Project Number. GFP1004

WPI Suite Assignment Submission Module

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:
Richard DiCroce
James Lawrence
Chance Miller

April 21, 2011

Approved:

Professor Gary F. Pollice, Major Advisor

1. Client-Server
2. Automated Homework Analysis
3. WPI Suite

Abstract

WPI Suite, a student-developed software collaboration tool, was in need of a module that facilitated the submission and processing (i.e. testing and grading) of student assignments. The Assignment Submission Module became the solution. Through WPI Suite, the Assignment Submission Module provides a mechanism for students to submit files to professor-specified assignments over a network, and for both professors and their assistants to provide manual and automatic feedback.

Acknowledgements

Gary Pollice, Project Advisor.

Web-CAT Project, for providing a basis for the data model used in our project.

Table of Contents

List of Illustrations

1. Introduction

The intent of our project was to create a system for students studying computer science to submit assignments for classes, and have this system that automatically provides feedback based on professor supplied specifications. In addition, we needed to have it function as a part of WPI Suite[1], a student-developed software collaboration tool. From these requirements, the WPI Suite Assignment Submission Module (ASM) was developed.

Working with WPI Suite provided us with a set of obstacles to overcome. The biggest challenge provided by WPI Suite was the lack of a real server. As outlined in the Background section, it was effectively nothing more than a database with next to no security system. This, obviously, would not be acceptable in a system which needed to store students' grades and homework submissions. Thus, we created a server of our own. In addition to being secure and having various permissions to limit user access, the server needed to have scalability, should it be used by many users at once; modularity, so it could be updated when needed; and fault tolerant, so that the data on the server could be accessed at all times. The details of how this server was implemented can be found later on, in the Methodology section of the report.

Of course, there was more to be done than the server; an application had to be developed. We looked at existing homework submission solutions Web-CAT[2], CCC Turnin[3], and Web Turnin[4] for inspiration and analysis. The Background section of this report discusses the information we discovered from these pieces of software, as well as why they were not viable solutions to our problem. The second half of the Methodology section explains how we developed the user interface, in addition to the file transfer system itself. Although we faced

[1] TeamForge : Project Home. April 27 2011. <https://sourceforge.wpi.edu/sf/projects/wpisuite/>
[2] Web-CAT Research Server Front Page. April 27 2011. <http://web-cat.cs.vt.edu/>
[3] Using the 'turnin' Program. April 27 2011. <http://web.cs.wpi.edu/Help/turnin.html>
[4] Turnin Instructions. April 27 2011. <http://web.cs.wpi.edu/~kfisler/turnin.html>

some setbacks, as outlined in Sections 4 and 5, after numerous revisions, we ultimately managed to create an Assignment Submission Module for WPI Suite which met all of our requirements for success.

2. Background

2.1 History of the WPI Suite Project

WPI Suite is the successor to the defunct WEBFOOT[5] (WPI Environment Built For Object-Oriented Teams) project. The WEBFOOT project aimed to provide a bridge between SourceForge[6] (a comprehensive suite of developer collaboration tools) and Eclipse[7] (an integrated development environment) by creating an Eclipse plugin. Unfortunately, this effort was unsuccessful. The maintainers of the WEBFOOT code found that the SourceForge and Eclipse APIs (Application Programming Interfaces) were changing very rapidly. They concluded that they were expending too much time and effort fixing breakages due to these API changes, leaving little time to add new features to WEBFOOT or improve existing ones. Consequently, in 2007, WEBFOOT was abandoned.

In early 2009, Professor Pollice launched a new effort to replace WEBFOOT, known as WPI Suite. WPI Suite has a far more ambitious goal than the WEBFOOT project. WEBFOOT failed because it was sandwiched between two APIs that the WEBFOOT developers had no control over, so Pollice decided to gain control over one of those APIs by building a SourceForge replacement. (There were also other motivating factors, but they are not relevant to this discussion.) Initial development was handled by a team of Software Engineering students as their term project. Their code, with some minor bug fixes, was released in May 2009 as WPI Suite 1.0.

[5] TeamForge : Project Home. April 27 2011. <https://sourceforge.wpi.edu/sf/projects/webfoot/>
[6] CollabNet TeamForge – The Agile ALM platform for distributed teams. April 27 2011. <http://www.open.collab.net/products/ctf/>
[7] Eclipse – The Eclipse Foundation open source community website. April 27 2011. <http://www.eclipse.org/>

Over the next two years, several more Software Engineering teams and MQPs added additional functionality to WPI Suite, leading to the release of WPI Suite 2.0 in May 2010 and the impending release of WPI Suite 3.0 later this year. WPI Suite now includes modules for project administration, tracking requirements and defects, measuring various types of metrics, and more.

2.2 Technical Details of the WPI Suite Project

One of the ASM project team's first tasks when the ASM project commenced was to study the existing WPI Suite code base and determine how the ASM could be added to WPI Suite. It was known, well in advance, that this would be problematic due to the architecture of the WPI Suite Client.

The most significant problem was the security of system data. At the time WPI Suite was created, security was not a primary concern. Consequently, WPI Suite was designed to function in a largely server-less manner, with only an SQL server required for operation. In other words, each client connected to and manipulated the database directly. This kept the design simple and reduced the amount of code that needed to be written, since no server component would exist. However, a corollary of direct connections to the database is that, with minimal effort, anyone can gain raw, full, read/write access to the entire database, regardless of any protections implemented in WPI Suite. Such a security hole is unacceptable in the ASM project, given that the module will be involved in grading student assignments.

Even if security were not a problem, the lack of a server still would have been. In order to be fair, the ASM's testing environment must be uniform for all student submissions. This effectively requires the existence of a server beyond a simple database. Again, WPI Suite had not been designed with a real server component in mind, so the team knew from the beginning that this functionality would have to be retrofitted onto the existing client.

2.3 Existing Systems

Before beginning an effort to write the entire ASM from scratch, the team studied some existing systems that are similar in nature to the system the team constructed. Surprisingly, not much work appears to have occurred in this area.

2.3.1 Web-CAT

The Web-CAT system is the nearest alternative to the ASM in terms of functionality. Like the ASM, Web-CAT's goal is to provide an automated unit testing and grading system for student programming assignments. At first glance, it seemed as though Web-CAT provided exactly what was required. However, further analysis showed that it was not realistic to reuse code from Web-CAT.

The most significant problem would have been integration with the rest of WPI Suite. One of Web-CAT's explicitly stated goals is that it should be a web application and should not require the installation of a special desktop client. Web-CAT's code reflects this goal, and the team concluded that attempting to integrate code from a web application into a desktop application would be more trouble than it was worth. Additionally, development of Web-CAT has stagnated. The project's last release was in 2008, and the team did not want to depend on code from any dormant or abandoned projects.

Research into Web-CAT was not a complete loss, however. Although the team determined that no code could be reused, the team also determined that several Web-CAT concepts would be useful when building the ASM. For example, Web-CAT makes a distinction between an assignment and an assignment offering. An assignment is a non-temporal entity; it includes such information as a name, a description, and files necessary to test submissions. Assignment offerings are temporal entities that are created from assignments. They "inherit" all of the assignment's data, but additionally include temporal information, such as a due date. This allows an assignment to be created once, and then reused repeatedly each time a class is offered without requiring the instructor to re-enter all of the assignment data and re-upload all of the assignment files. Web-CAT makes a similar distinction between courses and course offerings for the same reason. The team decided to emulate this model when building the ASM.

2.3.2 Turnin

WPI already has two existing homework submission systems. Confusingly, both are called Turnin. One operates as a command-line client on the CCC servers, while the other is a web application. The team evaluated both of these existing systems to determine whether they would be useful. In both cases, the conclusion was that no reuse was possible. As with Web-CAT, integration with either Turnin was judged to be problematic at best. Additionally, integration with either Turnin would not have provided as much benefit as integration with Web-CAT, since neither Turnin supports automated testing and grading of student submissions.

3. Methodology

3.1 Server Architecture

The WPI Suite Server was very carefully and deliberately designed to fulfill an ambitious set of goals that were sometimes at odds with each other. Among these goals were modularity, scalability, and fault-tolerance.

It is important to note that the term “WPI Suite Server” can be quite ambiguous, and it could be referring to many different scopes of code depending on the context it is used in. For clarity, in this document, the term will refer to the *entire* server, including all nodes and modules. At this time, there are two types of nodes, one of which has several different modules. These nodes and modules will be referred to by their individual names throughout the paper.

3.1.1 Clustering

At this time, WPI Suite is not in widespread use, but the team proceeded on the assumption that it eventually will be. Therefore, simple handling of increased workloads became a key design objective. The team concluded that a clustered design would be the best way to achieve this. If the workload became too heavy for the cluster to handle, then the problem could be solved by simply spawning additional nodes and expanding the cluster.

Designing the cluster on paper was simple, but actually implementing it in code proved more difficult. The team investigated two clustering frameworks in locating a library to provide the necessary support. The first was Hazelcast.[8] Hazelcast provides a simple way of making the same data available to multiple programs running on separate Java Virtual Machines (JVMs).

[8] In-Memory Data Grid – Hazelcast – Home. April 27 2011. <http://www.hazelcast.com/>

This option was pursued for some time until the team determined that Hazelcast could not guarantee the integrity of the shared data. This is because each node in a Hazelcast cluster "owns" some pieces of shared data. Hazelcast can be configured such that multiple copies of the same data exist on different nodes, but there is a limit to this redundancy and Hazelcast does not support persisting the contents of most shared data structures to disk. Since fault-tolerance could not be assured, Hazelcast was dropped from consideration.

The second option the team discovered (and used in the final implementation) was Terracotta.[9] The Terracotta name actually encompasses several products made by the company of the same name; here, it refers to the Terracotta Distributed Shared Objects (DSO) product. Terracotta is significantly more powerful than Hazelcast, but also significantly more complex to use. It is, essentially, an aspect-oriented clustering framework that injects itself into code at run-time. This has the benefit of meaning that the use of Terracotta is transparent in the code. Only a few parts of the WPI Suite Server are actually dependent on behavior supplied by Terracotta to function correctly, so, in most cases, the server can be run without it in a development environment. This is a significant benefit, given that instrumenting code and connecting to the Terracotta server adds a significant delay to the start-up process.

Terracotta fits nicely with the goal of having a scalable and fault-tolerant server. Unlike in Hazelcast, the cluster nodes (i.e. the WPI Suite Server nodes, which are, in turn, Terracotta clients) do not store any shared data; the Terracotta server handles this instead. The Terracotta server itself can be clustered, scaling easily from one machine with no persistence or fail-over all the way up to striping shared data across dozens of machines, mirroring that data across dozens of other machines, and persisting all of the cluster's data to disk. This means that, in theory, the size of a WPI Suite Server installation is limited only by the amount of hardware that you can afford to buy. In reality, contention for various locks undoubtedly creates an upper limit, but the team considers it unlikely that this ceiling will ever be reached.

[9] Terracotta. April 27 2011. <http://www.terracotta.org/>

3.1.2 Modularity

The WPI Suite Server is modular on two distinct levels. At a high level, the cluster can contain many types of nodes to support special requirements. In order to expose functionality to clients, the cluster must contain at least one Client Facing Node (CFN), but other node types can exist. The ASM, in fact, contains a separate type of node whose sole purpose is to process assignment submissions. That node will be discussed later in section 3.4 (Submission Processing). At a lower level, the CFN is designed to facilitate easy extension. The remainder of this section will focus only on the CFN.

Even before the project began, the team was already aware of potential future projects that could not be built without a true server for clients to connect to, such as virtual meetings and joint code reviews. The team thus determined that having a modular CFN that would allow easy extension by other project teams was an absolute necessity. Beyond allowing additional modules to be built easily, the team also desired to allow modules to share services with one another in order to promote code reuse.

3.1.2.1 Module Loading

The first step in accomplishing this was finding a way to dynamically load modules at run-time. The WPI Suite Client already had a homebrew framework that aimed to do this. The team evaluated this framework and found it lacking. It does not support truly dynamic loading of modules and is not designed for loading classes from separate module JAR files.

Next, the team considered OSGi[10], as this framework is a de facto standard for building modular systems. OSGi turned out to be problematic on several levels. First, its immense power comes at the price of being immensely complex. The team desired a simpler solution. Second, it provides features that the team preferred not to have. For instance, OSGi supports removing modules at run-time. Allowing this in the CFN would have required CFN modules to become

[10] OSGi Alliance | Main / OSGi Alliance. April 27 2011. <http://www.osgi.org/>

significantly more complicated. Third, and most importantly, OSGi conflicts with Terracotta. Other developers who attempted to use both encountered significant difficulties and were forced to build shims into their applications to solve the problem.

Finally, the team discovered the Java Simple Plugin Framework (JSPF).[11] JSPF turned out to have exactly the balance of simplicity and power the team was looking for. The CFN uses JSPF to discover and load its module JAR files when it starts up and performs some initialization on each module.

3.1.2.2 Module Dependencies

Most CFN modules are comprised of three elements: data models, persistence management classes, and client exports. The details of the first two elements are covered in section 3.3 (Persistent Storage), while details of the latter are covered in section 3.2 (Client-Server Communication). For this section, the important point is that the team desired to allow CFN modules to share these elements as necessary, preferably without requiring extensive amounts of configuration. PicoContainer[12], a dependency injection framework, was used to accomplish this.

As part of the module initialization process, the CFN asks each module to register any persistence management classes and client exports that the module contains. Internally, these are generically referred to as “implementations” since all persistence management classes and client exports are required to specify all functionality they wish to make available in interfaces, then provide classes that implement the interfaces. This was an explicit design decision aimed at reducing coupling within and between modules.

The module is required to specify a “scope” for each implementation it registers. Currently, only two scopes exist. Classes with “application” scope are effectively singletons. They will be instantiated at most once in the lifetime of the CFN process. They will not be

[11] jspf – Java Simple Plugin Framework. April 27 2011. <http://code.google.com/p/jspf/>
[12] PicoContainer – Coding projects. April 27 2011. <http://picocontainer.org/>

instantiated at all if no other implementations depend on them. All persistence management classes and utility classes have application scope. Classes with “session” scope are automatically assumed to be client exports. They will be instantiated exactly once for each connection established by a client to the CFN.

When PicoContainer is asked to provide an implementation of an interface, it checks its registry to find out what class implements the given interface. When it finds a match, it uses reflection to retrieve the available constructors for the implementing class. This is how dependencies are defined: if a class Foo requires an implementation of IBar to function, Foo simply requires an IBar to be passed into its constructor. If a class has multiple constructors, PicoContainer will consider them in descending order by number of parameters. In other words, it will attempt to satisfy as many dependencies as possible, but will fall back to fewer dependencies if the class can function without them and makes an appropriate constructor available.

PicoContainer thus allows the enforcement of two desirable attributes in the module system. First, since only implementations and not instantiations are registered, circular dependencies cannot exist, as this will cause PicoContainer to throw an exception. Second, since one PicoContainer can contain other PicoContainers, a hierarchy can be created that prevents the existence of dependencies that should not occur. This is enforced in combination with the scope mechanic: any implementations with session scope are allowed to depend on other implementations with session scope and on any implementations with application scope. Implementations with application scope, however, are only allowed to depend on other application-scope implementations. Any attempt to make an application-scope implementation depend on a session-scope implementation will cause an exception.

3.1.2.3 Simple Base

Finally, it is important to note that the CFN itself was deliberately designed to be extremely simple. Effectively, all the CFN does is the following:

- Provides clients with an open socket to connect to
- Exposes client exports provided by CFN modules to clients
- Provides CFN modules with a database connection and access to configuration data

The CFN does nothing else. Even basic system functionality, such as user authentication, is implemented as part of a module (specifically, the WPI Suite Commons Module), not as part of the CFN itself. The aim was, again, to reduce coupling and to support easy swapping in the event that future teams wish to develop alternative versions of even basic functionality. For instance, a future team might decide to implement authentication against a Windows domain instead of the database in order to promote tighter integration with other computing assets. Keeping the CFN simple and putting that functionality into a module allows this change to occur by extension rather than modification.

3.2 Client-Server Communication

The process of developing the CFN also involved deciding how the client and CFN would communicate with each other. In arriving at a conclusion, the team considered a wide variety of options. The final choice was made based on a small set of criteria. First, the team desired a protocol that would be transparent and easy to use. This criterion ruled out the use of message passing protocols such as ZMQ[13] or AMQP[14], since those would have required a large amount of boilerplate code to establish connections and serialize/deserialize data. Second, the team desired a protocol that would provide asynchronous, bidirectional communication. Although the team was not certain at the time of selection whether this would be needed for the

[13] Less is More – zeromq. April 27 2011. <http://www.zeromq.org/>
[14] Advanced Message Queuing Protocol. April 27 2011. <http://www.amqp.org/>

ASM, it was anticipated that future teams may need the functionality. This criterion ruled out the use of REST[15], SOAP[16], and all other protocols running over HTTP.[17]

The team concluded that Remote Method Invocation (RMI) was the best fit. This was not the end of the search, however. Standard Java RMI[18] is somewhat cumbersome to use, so the team first opted to use the Cajo[19] library to simplify using it. A significant amount of work had already been done using Cajo when the team discovered that the Java RMI system it is layered atop has some significant flaws. The most important of these is that Java RMI is easily derailed in a real network environment because it requires establishing at least two connections: one in each direction. Establishing a connection from the client to the server is not a problem, but most clients are behind network address translation (NAT) firewalls, preventing the establishment of a connection from the server to the client.

A variety of remedies to this situation were considered. Several options existed that would have allowed continued use of Cajo and Java RMI, but all were found lacking for various reasons. Some would have required extensive amounts of additional code to work around the problem, while others would have been relatively simple to implement but would not have guaranteed proper operation in all cases.

It was at this point that the team discovered the SIMON[20] and Dirmi[21] projects, both of which share the goal of providing an RMI system without the drawbacks inherent in Java RMI. Both of these libraries are very similar to Cajo; in fact, the same spike code was reused with only minor adaptations to test the qualifications of both alternatives. Dirmi was eventually

[15] Representational State Transfer. April 27 2011. <http://en.wikipedia.org/wiki/Representational_State_Transfer>
[16] SOAP. April 27 2011. <http://en.wikipedia.org/wiki/SOAP>
[17] RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1. April 27 2011. <http://tools.ietf.org/html/rfc2616>
[18] Remote Method Invocation Home. April 27 2011. <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html>
[19] The cajo project: Wiki: Home – Java.net. April 27 2011. <http://java.net/projects/cajo/pages/Home>
[20] SIMON – Start – root1.de – Software Engineering. April 27 2011. <http://dev.root1.de/wiki/2>
[21] Dirmi – Bidirectional RMI. April 27 2011. <http://sourceforge.net/projects/dirmi/>

selected over SIMON for several reasons. Development work on SIMON had stagnated, the author of the project is German, which made documentation difficult to understand, and SIMON is licensed under the GNU General Public License (GPL)[22], which is definitely not compatible with the Eclipse Public License (EPL)[23] that WPI Suite is licensed under. Dirmi, conversely, exhibits slow but consistent development, is licensed under the GNU Lesser General Public License (LGPL)[24], and is written by English-speaking programmers who provide solid documentation.

The one significant drawback to using Dirmi was that, unlike Cajo and SIMON, Dirmi does not provide a built-in system for serving multiple exports to clients. To solve this, the team built a simple registry, which was later integrated into the Session system. The Session system itself is also a response to a problem created by Dirmi's internal implementation. Namely, Dirmi does not provide any sort of publicly accessible connection identifier when it passes invocations up to the appropriate object. This means that there is no way to associate the invocation with any globally-stored session data. Since Dirmi also uses a thread pool to execute invocations, the same thread may service multiple clients over its lifetime, meaning that thread-local storage cannot be used to make the association either. The Session system is a workaround for this problem. Essentially, all client exports are instantiated every time a client connects in order to form a new object graph for each client. This is somewhat inefficient, as most exports care little about state beyond knowing which user is logged in. Unfortunately, the team could not find a more efficient way of solving the problem.

The Session object itself merits some special discussion as it contains instantiations of all of the client exports provided by CFN modules. When the client wishes to access a given export, it simply calls the Session and asks for an implementation of the appropriate interface.

[22] The GNU General Public License v3.0. April 27 2011. <http://www.gnu.org/licenses/gpl.html>
[23] Eclipse Public License (EPL) Frequently Asked Questions. April 27 2011. <http://www.eclipse.org/legal/eplfaq.php>
[24] GNU Lesser General Public License v3.0. April 27 2011. <http://www.gnu.org/licenses/lgpl.html>

Dirmi will then provide the client with a largely transparent proxy to the actual object on the CFN. (The only notably non-transparent part of the process is that any methods that can be invoked remotely must be declared as throwing RemoteException to handle the case where something goes wrong with the remote invocation.) The Session object also provides a utility method the client can use to determine if the CFN can provide the given exports without attempting to actually retrieve proxies to these exports. This is used as part of the WPI Suite Client connection process to determine whether the CFN has all of the CFN modules installed that are necessary to support all of the client modules specified in the client's connection profile.

3.3 Persistent Storage

First problem we solved in regards to persistent storage in the WPI Suite Server was determining what type of data were we working with. For the Assignment Submission Module it turned out we were dealing mostly with records and files. The record data managed all the users, courses and assignments. The file data handled all the assignment processing and student submissions. It was decided that when working with a combination of record data and files we would be best served by a database in combination with a file system. For our database we selected MySQL™[25] for its open-source nature and our familiarity with it.

Once we settled on the combination of MySQL and Files for persisting our data we looked at a number of ORMs for managing our database. The three ORMs we investigate were ActiveObjects[26], SimpleORM[27], and Hibernate.[28] SimpleORM was a nice simple ORM with a very short learning curve. However, it turned out not to provide auto-incremented columns for MySQL which we needed for the primary key of many tables within our database. ActiveObjects

[25]MySQL The World's Most Popular open-source database. April 26th 2011. <http://www.mysql.com/>
[26]ActiveObjects: Wiki: Home. April 26th 2011. <http://java.net/projects/activeobjects/pages/Home>
[27]SimpleORM (Hibernate without the complexity). April 26th 2011. <http://www.simpleorm.org/>
[28]Hibernate - JBoss Community. April 26th 2011. <http://www.hibernate.org/>

was also explored as a possible ORM because of its use in the current WPI Suite client. However, ActiveObjects has not been actively updated and does not play nice with plain old java objects. For example ActiveObjects generates its own class at run-time based off of an interface making it hard to define functions for the class. While this limitation is not impossible to overcome we decided there were better solutions. Because of these limitations to SimpleORM and the encapsulation problems with ActiveObjects, we settled on Hibernate which provided the functionality we needed, but had a very steep learning curve.

After we had selected the tools for persistent storage we needed to develop the WPI Suite server's framework for persisting data. We had one main goal for our data persistence framework; we wanted to completely hide the underlying tools (such as Hibernate/MySQL) for persisting the data instead only relying only on an interface for communication with the data storage. This goal was a direct response to ActiveObjects permeating the entirety of the WPI Suite Client making it difficult to replace with better solutions. We achieved this goal by using the following design.

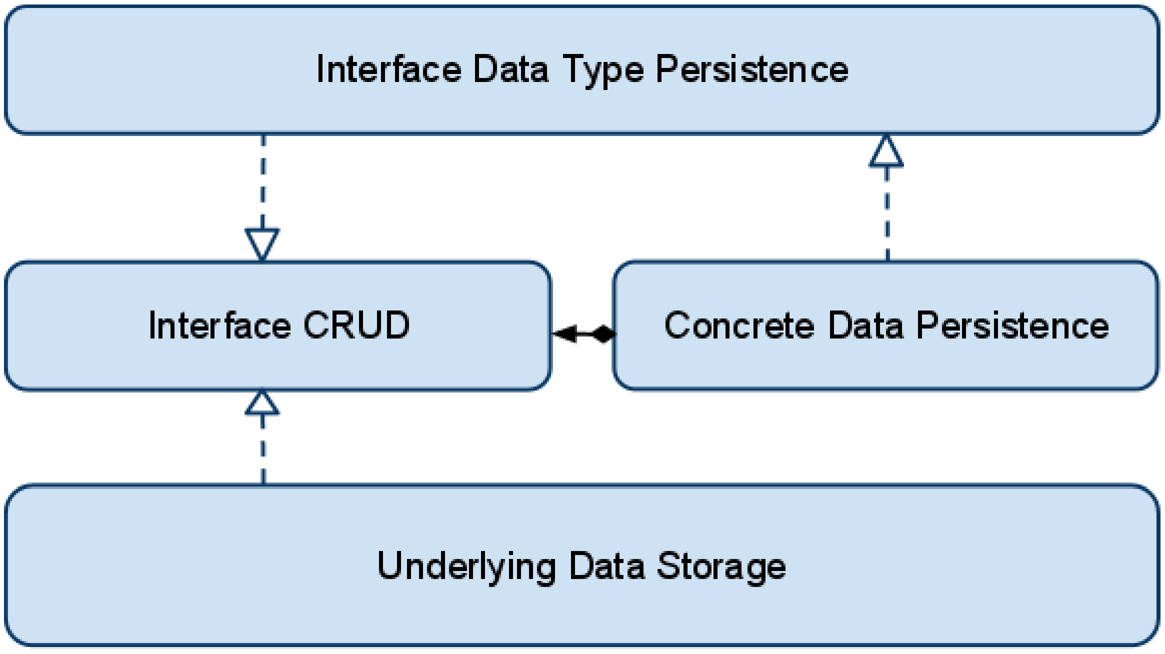


Figure 1: Diagram of the server's data persistence system

In this diagram there are two interfaces and two concrete classes. The CRUD interface means the class the implementing the interface must provide the CRUD functions for the given data type. This must be implemented by both the underlying data storage and the data persistence interface for each data type. The data type persistence interface contains any special functions that are specific to a data type in conjunction with the Data Persistence CRUD functions. The concrete data persistence in the diagram implements all the functions required by the data type persistence interface using the underlying data storage. This allows us to easily swap out different underlying data storage by simply replacing the concrete data persistence class, while hiding the change from the rest of the system.

3.4 Submission Processing

The submission processing portion of our system had a number of issues to resolve. First we had to allow for a diverse set of programs to be used to process submissions, second we needed to prevent programs from consuming system resources indefinitely, and finally we needed to be able to store the results from these programs so the results can be used for further processing.

In order to handle the diverse set of programs we decided on creating a shell-like node within the server cluster. This node, called the Submission Processing Node (SPN), receives a set of commands which have been placed on a queue and begins executing them in order, allowing professors of an assignment to concatenate a series of commands together to process a submission. To prevent the execution of commands from consuming system resources indefinitely each command is given a set amount of time to run. As a command is being executed the standard output and standard error streams are placed into text files which can then be used for further processing. For example, taking the results from executing JUnit[29] on a submission and generating a grade for that submission.

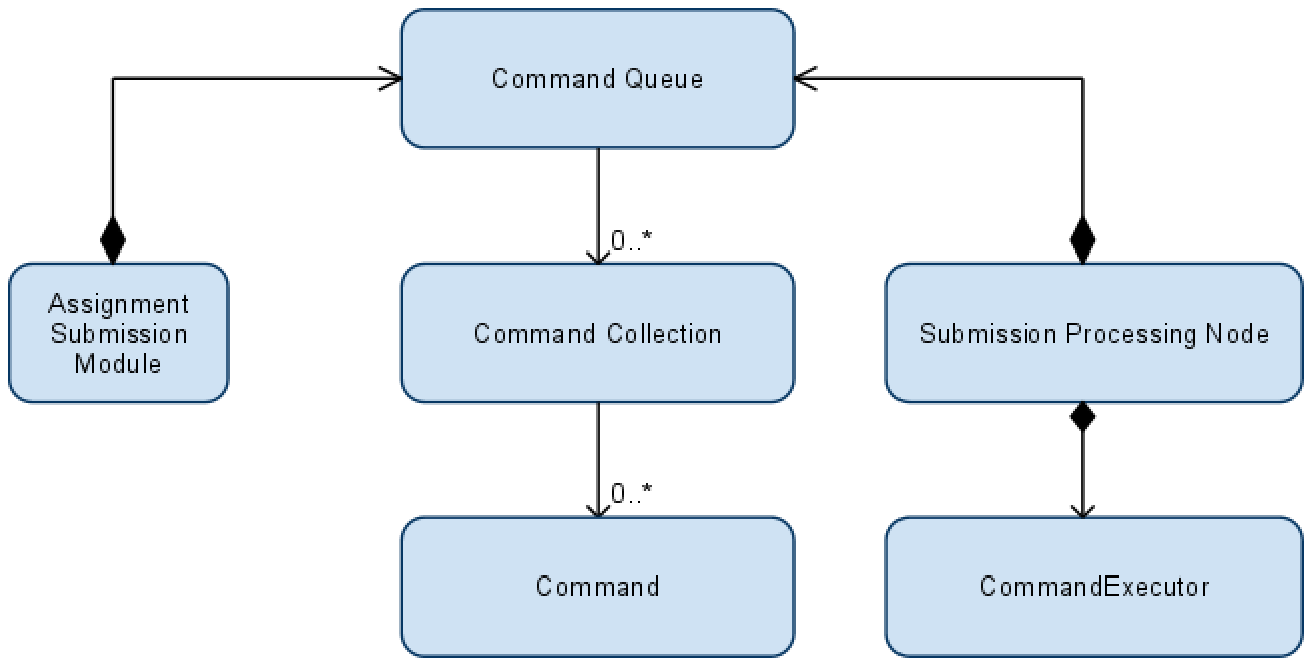


Figure 2: Diagram of submission processing

In order for the Assignment Submission Module to communication with the Submission Processing Node a command queue was created. The ASM places a collection of commands onto the queue which are then removed by a SPN and processed using a CommandExecutor shown below.

[29] Welcome to JUnit.org! | JUnit.org. April 27 2011. <http://www.junit.org/>

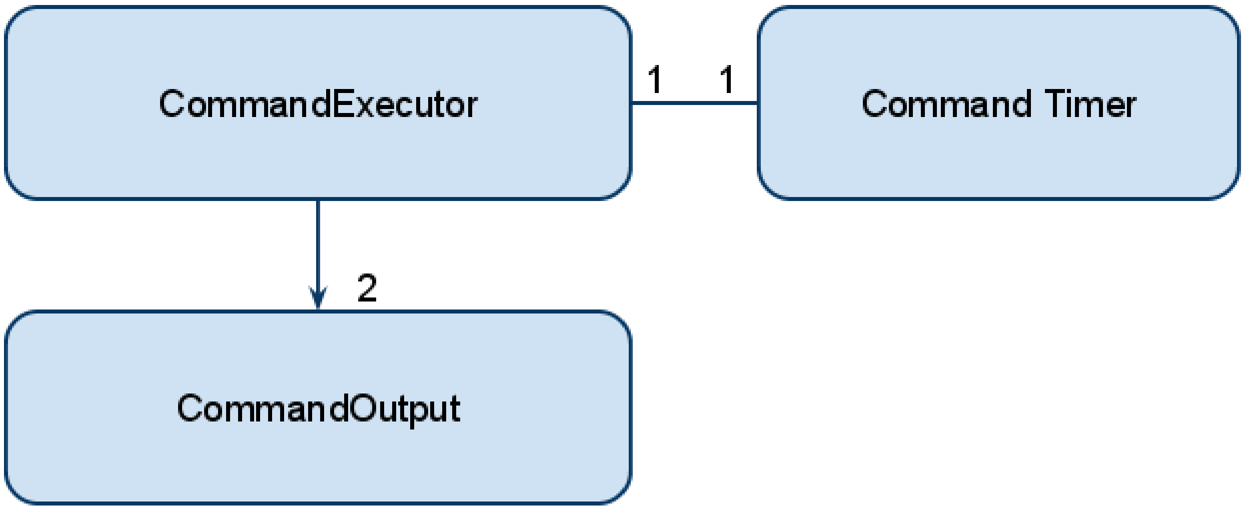


Figure 3: Diagram of command processing

The command executor system is made up of three parts, the command executor, the command timer, and the command output. The command executor is responsible for starting the command timer, creating the process, starting the command outputs, and finally waiting for the process to complete. The command timer is responsible for telling the command executor when to stop waiting for the process to finish. The command outputs are used for writing the standard output and standard error of the process to files.

The commands received by the SPN are generated for each submission using a CommandDescription.xml file which is provided to the system by a professor of an assignment. During a submission is file is read and allows the professor to control what programs and in what order are executed on the files submitted by a student. The command description file is made up of a collection of individual commands each command consists of the following information: maximum run-time, a working directory, an output file name, and the command arguments. Below is an example of the CommandDescriptions.xml file.

```
	<CommandDescriptions>
		<Command>
			<Runtime>60</Runtime>
			<WorkingDirectory>
				<Submission>
					<Directory/>
				</Submission>
			</WorkingDirectory>
			<Output>
				<Literal>Command1.txt</Literal>
			</Output>
			<Arguments>
				<Literal>ant.bat -buildfile</Literal>
				<Compound seperator="">
					<Assignment>
						<Directory/>
					</Assignment>
					<Literal>\adderTests\build.xml</Literal>
				</Compound>
				<Compound seperator="">
					<Literal>-Dstudent.dir=</Literal>
					<Submission>
						<Directory/>
					</Submission>
				</Compound>
				<Compound seperator="">
					<Literal>-Dassignment.dir=</Literal>
					<Assignment>
						<Directory/>
					</Assignment>
				</Compound>
				<Literal>compile</Literal>
			</Arguments>
		</Command>
</CommandDescriptions>
```

Figure 4: An example CommandDescriptions.xml file

This command description represents an example of executing an Ant[30] build file on a submission. After this command description is read the following command is created:

[30] Apache Ant – Welcome. April 27 2011. <http://ant.apache.org/>

```
Runtime : 60s
        Working Directory : /path/to/the/submission/directory
        output file name : Command1.txt
        command :      ant.bat -buildfile /path/to/the/assignment/directory/adderTests/build.xml
                       -Dstudent.dir=/path/to/the/submission/directory
                       -Dassignment.dir=/path/to/the/assignment/directory
compile
```

Figure 5: The command created by the example CommandDescriptions.xml file in Figure 4

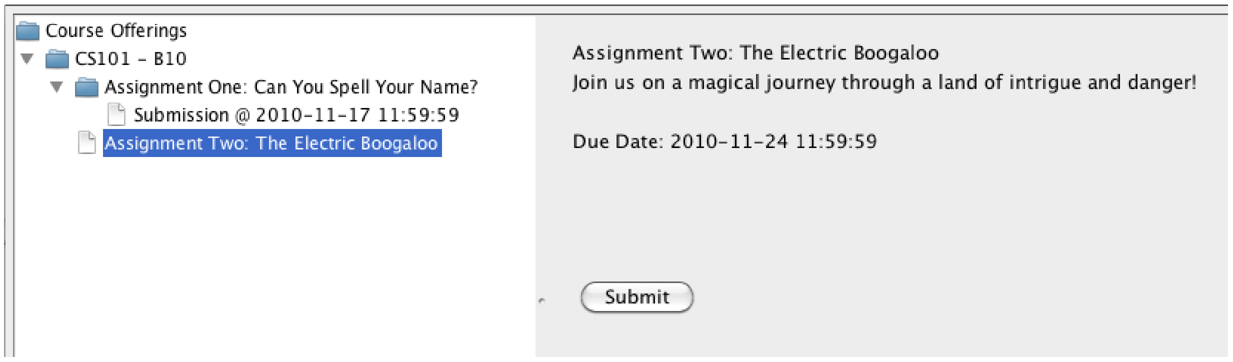
When the SPN receives this command it will run Ant for maximum of 60 seconds and generate the following files err_Command1.txt and Command1.txt along with any side effects of the Ant command (such as compiled files). By adding more <Command> nodes to this xml file allows the professor to chain a series of commands together that are executed in order to process the student's submission.

3.5 Graphical User Interface

The appearance of and the code behind the GUI of the Assignment Submission Module went through many iterations. However, certain aspects of the GUI have always been consistent. Professors have always seen Courses as the top level of their interfaces, intended to allow them to modify courses, assignments, and their offerings, in addition to enabling them to view and respond to student submissions. Meanwhile, students and teaching assistants (TAs) see Course Offerings as the base of their interface, which provides them to access to view course offerings and assignment offerings, and to respectively make or view and grade submissions. Also, because it is possible for someone to be any combination of professor, TA, and student across multiple courses or course offerings, it is possible to have one of each (Courses and Course Offerings) at the top level of the user interface.

The original GUI consisted of a panel which was split in half vertically. On the left there was a JTree[31] which displayed either courses or course offerings at the top level (or contained two top level nodes if necessary) as previously specified. On the right was the information for the currently selected node of the JTree. An Edit button allowed professors to change information, such as due dates, course professors, course offering members, and assignment descriptions.

A screenshot from an early revision of the GUI (student perspective):


Course Offerings
CS101 - B10
Assignment One: Can You Spell Your Name?
Submission @ 2010-11-17 11:59:59
Assignment Two: The Electric Boogaloo
Submit

Assignment Two: The Electric Boogaloo
Join us on a magical journey through a land of intrigue and danger!

Due Date: 2010-11-24 11:59:59

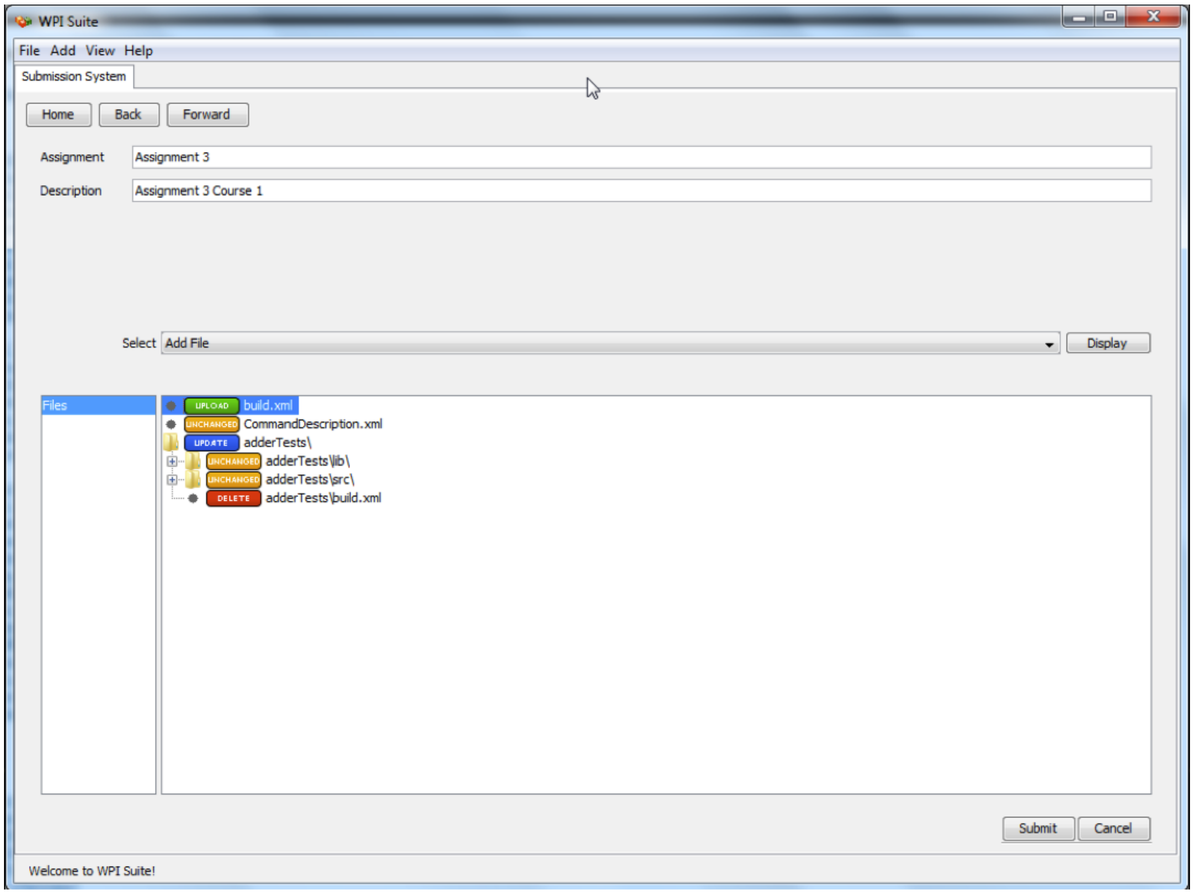
Figure 6: An early candidate for the design of the ASM GUI

In this original implementation of the GUI, the JTree nodes, along with their information, were supposed to periodically update to inform the users of the current status of the system. Unfortunately, there were problems with this realization of the system. The largest problem was that when the program would attempt to update in real time, the JTree nodes would frequently either be duplicated or missing. This was due to the odd way that the JTree handles removing nodes, adding nodes, and checking for equality when nodes are updated.

The duplicate nodes weren't the only problems with the original GUI. From a code perspective, the data structures for each type of element were overly complicated, and while they were simplified somewhat through successive revisions, they were still large and unwieldy. The application itself would perform slowly when the threads intended to check for updates were

[31] JTree (Java Platform SE 6). April 27 2011. <http://download.oracle.com/javase/6/docs/api/javax/swing/JTree.html>

running in the background. Even when this was negated somewhat, the first click of the JTree would frequently have a delayed response of several seconds where it should have been unnoticeable. In addition to the issues with poor performance and unwanted behavior, the design itself had some difficulties. The process of adding elements to lists, such as lists of professors or assignments, was cluttered. They would always appear on the right side of the split panel, but sometimes, depending on the element, they would also appear in the JTree to the left. Because each node could only have one set of actions, and that size of that set was constricted by the physical size of the right panel, there was a limited number of actions per node. Also, there was no easy way to manage file uploads; only single files or folders could be uploaded, and the effects of the uploads were ambiguous.



Assignment: Assignment 3
Description: Assignment 3 Course 1
Select: Add File

UPLOAD build.xml
UNCHANGED CommandDescription.xml
UPDATE adderTests\
UNCHANGED adderTests\lib\
UNCHANGED adderTests\src\
DELETE adderTests\build.xml

Welcome to WPI Suite!

Figure 7: The final GUI design

As a result, we decided to retire the vast majority of the GUI code and re-create it from scratch, as shown in figure 7. Learning from our mistakes on the previous revisions, we took a different approach to our new interface. Although we retained the concepts of using Course and Course Offerings as base levels, instead of using a JTree, which created the previously mentioned redundancy, we now have a small left column and a large right column. The left column shows what level of depth the user is currently at (for instance at Course Offerings or File Submissions), while the right column provides the user with a list of selectable items. Above the list of selectable items is a drop-down with a set of actions, such as View or Add File. Users can then decide what action to perform on the current selection. Above that is information for the item currently under observation. At the top, there is a navigation system which allows users to go to the Home level, or back and forth through the history of recently used levels.

In addition, we created a file manager for transferring files and entire directories. The new file manager allows users to add, remove, or update multiple files or folders at once and prior to executing provides feedback on what will be done with these items. The feedback informs the user whether an element will be Uploaded, Updated, Deleted, or Unchanged using professional looking icons with subtle, but important gradients. Users can then specify if they want to change these actions. For example, if a user has added a folder to the file manager, and within their target directory, this folder already exists, it will update all of the existing files which exist both on the source and target directories, delete files which only exist on the target directory, and upload files which currently exist only on the source directory. However, users could change this behavior before uploading should they want to ignore changes to one file or keep files on the target directory that were not in the source directory.

The standalone WPI Suite client has issues when accessing data stored within the database as each time the user selects something that is stored on the database it must be fetched and then displayed. In order to prevent this on our system we implemented a local cache for objects stored on the server and have a separate thread that periodically updates them.

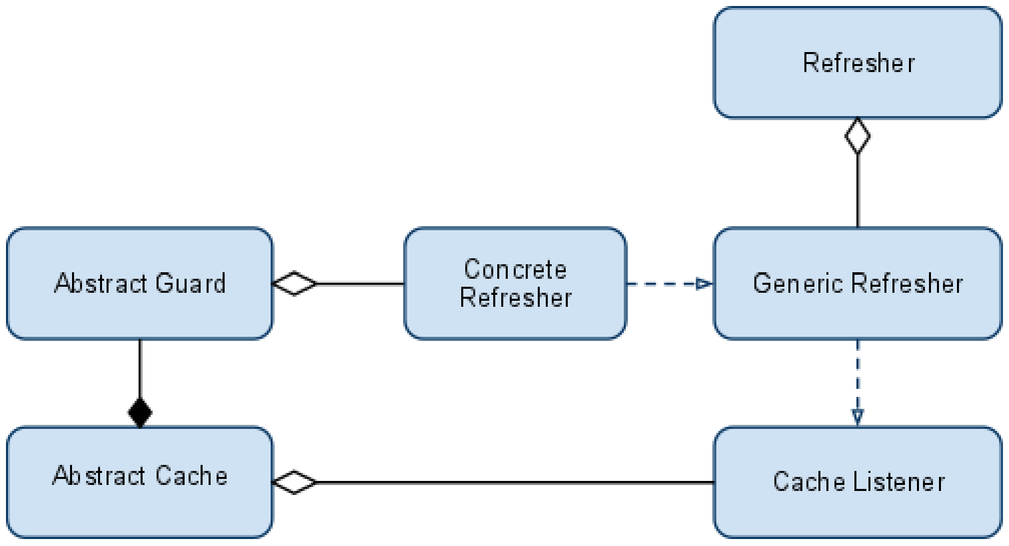


Figure 8: Diagram of the caching system

The updating system revolves around the Abstract Guard and Abstract Cache. The Abstract Guard has two responsibilities first it ensures that the object it is guarding is safely accessed during read and write operations and secondly provides the functionality for notifying other parts of the system that the underlying object was changed. The Abstract Cache ensures that each underlying object is represented only once within the client side system. On top of these two classes lies the Generic Refresher which is responsible for refreshing objects. The Refresher uses these Generic Refreshers in order to update the objects.

4. Results and Analysis

The project’s goal, building a module to support automated testing and grading of student submissions, was achieved. In addition, the team also constructed a scalable, modular, fault-tolerant server architecture, which should benefit future teams working on WPI Suite. The project also produced some fringe benefits for the WPI Suite Client, such as faster initial loading, a new configuration editor, and, in some cases, improved security. Additionally, we added dependency management to the entire WPI Suite project using Apache Ivy[32], a significant improvement over the manual dependency management system that was previously in use.

4.1 Metrics

	Packages	Classes	Lines of Code
Core	20	41	1512
Client	6	251	7654
Server	30	84	2943
Tests	3	8	363
Total	**59**	**374**	**12472**

Figure 9: Calculated metrics for code produced or affected by the project

[32] Home | Apache Ivy. April 27 2011. <http://ant.apache.org/ivy/>

5. Future Work and Conclusions

While the WPI Suite Assignment Submission Module is certainly a success, there is always more work to be done. For instance, the database persistence technologies on both the client and the server should be replaced. ActiveObjects, in particular, has a pressing need for a replacement. It was hardcoded into many existing parts of the WPI Suite Client without proper abstraction, which causes many problems. In addition, ActiveObjects has not released a new version in three years and makes it impossible to define utility methods on data model objects. Hibernate, meanwhile, is still actively developed and is very powerful, but it has a steep learning curve.

We recommend investigating DataNucleus[33] as a replacement for both. DataNucleus provides better performance in many cases, as it uses bytecode manipulation instead of reflection. It also provides an abstraction layer above the datastore, so that a wide variety of datastores can be used interchangeably. It also now includes a type-safe query mechanism that promotes easier code refactoring. Using only one data persistence technology would also reduce the learning curve for new developers getting started with WPI Suite.

Support for user groups, as suggested by Professor Kathryn Fisler, is something that should be implemented, and could be done with minimal changes to the system. User groups would be beneficial for handling group projects, allowing students to see each other's submissions, and making it so only one student in the group has to make a submission for an assignment.

More work on testing also needs to be done. Due to time constraints, we did not perform as much testing as we would have liked. Although the GUI seems to work well, usability studies

[33] DataNucleus – DataNucleus. April 27 2011. <http://www.datanucleus.org/>

should be performed, and the underlying structure has room for improvement. More rigorous testing of the server, such as how much concurrent access it is capable of handling, could be explored as well. Some of this could be accomplished by having students in a course, possibly Software Engineering, use the system to submit homework. Also, more logging and unit and integration tests need to be implemented for the system as a whole.

Glossary

API: Application Programming Interface

AMQP: Advanced Message Queuing Protocol. See http://www.amqp.org/ for additional information.

Apache Ant: An open-source Java tool that can build complex projects from script files. See http://ant.apache.org/ for additional information.

Apache Ivy: An open-source dependency management tool that is part of the Apache Ant project. See http://ant.apache.org/ivy/ for additional information.

Application scope: A type of scope that implies the implementation should be a singleton and is not a client export. Does not imply that the implementation is a persistence management class, although that is frequently the case. Implementations with this scope may only rely on other implementations that also have application scope.

ASM: Assignment Submission Module. Building this was the goal of the project.

Assignment: The part of the ASM data model that represents an assignment in a course. Includes an assignment name and description, and is associated with the files necessary to test submissions. May have multiple associated assignment offerings.

Assignment offering: The part of the ASM data model that represents an offering of an assignment in a course offering. “Inherits” all of the data from the assignment it is associated with and adds information specific to the particular offering, such as open and due dates.

Cajo: An open-source library designed to simplify use of the Java RMI system. See http://java.net/projects/cajo/pages/Home for additional information.

CCC: Computing and Communications Center. Part of WPI’s Information Technology Division. See http://www.wpi.edu/Academics/CCC/ for additional information.

CFN: Client Facing Node. This is the part of the WPI Suite Server that the client connects to.

Client Export: A class that is part of a CFN module that exposes functionality of some sort to client applications.

Command: A description of a process to be run that includes a maximum run-time, a file name to output store the standard output, and the program to run.

Command Executor: Responsible for setting up Command Timers and Command Outputs and then executing a Command.

Command Output: Responsible for storing the standard output and standard error streams in files.

Command Timer: Responsible for letting the Command Executor know when to stop executing a command.

Course: The part of the ASM data model that represents a course at a university. Includes a course name and description, and may have multiple associated course offerings.

Course offering: The part of the ASM data model that represents an offering of a course at a university. “Inherits” all data from the course it is associated with and adds information specific to the particular offering, including start and end dates as well as a list of users and their roles in the course offering.

CRUD: Create, Read, Update, and Delete. The four basic functions of persistent storage.

DataNucleus: An open-source data persistence solution for Java. Supports use of a wide variety of datastores to persist data and APIs to query them. See http://www.datanucleus.org/ for additional information.

Dirmi: An open-source Java library that provides a custom implementation of RMI aimed at fixing problems with the standard Java RMI system. See http://sourceforge.net/projects/dirmi/ for additional information.

DSO: See Terracotta DSO.

Eclipse: An open-source integrated development environment primarily used for building Java applications. See http://www.eclipse.org/ for additional information.

EPL: Eclipse Public License. See http://www.eclipse.org/legal/eplfaq.php for additional information.

GPL: The GNU General Public License. See http://www.gnu.org/licenses/gpl.html for additional information.

GUI: Graphical User Interface.

Hazelcast: An open-source clustering and data distribution platform for Java. See http://www.hazelcast.com/ for additional information.

HTTP: Hypertext Transfer Protocol is an application-level protocol for distributed, collaborative, hypermedia information systems. See http://tools.ietf.org/html/rfc2616 for addition information.

Implementation: A class provided by a CFN module that implements an interface and provides some functionality that other classes in the module or other modules are interested in utilizing. Instantiations of the implementation will be injected into other implementations that require it automagically. Must be associated with an appropriate scope.

JAR: Java Archive file. Aggregates many Java class files and other metadata into a single file, convenient for downloads and installations.

Java RMI: The RMI system that is included as a standard part of the Java language. See http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html for additional information.

JSPF: Java Simple Plugin Framework. Not to be confused with the Java Plugin Framework (JPF), which is a completely different project. See http://code.google.com/p/jspf/ for additional information.

JUnit: A Testing framework for the Java Programming Language. See http://www.junit.org for additional information.

JVM: Java Virtual Machine

LGPL: The GNU Lesser General Public License (previously known as the GNU Library General Public License). See http://www.gnu.org/licenses/lgpl.html for additional information.

MQP: Major Qualifying Project. A bachelor degree capstone project at WPI.

NAT: Network Address Translation, the process of modifying IP addresses and ports specified in headers as packets pass through a network router. Frequently used to connect multiple hosts to a larger network without needing to allocate multiple IP addresses. Connecting to hosts that are “behind” NAT is difficult and requires the use of special traversal systems.

OSGi: A widely-used framework specification for constructing modular software. Various commercial and open-source realizations of the specification exist. See http://www.osgi.org/ for additional information.

Persistence management class: Class that is part of the WPI Suite Server (usually as part of a CFN module) that manages interactions with the datastore and abstracts the details of the datastore.

PicoContainer: An open-source dependency injection framework for Java. See http://picocontainer.org/ for additional information.

REST: Representational State Transfer is a style of software architecture for distributed hypermedia systems such as the World Wide Web. See http://en.wikipedia.org/wiki/REST for addition information.

RMI: Remote Method Invocation. Allows a process running in one JVM to invoke a method on an object residing in another JVM. The JVMs may be on the same physical machine or on different physical machines.

Scope: This determines what other implementations a given implementation is allowed to rely on, as well as whether the implementation is a client export or not.

Session scope: A type of scope that implies the implementation is not a singleton and is a client export. Implementations with this scope may rely on other implementations with session scope and on implementations with application scope.

SIMON: An open-source Java library that provides a custom implementation of RMI aimed at fixing problems with the standard Java RMI solution. See http://dev.root1.de/wiki/2 for additional information.

Singleton: A class that will be instantiated at most once over the application’s life cycle.

SOAP: Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. See http://en.wikipedia.org/wiki/SOAP for additional information.

SourceForge: Refers to SourceForge Enterprise Edition (now called TeamForge), not SourceForge.net. SourceForge/TeamForge is an “integrated suite of web-based development and collaboration tools for Agile software development.” See http://www.open.collab.net/products/ctf/ for additional information.

SPN: Submission Processing Node. This is the part of the WPI Suite Server that runs the professor-specified commands on a submission. Early in the project, it was known as the TEN (Test Execution Node).

SQL: Structured Query Language

Submission: The part of the ASM data model that represents an attempt to satisfy an assignment offering. Submissions have files associated with them, and these files are tested according to the process defined by the professor in (and using the files associated with) the assignment that the assignment offering is an instance of.

TA: Teaching Assistant, a role users may have in course offerings. TAs have more privileges than students, but fewer privileges than professors.

Terracotta DSO: Terracotta Distributed Shared Objects, an open-source clustering framework for Java. Injects itself into code in an aspect-oriented manner at run-time to share designated objects across multiple JVMs. See http://www.terracotta.org/ for additional information.

Turnin: A homework submission system used by some Computer Science courses at WPI. Two separate versions exist: a web-based system (see http://www.cs.wpi.edu/~kfisler/turnin.html for additional information) and a command-line system on the CCC servers (see http://web.cs.wpi.edu/Help/turnin.html for more information).

Web-CAT: Web-based Center for Automated Testing. Specifically designed to automate unit testing and grading of student assignments. See http://web-cat.cs.vt.edu/ for additional information.

WEBFOOT: WPI Environment Built For Object-Oriented Teams. See https://sourceforge.wpi.edu/sf/projects/webfoot for additional information.

WPI: Worcester Polytechnic Institute. See http://www.wpi.edu/ for additional information.

ZMQ: a high-performance asynchronous messaging library aimed to use in scalable distributed or concurrent applications. See http://www.zeromq.org/ for additional information.

Bibliography

ActiveObjects: Wiki: Home. (n.d.). Retrieved April 20, 2011, from http://java.net/projects/activeobjects/pages/Home

Advanced Message Queuing Protocol. (n.d.). Retrieved April 27, 2011, from http://www.amqp.org/

Apache Ant - Welcome. (n.d.). Retrieved April 27, 2011, from http://ant.apache.org/

CollabNet TeamForge - The Agile ALM platform for distributed teams. (n.d.). Retrieved April 27, 2011, from http://www.open.collab.net/products/ctf/

DataNucleus - DataNucleus. (n.d.). Retrieved April 27, 2011, from http://www.datanucleus.org/

Dirmi - Bidirectional RMI. (n.d.). Retrieved April 27, 2011, from http://sourceforge.net/projects/dirmi/

Eclipse - The Eclipse Foundation open source community website. (n.d.). Retrieved April 27, 2011, from http://www.eclipse.org/

Eclipse Public License (EPL) Frequently Asked Questions. (n.d.). Retrieved April 27, 2011, from http://www.eclipse.org/legal/eplfaq.php

GNU Lesser General Public License v3.0. (n.d.). Retrieved April 27, 2011, from http://www.gnu.org/licenses/lgpl.html

Hibernate - JBoss Community. (n.d.). Retrieved April 20, 2011, from http://www.hibernate.org/

Home | Apache Ivy. (n.d.). Retrieved April 27, 2011, from http://ant.apache.org/ivy/

In-Memory Data Grid - Hazelcast - Home. (n.d.). Retrieved April 27, 2011, from http://www.hazelcast.com/

jspf - Java Simple Plugin Framework. (n.d.). Retrieved April 27, 2011, from http://code.google.com/p/jspf/

JTree (Java Platform SE 6). (n.d.). Retrieved April 27, 2011, from http://download.oracle.com/javase/6/docs/api/javax/swing/JTree.html

Less is More - zeromq. (n.d.). Retrieved April 27, 2011, from http://www.zeromq.org/

MySQL The World's Most Popular open-source database. (n.d.). Retrieved April 20, 2011, from http://www.mysql.com

OSGi Alliance | Main / OSGi Alliance. (n.d.). Retrieved April 27, 2011, from http://www.osgi.org

PicoContainer - Coding projects. (n.d.). Retrieved April 27, 2011, from http://picocontainer.org/

Remote Method Invocation Home. (n.d.). Retrieved April 27, 2011, from http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

Representational State Transfer. (n.d.). Retrieved April 27, 2011, from http://en.wikipedia.org/wiki/Representational_State_Transfer

RFC 2616 - Hypertext Transfer Protocol -- HTTP/1.1. (n.d.). Retrieved April 27, 2011, from http://tools.ietf.org/html/rfc2616

SIMON - Start - root1.de - Software Engineering. (n.d.). Retrieved April 27, 2011, from http://dev.root1.de/wiki/2

SimpleORM (Hibernate without the complexity). (n.d.). Retrieved April 27, 2011, from http://www.simpleorm.org/

SOAP. (n.d.). Retrieved April 27, 2011, from http://en.wikipedia.org/wiki/SOAP

TeamForge : Project Home. (n.d.). Retrieved April 27, 2011, from https://sourceforge.wpi.edu/sf/projects/wpisuite

TeamForge : Project Home. (n.d.). Retrieved April 27, 2011, from https://sourceforge.wpi.edu/sf/projects/webfoot

Terracotta. (n.d.). Retrieved April 27, 2011, from http://www.terracotta.org/

The cajo project: Wiki: Home - Java.net. (n.d.). Retrieved April 27, 2011, from http://java.net/projects/cajo/pages/Home

The GNU General Public License v3.0. (n.d.). Retrieved April 27, 2011, from http://www.gnu.org/licenses/gpl.html

Turnin Instructions. (n.d.). Retrieved April 27, 2011, from http://www.cs.wpi.edu/~kfisler/turnin.html

Using the 'turnin' Program. (n.d.). Retrieved April 27, 2011, from http://web.cs.wpi.edu/Help/turnin.html

Web-CAT Research Server Front Page. (n.d.). Retrieved April 20, 2011, from http://web-cat.cs.vt.edu/

Welcome to JUnit.org! | JUnit.org. (n.d.). Retrieved April 27, 2011, from http://www.junit.org/