Calls (JAX-RPC) which was only compatible with older versions of Java and has mostly been phased out in favor of the newer Java API for XML Web Services (JAX-WS.) This led to many difficulties and after numerous attempts ultimately failed as a choice for a backend. We did temporarily build a functioning controller package to work with the database that XPlanner utilized but it was ultimately abandoned for our next choice in Web application backends.

The final solution to our issue of what application should be used for our Task Manager backend came from the recommendation of our advisor, Professor Pollice. He had recently been looking into a simplistic Web based project manager application that was very easy to use and setup. This Web application was called Pivotal Tracker and fortunately for us it ran on a RESTful Web services backend. This allowed us to get information about the tasks stored in Pivotal Tracker in the form of XML data which we could then parse and store in our task manager application. This was exactly what we had been looking for and we quickly set to work implementing our controller package with the new Pivotal Tracker backend. We created a few classes `PTConnector`, `PTProject`, and `PTTask` which were the implementations of our three interfaces. In addition we created a class, `HTTPCommunicator` which took care of communication between our application and the RESTful Web services of Pivotal Tracker. The two main functions in `HTTPCommunicator` were `sendGetRequest()`, and `sendPostRequest()`. These were used to either acquire information from the Web services, or alter data on Pivotal Tracker by sending requests to modify the data accordingly. Everything ended up functioning correctly with the basic project management aspects of our application but unfortunately, the login manager ultimately was not able to be implemented. This is due to the fact that Pivotal Tracker uses hash keys to allow users to access the various task data. Each user can only acquire their key by logging in to the Website so that is how they manage the accounts. Once acquired, though, this key can be input into a configuration file in our application which then uses it to access the appropriate account data. The final step to make our program work was to parse the XML data we had acquired from Pivotal Tracker's RESTful services. This is discussed in the next section.

XML

There are two common types of XML parsing: the Document Object Model (DOM), and the SAX[13] model. SAX, or Simple API for XML, was one of the first widely adopted APIs for XML, but from personal experience it was found to be buggy and often fond of throwing exceptions for no good reason. The DOM works through holding an entire XML document as a Java object, and then it can access the entire tree structure of the document, which is done using `XPaths`. An `XPath` is a path that points to part of an XML file, using tag names and other special syntax. This is the XML parsing method that was employed for decoding the messages sent by Pivotal Tracker in response to the database queries sent by the project.

The normal source for the XML document in the DOM is to open a file from a provided file path, but the project needed to pull the Document object from a string of XML. This required the source to be set up through a `StringReader` that fed in the information to the file, which could be parsed by the `XPath` and the Document object. This allowed the project to decipher the answers to the queries of the Pivotal Tracker database, which were then used to update the UI content and in turn the server data.

4. Results and Analysis

Several steps were taken to achieve the backend package contained in our final product. First, what a task is was established and the abstract package was set up so it could be used while the UI was being developed simultaneously. To test functionality, an implementation of the Controller package was created to read a project stored in local text files. This allowed for work on our UI functionality to continue and also provided a simplistic working backend for testing purposes.

Following our textual backend and our failed attempts at getting the XPlanner software to work, a version of the backend was created to directly access a database over the Internet. This version of the backend was not necessarily complete, but provided an opportunity to work with a system with its own definition of a task, even though it had no functionality of its own. This wasn't considered a final solution, but it allowed for progression from text based data storage to an actual database backend. This paved the way for transferring data through Web services.

Pivotal Tracker was chosen in the end as the system the backend would be accessing, and once again the backend was modified. Tweaking needed to be done to finalize the abstract package and accessing that package, but conversion to Pivotal Tracker was primarily smooth sailing. It did require a series of XML parsers to be written for reading the messages from the system, but those came together nicely in the final stages of the project.

The plan for the user interface was to use simple Swing objects in a series of easy to use layouts. These goals were diverted at the discovery of Swing not being part of Project Wonderland v0.4, and a simpler UI was designed that only used the AWT libraries. This made for much more complicated and dense code without as much functionality but it still came together as a responsive, working interface.

The majority of the design was implemented rather smoothly into the final product, with minor hiccups involving the Wonderland API and the various graphics and backend problems. A great success of the design was the abstracted backend, which made testing the product very easy. The ability to easily transition between several testing backends while the final Pivotal Tracker backend was being worked on allowed progress on both the UI and backend to proceed smoothly. This aspect of the design will also make the project easier for a later group to extend. Possible extensions include the ability to work with a different task management system, or maybe a later build of Pivotal Tracker.

Not all aspects of our initial design were able to make it into the final product though, and several features are still a possible addition to a later version. One of the most important of these features is the ability for the user to log in as a user in the task management system. Presently, the user is supplied internally within the extended backend code for the prototype. The desired functionality would be to allow for the login panel of our application to communicate directly with the backend's login management system.

Another aspect of the design that did not make the prototype is the use of the Java Swing libraries for the user interface. This is simply something that couldn't be achieved because it is not a part of the Project Wonderland v0.4 API. The Swing libraries are scheduled to be a part of Project Wonderland v0.5, so if a build of this project were to be created after that release, then it would be possible to implement Swing components into the panels of the user interface.

Finally, the last issue we encountered when creating our application was the inclusion of unit tests. This was a result of the way Project Wonderland works. Since all input is handled through Project Wonderland, there is no way to send virtual input to the application to test it. The only way to test the application is to actually run Project Wonderland and test the application manually. It is important to note that even Project Wonderland itself does not have any unit tests.

5. Future Work and Conclusions

The first priority for the future of this project would be to make this application compatible with Project Wonderland v0.5. Project Wonderland v0.5 is mainly a release to add more features to Project Wonderland, so to make this project compatible would require only one change. In Project Wonderland v0.4, messaging can be done either by packing up the information in an object yourself, or by using Java's object serialization. In Project Wonderland v0.5, though, messages can only be packed by object serialization. In order to comply with Project Wonderland v0.5, we would need to change our messaging to use object serialization.

Another important feature that should be looked into for future work on our project is the development of a more robust Controller package that would allow for added functionality in our system. Although, Pivotal Tracker works for our project as a proof of concept design, it only provides basic task management functionality. It would be an interesting feature to allow for the sorting and filtering of tasks using our system. In addition, it would be beneficial to implement a login system that would interface with an appropriate task management backend since Pivotal Tracker does not allow for this. The SourceForge API would be able to allow for all of this functionality, so that may be something a future team could look into.

Implementing Swing into the project wouldn't change the design much, but would clean up the code and user interface greatly. Already the UI is set up to use interchangeable views that work to support the framework this project developed. If the project was converted to use Swing, each of these views would become an extended version of `JPanel`. This would remove any hard-coded paint functions, and all the UI code would then be handled by extended components provided by the Swing package. The several abstracted components already implemented, the `ButtonBar` and the `ListViewer`, would transition to `JButton` and `JListScroller`. All responses made by the interface are already implemented with `MouseListener` and `KeyListener`, which would change to just `MouseListener`, as the `JTextBox` component can handle keyboard listening itself.

The conversion to using Swing for the interface instead of only AWT would not require large changes in program function or design, merely aesthetic and efficiency changes. The benefits here include a smoother loading interface that is easier to use and more familiar to the average user, as well as a more efficient program simply for lack of extraneous code. Upgrading this project to Swing would be a very beneficial and appropriate change, which should and hopefully will occur when Project Wonderland v0.5 is released.

When this project was being discussed the concept of implementing an optional 3-Dimensional interface that worked off of the virtual environment provided by Project Wonderland. The idea was that a "virtual office" could be simulated and a project leader could simply interact with a person's avatar and be able to assign project-specific tasks to them. This virtual representation would also allow for a visual display of the relationships between certain tasks and the avatars of the people involved in those tasks. For example, if a project had nine people with three tasks, each task with three people working on it, a visual display might show three triangles linking the avatars of each task group.

Discussions about possible display tokens, varying colors and connecting lines, and other varying interface idea were had, but no final conclusion about how the displays would happen was decided upon.

This notion of a 3-Dimensional display revolved heavily upon a mouse-click based system for interactions with the interface. It was decided that the team should write up a design for an abstract controller that would allow for the connection of a `MouseListener` class, a visual display of the information, and the framework for handling the information itself. This way an implementation of the code could exist in the future that allowed for someone to link up our framework easily with the Wonderland project, while providing easy access to visual display and user control.

The goal of our project initially was to explore the usefulness of a virtual office or project team room. The reason a 3-Dimensional display of the task information for a project was discussed at all was because it was considered a possible route to accomplishing the project goal. The team agreed that to accomplish a 3-Dimensional display, a framework would need to be created and tested, preferably on a 2D display that was proven to work already. The project has accomplished this initial step of a 2D display and framework, but the 3D design was shelved and left to be a possible goal of a future project.

Known Issues

Currently, there are a number of small bugs in the implementation of our UI. Because we were forced to use AWT as opposed to Swing, we were unable to use real text boxes, and as a result text can run off of the end of the smaller text boxes (the description field in the task editor wraps correctly). The buttons in our application are also not as responsive as we'd like because they are not actually buttons. When a mouse clicks on the screen, it checks through a list of known button locations to see if the location it clicked is on a button or not. This method works to register the majority of button clicks, but not all, and is very inefficient. Both of these bugs would be fixed easily by moving to Swing.

Also in our application, the user is unable to double click on a task or a project to select it. Ideally, the user would be able to double click on a project in the project list and that project's tasks would be displayed, without having to hit the "Select" button in the lower left. The same should also apply to tasks in the task list, so that the user should be able to double click on a task and have it open the View/Edit screen.

Finally, there might be a bug in our application with respect to Web containers. We found that on one specific computer we were unable to launch Project Wonderland from the Web application. When we attempted to run it, a security exception was thrown with very little error tracing. As this wasn't able to be reproduced on other computers, we are unsure if this is even a bug, but if it is, it is most likely that we configured something incorrectly in Jetty.

Design Decisions for Login

In order to clear up any ambiguity in the implementation of our login system, this next section will deal with the design decisions that lead to the current functionality. The most important factors that impacted the way login worked were based upon the limitations of the backend we chose, and in addition the way that Project Wonderland handled synchronization.

The first factor that influenced the design of our final controller package was the way Pivotal Tracker handled accessing its RESTful services. This was accomplished by acquiring a hash code from your login at the Pivotal Tracker website and then supplying that hash code each time the user wishes to access the Web services. This design allowed for Pivotal Tracker to supply secure access to the projects since each user would have to login to their account to get their hash code in order to access their projects. We were unsure about how to integrate this into our project since we were expecting users to supply user names and passwords to access the task manager, but Pivotal Tracker did not support this method. We considered the option of having users supply their hash codes upon accessing the interface but since we were implementing the keyboard input functionality ourselves and there was no copy and paste option, we were concerned that users of our software might find it cumbersome to enter in a 28 character key. Since our project was designed with the idea that a group would be using the software in meeting environments, we decided that it made more sense to have them input their user name and hash key into a configuration file before deploying their Project Wonderland build. That way, their projects would be available to them immediately upon the launch of Project Wonderland without requiring them to log in. Also this would provide for an easier method of entry of their hash code for Pivotal Tracker using their preferred text editor.

Our decision to use this method of login access was also influenced by Project Wonderland's handling of synchronization. Presently, this is done by sending messages to the server each time the client is updated in order to ensure that each instance of Project Wonderland running our application displays the same information as the others. With this current implementation, if our program allowed for multiple users to login then these messages would cause our application to function incorrectly. For example, if two users were logged into our application with differing logins, then when either user attempted to change the panel they were viewing in our application, the application that the other user was viewing would change as well since Project Wonderland's messaging system enforces a synchronized view between clients. This would make it impossible for two users to use the application separately under one instance of Project Wonderland since any change that one user makes will affect the display of all other users. Rewriting the way that messages are handled would most likely be able to remedy this situation.

References

1. "Apache Ant User Manual." Apache Ant. 16 Feb. 2009 <http://ant.apache.org/manual/index.html>.

2. Charette, Robert N. "IEEE Spectrum: Why Software Fails." IEEE Spectrum Online: Technology, Engineering, and Science News. 17 Feb. 2009 <http://www.spectrum.ieee.org/sep05/1685>.

3. Dickey, Michele D. "Three-dimensional virtual worlds and distance learning: two case studies of Active Worlds as a medium for distance education." British journal of educational technology 36 (2005): 439.

4. Horstmann, Cay S. Big Java. New York: Wiley, 2005.

5. "Java.awt (Java 2 Platform SE v1.4.2)." Developer Resources for Java Technology. 20 Feb. 2009 <http://java.sun.com/j2se/1.4.2/docs/api/java/awt/package-summary.html>.

6. "JDK 5.0 Swing (Java Foundation Classes (JFC))-related APIs & Developer Guides from Sun Microsystems." Java 1.5 Swing API. 21 Feb. 2009 <http://java.sun.com/j2se/1.5.0/docs/guide/Swing/index.html>.

7. Kohler, Thomas, Kurt Matzler, and Johann Füller. "Avatar-based innovation: Using virtual worlds for real-world innovation." Technovation (2009). Technovation. 21 Jan. 09. 19 Feb. 09 <http://www.sciencedirect.com/science/article/B6V8B-4VF0XP0-1/2/e9240c0e542531c76709fe7af5274a16>.

8. "NetBeans IDE - Base IDE Features." NetBeans. 24 Feb. 2009 <http://www.netbeans.org/features/ide/index.html>.

9. "Pivotal Tracker: API." Pivotal Tracker API. 17 Feb. 2009 <http://www.pivotaltracker.com/help/api>.

10. "Project Wonderland New Cell." TWiki- Project Wonderland New Cell. 21 Feb. 2009 <http://wiki.java.net/bin/view/Javadesktop/ProjectWonderlandNewCell>.

11. "Project Wonderland Web Administration." Twiki- Wonderland Web Admin. 18 Feb. 2009 <http://wiki.java.net/bin/view/Javadesktop/ProjectWonderlandWebAdminAdmin>.

12. "Project Wonderland Visons and Goals." Twiki- Wonderland Visons and Goals. 21 Feb. 2009 <http://wiki.java.net/bin/view/Javadesktop/ProjectWonderlandAbout#VisionAndGoals>.

13. "SAX." SAX API for XML. 20 Feb. 2009 <http://www.saxproject.org/>.

14. "Statistics over IT Failure Rate." IT Cortex Web Portal. 17 Feb. 2009 <http://www.it-cortex.com/Stat_Failure_Rate.htm>.

15. "Wonderland Roadmap." Twiki- Wonderland Major Architecture. 20 Feb. 2009 <http://wiki.java.net/bin/view/Javadesktop/WonderlandRoadmap#Release_0_5_July_2008_Major_Arch>.