

CSC408 Software Engineering Project

Part B

University of Toronto

Nov. 7th, 2004

Submitted by:

Jianlei Su – 992529652

Yuen-nung Chiao – 990155728

Yang Jia – 992110069

Kelvin Chan – 991461208

TABLE OF CONTENTS

TABLE OF CONTENTS	2
1. ABOUT OUR WEB SERVICE.....	4
1.1. OVERVIEW (INCL. URL)	4
1.2. WSDL FILE	4
1.3. DEPLOYMENT AND INSTALLATION	8
1.3.1. <i>Manually</i>	8
1.3.2. <i>Deploy and Install Automatically</i>	9
2. CUSTOMER – OMNIEDITOR USER’S GUIDE.....	10
2.1. INTRODUCTION.....	10
2.1.1. <i>The editors</i>	10
2.1.2. <i>Requirements</i>	10
2.1.3. <i>What is OmniEditor?</i>	11
2.1.4. <i>What's in this release?</i>	11
2.2. USING OMNIEDITOR WEB SERVICE	12
2.2.1. <i>Basics - Getting Started</i>	12
2.2.2. <i>Recommend Procedure</i>	13
2.3. PUBLISHED INTERFACES	13
3. UNIT TEST REPORTS.....	14
3.1. TEST ENVIRONMENT	14
3.2. FUNCTIONAL TEST REPORT.....	14
3.2.1. <i>Functional Test – JUnit</i>	14
3.2.2. <i>Overall test on register, logout, upload, download, find, and update in OmniEditingService class</i>	18
3.2.3. <i>Bug Reports</i>	20
3.2.4. <i>Integration Test Plan – Phase C</i>	21
3.3. NON-FUNCTIONAL TESTS.....	25
3.3.1. <i>Correctness and Reliability</i>	25
3.3.2. <i>Performance and Complexity</i>	27
4. MAINTENANCE PLAN.....	29
4.1. CUSTOMER COMMUNICATION	29
4.2. CORRECTIVE	29
4.2.1. <i>Bug Reporting</i>	29
4.2.2. <i>Bug Solving and Tracking</i>	29
4.3. ADAPTIVE	30
4.4. PERFECTIVE	30
4.5. PREVENTIVE	30
5. REQUIREMENT SPECIFICATION.....	31
5.1. FUNCTIONAL REQUIREMENTS:	31
5.1.1. <i>Upload</i>	31
5.1.2. <i>Download</i>	31
5.1.3. <i>Find</i>	31
5.1.4. <i>Register</i>	31
5.1.5. <i>Update</i>	32
5.1.6. <i>getFileNames</i>	32
5.1.7. <i>Logout</i>	32
5.2. NON-FUNCTIONAL REQUIREMENTS	32
5.3. FEASIBILITY	33
5.3.1. <i>Technical feasibility</i>	33
5.3.2. <i>Operational feasibility</i>	34
5.3.3. <i>Schedule feasibility</i>	35

5.4.	OTHER FEASIBILITY ANALYSIS	36
5.5.	ARCHITECTURE DESIGN	37
5.5.1.	Overview – Basic Functions.....	37
5.5.2.	Overview – Additional Functions.....	38
5.5.3.	High Level Use Case	39
5.5.4.	Sequence Diagrams (special cases).....	40
6.	PROJECT PLAN.....	46
6.1.	TASK ALLOCATION.....	46
6.2.	GANTT CHART	46
7.	TEAM ORGANIZATION.....	46
7.1.	TEAM MANAGEMENT.....	46
7.2.	SKILLS AND PREFERENCES	46
7.3.	TEAM MEETING SCHEDULE.....	46
7.4.	METHODS OF COMMUNICATION:	47
8.	RISK ANALYSIS.....	48
8.1.	PERSONNEL SHORTFALL	48
8.1.1.	Impact.....	48
8.1.2.	Prevention.....	48
8.2.	UNREALISTIC SCHEDULE/BUDGET	48
8.2.1.	Impact.....	48
8.2.2.	Prevention.....	48
8.3.	WRONG FUNCTIONALITY	49
8.3.1.	Impact.....	49
8.3.2.	Prevention.....	49
8.4.	GOLD-PLATING	49
8.4.1.	Impact.....	49
8.4.2.	Prevention.....	49
8.5.	REQUIREMENT VOLATILITY	49
8.5.1.	Impact.....	49
8.5.2.	Prevention.....	49
8.6.	BAD EXTERNAL TASKS	50
8.6.1.	Impact.....	50
8.6.2.	Prevention.....	50
8.7.	CAPABILITY SHORTFALLS	50
8.7.1.	Impact.....	50
8.7.2.	Prevention.....	50
9.	APPENDIX A: TASK ALLOCATION	51

1. About Our Web Service

1.1. Overview (Incl. URL)

Web Service URL:

<http://seawolf.cdf.toronto.edu:8086/service/OmniEditingService>

Project website (contains useful files and documentation)

<http://seawolf.cdf.toronto.edu:8086/doc/index.html>

Our team provides three web service functionalities: UPLOAD, DOWNLOAD and FIND. You can find the javadoc of the WS interface and the java code at the project website (see above).

1.2. WSDL File

```
=====
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
xmlns:intf="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tns1="http://omnieditor" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"><wsdl:types><schema
targetNamespace="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
xmlns="http://www.w3.org/2001/XMLSchema"><import
namespace="http://schemas.xmlsoap.org/soap/encoding/" /><complexType
name="ArrayOf_xsd_string"><complexContent><restriction
base="soapenc:Array"><attribute ref="soapenc:arrayType"
wsdl:arrayType="xsd:string[]" /></restriction></complexContent></schema></schem
a><schema targetNamespace="http://omnieditor"
xmlns="http://www.w3.org/2001/XMLSchema"><import
namespace="http://schemas.xmlsoap.org/soap/encoding/" /><complexType
name="OmniEditor"><sequence><element maxOccurs="unbounded" name="fileNames"
nillable="true"
type="impl:ArrayOf_xsd_string" /></sequence></complexType></schema></wsdl:types>
  <wsdl:message name="uploadRequest">
    <wsdl:part name="userId" type="xsd:int" />
    <wsdl:part name="fileName" type="xsd:string" />
    <wsdl:part name="fileContent" type="xsd:string" />
  </wsdl:message>
  <wsdl:message name="downloadRequest">
    <wsdl:part name="userId" type="xsd:int" />
    <wsdl:part name="fileNames" type="impl:ArrayOf_xsd_string" />
  </wsdl:message>
  <wsdl:message name="getFileNamesRequest">
    <wsdl:part name="userId" type="xsd:int" />
  </wsdl:message>
  <wsdl:message name="registerResponse">
    <wsdl:part name="registerReturn" type="xsd:int" />
  </wsdl:message>
```

```
<wsdl:message name="uploadResponse">
</wsdl:message>
<wsdl:message name="getFileNamesResponse">
  <wsdl:part name="getFileNamesReturn" type="impl:ArrayOf_xsd_string"/>
</wsdl:message>
<wsdl:message name="updateResponse">
  <wsdl:part name="updateReturn" type="impl:ArrayOf_xsd_string"/>
</wsdl:message>
<wsdl:message name="downloadResponse">
  <wsdl:part name="downloadReturn" type="impl:ArrayOf_xsd_string"/>
</wsdl:message>
<wsdl:message name="logoutRequest">
  <wsdl:part name="userId" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="findResponse">
  <wsdl:part name="findReturn" type="xsd:int"/>
</wsdl:message>
<wsdl:message name="findRequest">
  <wsdl:part name="userId" type="xsd:int"/>
  <wsdl:part name="text" type="xsd:string"/>
  <wsdl:part name="fileName" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="updateRequest">
  <wsdl:part name="userId" type="xsd:int"/>
  <wsdl:part name="fileName" type="xsd:string"/>
</wsdl:message>
<wsdl:message name="registerRequest">
</wsdl:message>
<wsdl:message name="logoutResponse">
</wsdl:message>
<wsdl:portType name="OmniEditingService">
  <wsdl:operation name="register">
    <wsdl:input message="impl:registerRequest" name="registerRequest"/>
    <wsdl:output message="impl:registerResponse" name="registerResponse"/>
  </wsdl:operation>
  <wsdl:operation name="find" parameterOrder="userId text fileName">
    <wsdl:input message="impl:findRequest" name="findRequest"/>
    <wsdl:output message="impl:findResponse" name="findResponse"/>
  </wsdl:operation>
  <wsdl:operation name="update" parameterOrder="userId fileName">
    <wsdl:input message="impl:updateRequest" name="updateRequest"/>
    <wsdl:output message="impl:updateResponse" name="updateResponse"/>
  </wsdl:operation>
  <wsdl:operation name="upload" parameterOrder="userId fileName fileContent">
    <wsdl:input message="impl:uploadRequest" name="uploadRequest"/>
    <wsdl:output message="impl:uploadResponse" name="uploadResponse"/>
  </wsdl:operation>
  <wsdl:operation name="download" parameterOrder="userId fileNames">
    <wsdl:input message="impl:downloadRequest" name="downloadRequest"/>
    <wsdl:output message="impl:downloadResponse" name="downloadResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getFileNames" parameterOrder="userId">
    <wsdl:input message="impl:getFileNamesRequest" name="getFileNamesRequest"/>
    <wsdl:output message="impl:getFileNamesResponse"
name="getFileNamesResponse"/>
  </wsdl:operation>
  <wsdl:operation name="logout" parameterOrder="userId">
```

```
<wsdl:input message="impl:logoutRequest" name="logoutRequest"/>
<wsdl:output message="impl:logoutResponse" name="logoutResponse"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="OmniEditingServiceSoapBinding"
type="impl:OmniEditingService">
  <wsdlsoap:binding style="rpc"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="register">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="registerRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://omnieditor.c408h003" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="registerResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="find">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="findRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://omnieditor.c408h003" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="findResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="update">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="updateRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://omnieditor.c408h003" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="updateResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="upload">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="uploadRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://omnieditor.c408h003" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="uploadResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="download">
```

```
<wsdlsoap:operation soapAction=""/>
<wsdl:input name="downloadRequest">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://omnieditor.c408h003" use="encoded"/>
</wsdl:input>
<wsdl:output name="downloadResponse">
  <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
use="encoded"/>
</wsdl:output>
</wsdl:operation>
<wsdl:operation name="getFileNames">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="getFileNamesRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://omnieditor.c408h003" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="getFileNamesResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
use="encoded"/>
  </wsdl:output>
</wsdl:operation>
<wsdl:operation name="logout">
  <wsdlsoap:operation soapAction=""/>
  <wsdl:input name="logoutRequest">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://omnieditor.c408h003" use="encoded"/>
  </wsdl:input>
  <wsdl:output name="logoutResponse">
    <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"
use="encoded"/>
  </wsdl:output>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="OmniEditingServiceService">
  <wsdl:port binding="impl:OmniEditingServiceSoapBinding"
name="OmniEditingService">
    <wsdlsoap:address
location="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```

=====End of WSDL File=====

1.3. Deployment and Installation

You can deploy and install the service manually or automatically.

1.3.1. Manually

=====

1. Package and compile your java code.

2. The package should be under:

Tomcat 5.0\webapps\axis\WEB-INF\classes\omnieditor

For example,

Tomcat 5.0\webapps\axis\WEB-INF\classes\omnieditor\OmniEditingService.class

In each class (eg, OmniEditingService.java), the package is: **package omnieditor**

3. Put all class file for the web service under:

Tomcat 5.0\webapps\axis\WEB-INF\classes\packageDir

For example:

Tomcat 5.0\webapps\axis\WEB-INF\classes\packageDir\OmniEditingService.class

4. Use the **java2wsdl** tool to generate OmniEditingService.wsdl.

5. Start Tomcat

6. Write (or generate) **deploy.wsdd** as follows:

=====

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
  <service name="OmniEditingService" provider="java:RPC">
    <parameter name="className" value="c408h003.omnieditor.OmniEditingService"/>
    <parameter name="allowedMethods" value="*" />
  <typeMapping
    xmlns:ns="http://seawolf.cdf.toronto.edu:8086/services/OmniEditingService "
    qname="ns:ArrayOf_xsd_string"
    type="java:java.lang.String[]"
    serializer="org.apache.axis.encoding.ser.ArraySerializerFactory"
    deserializer="org.apache.axis.encoding.ser.ArrayDeserializerFactory"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
  />
  <typeMapping
    xmlns:ns="http://omnieditor"
    qname="ns:OmniEditor"
    type="java:omnieditor.OmniEditor"
    serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
```



```
    deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"  
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"  
  />
```

```
</service>  
</deployment>
```

=====End of WSDD File=====

7. Deploy the web service with the WSDD file at a command prompt in the "omnieditor" directory:

```
<Your root>Tomcat 5.0\webapps\axis\WEB-INF\classes\omnieditor>  
java org.apache.axis.client.AdminClient deploy.wsdd
```

1.3.2. Deploy and Install Automatically

1. Make sure \$AXISCLASSPATH has been set to be able to run:
org.apache.axis.client.AdminClient
2. tar -xvf the deploy_me.tar to your working folder
3. sh deploy.sh //to deploy
4. sh deploy.sh //to undeploy

2. Customer – OmniEditor User's Guide

We have provided a User's Guide for our customers. This file can also be obtained separately on the project website.

2.1. Introduction

2.1.1. *The editors*

Text editors are the best friends of programmers. There are, however, multiple choices of editors that may confuse a new programmer once he or she is used to one of them. It wastes the programmer's time to study a new set of key bindings as the time can be spent on more creative tasks.

For example, VIM (developed in C) and Emacs/XEmacs (developed in Lisp/C) were long seen as the best of breed in the programming world. They have different key bindings that lock the programmer in the pro/con camps. Yet another set of editors catches up with a huge set of new features, such as Eclipse and jEdit (developed in Java).

As software engineers, we want to bridge the gap between the different editors and help the programmer to use any text editor through familiar key bindings.

These open-source editors have been engineered for years. In addition, they are good quality editors that have been widely used as integrated development environments (IDE). Unlike commercial editors such as Microsoft Word, Notepad, Wordpad, UltraEdit etc., open-source editors can be studied and modified to satisfy end-user's needs.

In software engineering practice, a program with more than 75,000 lines of code (LOC) can be considered large. The following table lists some statistics of the above mentioned open-source editors. Note that XEmacs is a graphical user interface (GUI) variant of Emacs.

The project involves software reengineering, testing and maintenance. The spirit of team work is very important to the success of software engineering.

2.1.2. *Requirements*

The easiest way of sharing information between different editors is through a file interface. However, it is inconvenient because it serializes the development, in other words, one editor can load a file only after another editor has saved it. Another way is to share by Copy-Cut-Paste operations on a shared clipboard. Remote cooperation is limited because the clipboard is local. Therefore we would like to communicate the editing events across editors through an interoperable interface, such as web services.

The idea of sharing editing skills among different IDEs is not new, for example, a VIM editing session can be associated with the NetBeans IDE (the one that accompanied with JDK) or Visual Studio. However, it is preferred to have a general design that any IDE can communicate with each other through a common interface.

The objective of the project is to bridge different editors through web services. For example, a programmer familiar with VIM can edit a text file while at the same time the change can be reflected in Eclipse, on a remote machine. It is not required to reinvent the wheel, i.e., only need to use one legacy editor that exists for years.

The students are not recommended to spend lots of time to write a new text editor. Instead should understand the existing one, and extend it.

2.1.3. *What is OmniEditor?*

OmniEditor is a web service developed for the aforementioned situation. It is implemented using JAX-RPC technology.

After weeks of continued discussion and coding effort in this direction, OmniEditor now delivers the following key features:

- **Flexibility.** The web service architecture gives the developer freedom to easily implement their editor for custom text editing.
- **Stability.** OmniEditor defines a set of **published interfaces** which change relatively slowly compared to the rest of OmniEditor.
- **Component-oriented deployment.** You can easily define reusable networks of Handlers to implement common patterns of processing for your editors, or to distribute to partners.
- **Common Editing framework.** We have a clean and simple abstraction for editing functions (i.e., UPLOAD, DOWNLOAD, FIND, MOVE, etc), and the core of the editor is completely platform and transport-independent.

We hope you enjoy using OmniEditor. Please note that this is an open-source effort - if you feel the code could use some new features or fixes, please get involved and lend a hand! The OmniEditor developer community welcomes your participation.

2.1.4. *What's in this release?*

This release includes the following features:

- **UPLOAD:** inform the OmniEditor to open a channel with the editing state, such as the content of the local text buffer and the position of the cursor in the buffer;
- **DOWNLOAD:** query the OmniEditor about the editing state of potentially other editors, such as the content of a certain channel and the position of the cursor;
- **FIND:** search a string and move the current to the found occurrence, if it does not exist, then do not move the cursor and return false;
- **REGISTER USER:** register to use the OmniEditor service; this will start a editing session within the server;
- **UPDATE:** get the difference between the local file buffer and the OmniEditor file buffer, then, the local editor can do what appropriate to the local file;
- **GET FILES:** retrieve the names of all files currently available in the OmniEditor;
- **LOGOUT:** end an editing session with the OmniEditor.

2.2. Using OmniEditor Web Service

2.2.1. Basics - Getting Started

Let's take a look at an example OmniEditor client that will call the upload method on the public OmniEditor server at cdf.

```
1.  package c408h003.omnieditor;

2.  import javax.xml.namespace.QName;
3.  import org.apache.axis.client.Call;
4.  import org.apache.axis.client.Service;
5.  import org.apache.axis.utils.Options;

6.  public class TestOmniEditorService_Client_download {
7.      public static void main(String [] args)
8.      {
9.          try {
10.             Options options = new Options(args);
11.             String endpointURL = options.getURL();
12.             String textToSend;

13.             args = options.getRemainingArgs();
14.             Service service = new Service();
15.             Call cal = (Call) service.createCall();

16.             call.setTargetEndpointAddress( new java.net.URL(endpointURL) );
17.             System.out.println(endpointURL);

18.             call.setOperationName( new QName("OmniEditingService", "download") );

19.             System.out.println( ((String[])call.invoke(new Object[] {new Integer("1"),
20.                 new String[] { "testFile" } }))[0] );

21.         } catch (Exception e) {
22.             System.err.println(e.toString());
23.         }
24.     }
25. }
```

So what's happening here? On lines 14 and 15 we create new Service and Call objects. These are the standard JAX-RPC objects that are used to store metadata about the service to invoke. On line 16, we set up our endpoint URL - this is the destination for our SOAP message. On line 18 we define the operation (method) name of the Web Service. And on line 19 we actually invoke the desired service, passing in an array of parameters - in this case just one String array.

2.2.2. *Recommend Procedure*

To use the OmniEditor web service, we recommend the following procedure, though it's not mandatory.

1. Use `registerUser()` to get a user identification number.
2. Upload a new file onto the OmniEditor; or, use `getFiles()` to retrieve all files (file names) currently available on the OmniEditor, then use `download()` to get the contents of the files want to edit.
3. Then, do all editing operations (for now, only find is provided, ☺, others are coming soon.)
4. Use `logout()` to end the current editing session.

2.3. Published interfaces

- `java.lang.String[] download(int userId, java.lang.String[] filenames);`
- `int find(int userId, java.lang.String text, java.lang.String filenames);`
- `java.lang.String[] getFileNames(int userId);`
- `void logout(int userId);`
- `int register();`
- `void upload(int userId, java.lang.String filename, java.lang.String fileContent);`
- `java.lang.String[] update(int userId);`

3. Unit Test Reports

3.1. Test Environment

Operating System	Linux 2.4.27 -i686
Editor	Eclipse 3.0 for Linux (x86/Motif)
Soap Engine	Apache Axis 1.1
Web Application Server	Apache Tomcat 5.5

3.2. Functional Test Report

3.2.1. Functional Test – JUnit

All the unit test cases were done using JUNIT. Each method of every class has gone through an extensive testing phase. The programmer of the codes had done testing while programming but to make the testing phase more robust we had another team member design and test the system without knowledge of how the program with coded. The testers were instructed to try and “break” the system as much as possible and in as many ways as possible. Although at first there were many bugs found in the system, we believe it is better to find them now then later. This method of testing ensures that our web-service is robust and bugs would be found. The testers have no reason not to find bugs because if any bugs were found the programmer must fix it and tester does not have to do more work. On the other hand because the programmer has to fix all the bugs, the programmer would want to make it as robust as possible such that less work would be needed to be done. Thus everyone is working hard to find bugs and solve them. In this environment or strategy of testing it should be the most efficient and stable.

The testing strategy that was used for testing the different methods was using all the different possible types of arguments that each method would take and comparing the results. The test cases have been written a head of time such that while comparing the outputs there is no chance of errors.

The following are test cases for each of the methods in every class. For more information please refer to the JUNIT testing code.

Note: All following expected results are satisfied by our present OmniEditor version

3.2.1.1. FindAction Class

- FindAction(int, String, Boolean, Boolean)
 - positive, negative and zero user id
 - search empty string, single words and phrases
 - all combinations of true/false for searching forward/backward and being case-sensitive.

- toString()
 - compared the output all the above tested cases, which is basically all possible outputs.

3.2.1.2. User

- User(int)
 - user with positive, negative and zero user Ids.
- addFile(String)
 - add files with empty file names, single word file names, and multi word file names.
- removeFile(String)
 - removed files with empty file names, single word file names, and multi word file names.
 - remove files that exist and don't exist
- getId()
 - get Ids with positive, negative and zero user Ids.
- getFiles()
 - get files for a user that has no files.
 - get files for a user that has only one file.
 - get files for a user that has multiple files.
- getVersionNumber(String)
 - retrieved a version number for a file.

3.2.1.3. UserFile Class

- UserFile(String, int)
 - version number negative, empty file name
 - version number positive, non-empty name
 - version number was zero, multi word file name
- Version
 - retrieved the variable while it's negative, positive and zero.
- Name
 - retrieved the variable while it's empty and not empty.

3.2.1.4. File Class

- File(String, String)
 - add a file with no file name, no content.
 - add a file with a file name with no content.
 - add a file with no file name with content.
 - add a file with a file name and file content.
- addUser()
 - adding a user to a file with no users.
 - adding a user to a file with one or more user

- removeUser()
 - removing user from a file with more than one user
 - removing user from a file with one user
 - removing a user from a file with no user
- find(int, String, Boolean, Boolean)
 - positive, and zero start positive
 - search empty string, single characters, single words and phrases
 - all combinations of true/false for searching forward/backward and being case-sensitive.
- getContent()
 - get the file content of a file that is empty
 - get the file content of a file that has one character
 - get the file content of a file that has one word
 - get the file content of a file that has one sentence
 - get the file contents of a file that is extremely large
- getName()
 - get file name that is a empty string
 - get a file name with one word
 - get a file name with multiple words
 - get a file name with an extension
- getVersionNumber()
 - get a file that is at version zero
 - get a file that is at positive version
- getAction()
 - retrieve a the action of a file with no actions performed
 - retrieve a the action of a file with one action performed
 - retrieve a the action of a file with many action performed
- addAction()
 - a find action
 - adding find actions multiple times
- getUserCount()
 - no users after the file is initialized
 - file with one user
 - file with twenty user

3.2.1.5. OmniEditingService Class

- OmniEditingService()
 - Build OmniEditor object with this constructor 1 time
Expected: an OmniEditor object is created
 - Build OmniEditor object with this constructor 10 times
Expected: only one OmniEditor object is created as a singleton
- register()
 - Register a user
Expected: a non-negative integer as userId is returned and the user are created in user list

- Register 10 users (even number of users)
Expected: 10 non-negative integers as userId are returned and the 10 different users are created in user list
 - Register 7 users (ode number of users)
Expected: 7 non-negative integers as userId are returned and the 7 different users are created in user list
 - More test cases in Overall test
- logout (int)
 - Logout an unexisting userId before the user register
Expected: No exception occurs
 - Logout an existing user with specific userId
Expected: The user is removed from user list.
 - Logout all existing users (even num: 10) with specific userIds
Expected: The users are removed from user list.
 - Logout all existing users (ode num: 7) with specific userIds
Expected: The users are removed from user list.
 - More test cases in Overall test
- upload(int,String,String)
 - Upload an empty file with empty file name (“”)
Expected: Nothing gets created in the buffer of OmniEditorBuffer
 - Upload an empty file with nonempty file name
Expected: an empty buffer is created in OmniEditorBuffer with the specified file name
 - Upload an nonempty file with nonempty file name
Expected: a buffer with exactly the same content as original file is created in OmniEditorBuffer with the specified file name
 - Upload 10 files with large size (1 MB) each.
Expected: All files are stored correctly in OmniEditorBuffer with specified file names.
 - More test cases in Overall test
- download(int,String[])
 - Download an unexisting file.
Expected: null is returned
 - Download an existing file.
Expected: the file content is returned as the 1st element in returned String Array
 - Download 10 existing files.
Expected: the file content is returned as the first 10 elements in returned String Array
 - Download 7 existing files.
Expected: the file content is returned as the first 7 elements in returned String Array
 - More test cases in Overall test

- update(int, String)
 - update when no new action has been done on the file
Expected: null is returned
 - More test cases in Overall test
- find (int, String , String)
 - find a string in an unexisting file
Expected: return -1
 - find a string in an existing file
Expected: return correct position of the string's 1st occurrence
 - More test cases in Overall test
- getFileNames (int, String , String)
 - do getFileName before any user upload any file
Expected: the returned String[] has length 0
 - upload a file and do getFileName
Expected: the returned String[] has length 1 with uploaded file name as 1st element.
 - register 2 users and each upload 1 file and then do getFileName
Expected: the returned String[] has length 2 with 2 uploaded file names as 1st and 2nd elements.

3.2.2. Overall test on register, logout, upload, download, find, and update in OmniEditingService class

- Register 10 users, and upload a file and download the file, and then logout all 10 users and download the file again
Expected: The last download of the file should return null since all users logout and the file should have been removed from OmniEditorBuffer
- Register 10 users, upload a file and download the file, and then logout 9 out 10 users currently in system, and download the file again
Expected: The last download of the file should return the contend of the file and shouldn't be null since user 10 is still using the file.
- Register 2 users, upload 7 files with one user, and logout the 2 users out. Change the content of the 7 files, and upload with 1 user again, and download the 7 files with another user.
Expected: The downloaded files should have new contend rather than old one
- With 2 users in the system, user 1 does a find action and use 2 does update
Expected: return a String[] with find information as first array element
- Register 5 users, and user 1 upload a file and user 2 find a string in the uploaded file
Expected: Correct postion of the first occurrence of the string is returned.

3.2.2.1. OmniEditorBuffer Class

- OmniEditorBuffer()
 - created an new Omni Editor Buffer
- registerUser()
 - registered 100 users
- removeUser(int)
 - remove the first user
 - remove the last user
 - remove all the users
- isUserOnline(int)
 - make sure registered users are online
 - check that removed users are no longer online
- isFileOnline(int)
 - check that added files are online
 - check removed files are no longer online
- addFile(int, String, String)
 - add a file with no file name, no content.
 - add a file with a file name with no content.
 - add a file with no file name with content.
 - add a file with a file name and file content.
 - added 100 files
- getFileContent(int, String[])
 - get the file content of a file that is empty
 - get the file content of a file that has one character
 - get the file content of a file that has one word
 - get the file content of a file that has one sentence
 - get the file contents of a file that is extremely large
- find(int, int, String, String, Boolean, boolean)
 - positive, and zero start positive
 - search empty string, single characters, single words and phrases
 - all combinations of true/false for searching forward/backward and being case-sensitive.
- getFileNames()
 - get file name that is a empty string
 - get a file name with one word
 - get a file name with multiple words
 - get a file name with an extension
- getChanges(int, String)
 - get file with no changes
 - get file with single change
 - get file with multiple changes
- removeFile(File)
 - remove first file
 - remove last file
 - remove a file that is not online

- remove all files
- removeFileByName(String)
 - remove first file
 - remove last file
 - remove a file that is not online
 - remove all files

3.2.3. Bug Reports

Date found	Date resolved	Type	Bug Description	Action	Status
File Class					
Nov 01	Nov 01	Failure	Unused argument in removeUser()	removed	resolved
OmniEditorBuffer Class					
Nov 04	Nov 04	Failure	isUserOnline: index out of bound	Incorrect variable used in the for-loop.	resolved
Nov 04	Nov04	Fault	getFileContent: only retrieves the last file content the rest is null	removed commented else statement	resolved
Nov 05	Nov 05	Fault	removeUser(): after the user removed, the user count of all files editing should be decremented and remove the file if the count becomes zero, but all user's get removed.	Incorrect return statement used in the for-loop, changed to break statement	resolved
Nov 05	Nov 05	Fault	registerUser: when add user, the userId returned could be duplicate	Changed the compare condition from 'equal to' to 'less than and equal to'	resolved
Nov 05	Nov 05	Failure	getchanges: couldnt retrieve changes	loop was incorrect. It was re-written.	resolved
Nov 06	Nov 06	Failure	Testing of user-logout failed	When a user connect to the web service and download an existing file, the getFileContent does not add the file to the user's list and increment the user count of the file. Add those into the method. Add a method to the User class for the convenience of the programming.	resolved
User Class					
Nov 04	Nov 04	Failure	getFile: null pointer exception	retrieved the UserFile object instead of the its	

				name	
OmniEditingService					
Nov 03	Nov 03	Failure	Upload: if upload a file twice, we will get ArrayIndexOutOfBoundsException	Fixed when isUserOnline above is fixed	resolved
Nov 04	Nov 04	Failure	Logout: before all user logout the file they have are removed	Fixed when getFiletoContent problem in OmniEditorBuffer is fixed	resolved

3.2.4. Integration Test Plan – Phase C

We will use Sandwich Integration testing

Our OmniEditor works like this: When a client request a web service, the request is sent to OmniEditor (wrapped by OmniEditingService object to make singleton of the OmniEditor object), which hands the request to an OmniEditorBuffer object that controls the private data members and objects residing in the OmniEditor and contains functions to manipulate these objects. Therefore OmniEditorBuffer is the middle level between the lower-level and top-level.

3.2.4.1. Top-Down:

Test the major interface to users in OmniEditor and OmniEditingServices, namely Upload, Download, Find, Register, Logout, Update and getFileNames. Create stubs for the functions in OmniEditorBuffer.java that correspond to each web service function we provide, namely:

```
- registerUser    // For Register
- removeUser     // For Logout
- removeFile     // For Logout
- isUserOnline   // For Upload/Download
- isFileOnline   // For Upload/Download
- addFile        // For Upload
- getFileContent // For Download
- find           // For Find
- getFileNames   // For GetFileNames
- getChanges     // For Update
```

This is to make sure the major control and decision-making is correct.

We deployed our websevice on one of our CDF machines and simulated editor-clients as follows:

- **test upload as remote client:**

The result of this test is to output a file with the uploaded file contest and the test succeeded.

```
public class TestOmniEditorService_Client {
```

```
public static void main(String [] args)
{
    try {
        Options options = new Options(args);

        String endpointURL = options.getURL();
        String textToSend;

        args = options.getRemainingArgs();

        Service service = new Service();
        Call call = (Call) service.createCall();

        call.setTargetEndpointAddress( new java.net.URL(endpointURL) );
        System.out.println(endpointURL);

        call.setOperationName( new QName("OmniEditingService", "upload") );
        call.addParameter( "arg1", XMLType.XSD_INT, ParameterMode.IN);
        call.addParameter( "arg2", XMLType.XSD_STRING, ParameterMode.IN);
        call.addParameter( "arg3", XMLType.XSD_STRING, ParameterMode.IN);
        call.setReturnType( XMLType.AXIS_VOID);

        call.invoke( new Object[] { new Integer("2"), "testFile","i'm this cool..." } );

    } catch (Exception e) {
        System.err.println(e.toString());
    }
}
```

- **Test download as remote client:**

The result of this test is to output the content of the file just uploaded, and it succeeded.

```
public class TestMultiUpload_download {
    public static void main(String [] args)
    {
        try {
            Options options = new Options(args);

            String endpointURL = options.getURL();
            String textToSend;

            args = options.getRemainingArgs();

            Service service = new Service();
            Call call = (Call) service.createCall();
```

```
call.setTargetEndpointAddress( new java.net.URL(endpointURL) );
System.out.println(endpointURL);

call.setOperationName( new QName("OmniEditingService", "upload") );
call.addParameter( "arg1", XMLType.XSD_INT, ParameterMode.IN);
call.addParameter( "arg2", XMLType.XSD_STRING, ParameterMode.IN);
call.addParameter( "arg3", XMLType.XSD_STRING, ParameterMode.IN);
call.setReturnType( XMLType.AXIS_VOID);

call.invoke( new Object[] { new Integer("1"), "testFile1","this is content1" } );
call.invoke( new Object[] { new Integer("1"), "testFile2","this is content2" } );
call.invoke( new Object[] { new Integer("1"), "testFile3","this is content3" } );
call.invoke( new Object[] { new Integer("1"), "testFile4","this is content4" } );
call.invoke( new Object[] { new Integer("1"), "testFile5","this is content5" } );
call.invoke( new Object[] { new Integer("1"), "testFile6","this is content6" } );
call.invoke( new Object[] { new Integer("1"), "testFile7","this is content7" } );
call.invoke( new Object[] { new Integer("1"), "testFile8","this is content8" } );
call.invoke( new Object[] { new Integer("1"), "testFile9","this is content9" } );
call.invoke( new Object[] { new Integer("1"), "testFile10","this is
content10" } );

System.out.println("finished uploading 10 files");

Service service1 = new Service();
Call call1 = (Call) service1.createCall();

call1.setTargetEndpointAddress( new java.net.URL(endpointURL) );
call1.setOperationName( new QName("OmniEditingService",
"download") );

String[] files = ( String[] )call1.invoke(new Object[] {new Integer("1"),
    new String[]{ "testFile1",
        "testFile2",
        "testFile3",
        "testFile4",
        "testFile5",
        "testFile6",
        "testFile7",
        "testFile8",
        "testFile9",
        "testFile10",}});
for (int i=0; i<files.length;i++){
    System.out.println(files[i]);
}

} catch (Exception e) {
    System.err.println(e.toString());
}
}
```

- **Test Find as Remote client:**

The result of this test is to output the position of the sting to be found, and it succeeded.

```
public class TestOmniEditorService_Client_find {
    public static void main(String [] args)
    {
        try {
            Options options = new Options(args);

            String endpointURL = options.getURL();
            String textToSend;
            args = options.getRemainingArgs();
            Service service = new Service();
            Call call = (Call) service.createCall();

            call.setTargetEndpointAddress( new java.net.URL(endpointURL) );
            System.out.println(endpointURL);

            call.setOperationName( new QName("OmniEditingService", "find") );
            call.addParameter( "arg1", XMLType.XSD_INT, ParameterMode.IN);
            call.addParameter( "arg2", XMLType.XSD_STRING, ParameterMode.IN);
            call.addParameter( "arg3", XMLType.XSD_STRING, ParameterMode.IN);
            call.setReturnType( XMLType.XSD_INT);
            Object tempInt = call.invoke(new Object[] {new Integer(1),"this",
"testFile"});
            System.out.println("tempInt:"+tempInt.toString());

        } catch (Exception e) {
            System.err.println(e.toString());
        }
    }
}
```

3.2.4.2. Bottom-up:

Low-level modules test steps:

1. *Action.java and FindAction.java:*

Action is an interface, which FindAction implements. It is just an action object that contains a constructor and a toString() function and can be easily tested.

2. *User.java & UserFile.java :*

These two classes are a smallest interdependent set, and can be tested independently from other files. UserFile.java contains only a constructor and two data members, filename and version. It contains the file name of the file that the User is editing and the user's local version for that file. User.java maintains a list of UserFiles and functions to manipulate the list.

By this time the stubs for register() and isUserOnline() in top-down testing can be removed.

3. *File.java*

The File object is a representation of an open file in the server that several clients are editing. Each File keeps track of a list of Users that are editing it, the number of the users editing it, a list of the Actions done on it, and other information such as cursor position and version number. It also contains functions to manipulate the User list and Action list, and a find() function that supports the higher level find()'s. File is tested here because the correctness of File is dependent on the correctness of all four classes mentioned above.

4. *OmniEditorBuffer.java – Meet with Top-Level*

Once File.java is tested, the low-level modules needed for the functions in OmniEditorBuffer are all tested and so the stubs for these functions can be removed.

3.3. Non-Functional Tests

3.3.1. Correctness and Reliability

We can see the evolution of the number failures (reliability) and faults (correctness) from the data obtained in our bug report (See 3.2.3).

Test Period: Nov 01 to Nov 07, 2004.

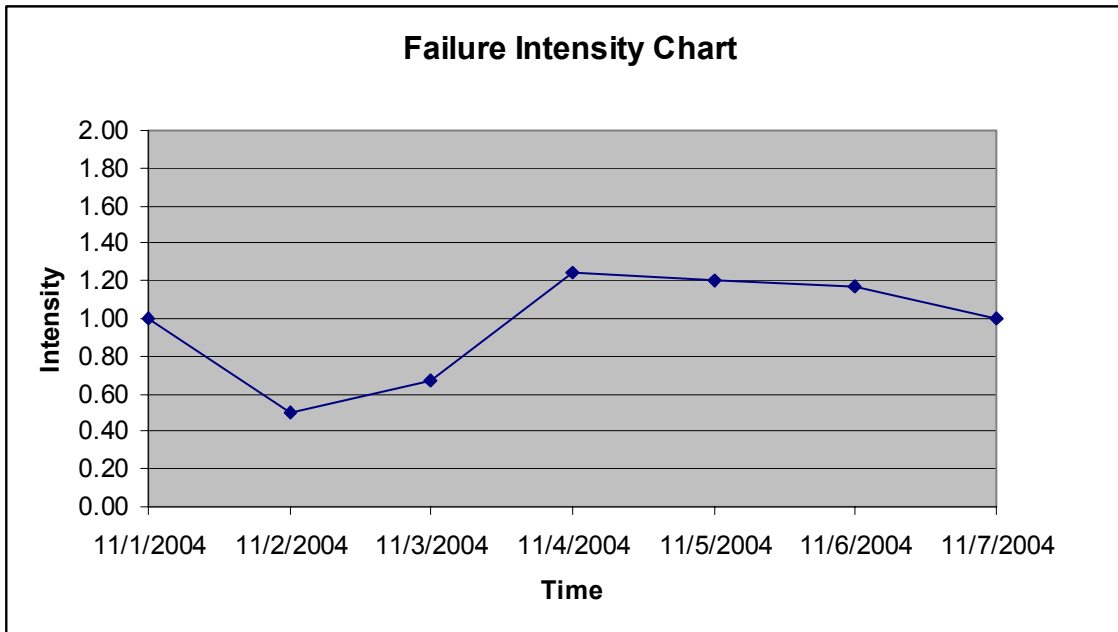
Bug Report Summary			
Date	Type	Location	Date resolved
11/01/04	failure		11/01/04
11/03/04	failure	OmniEditingService: upload()	11/03/04
11/04/04	failure	OmniEditingService: logout()	11/04/04
11/04/04	failure	User: getFile()	11/04/04
11/04/04	fault	OmniEditorBuffer: getFileContent()	11/04/04
11/04/04	failure	OmniEditorBuffer: isUserOnline()	11/04/04
11/05/05	fault	OmniEditorBuffer: removeUser()	11/05/05
11/05/05	fault	OmniEditorBuffer: registerUser()	11/05/05
11/05/05	failure	OmniEditorBuffer: getChanges()	11/05/05
11/06/06	failure	OmniEditorBuffer: getFileContent()	11/06/06

For failures:

Failure data summary			
Failure date	Cumulative Failures	Failure in interval	Failure Intensity
11/01/04		1	1
11/02/04		1	0
11/03/04		2	1
			1.00
			0.50
			0.67

11/04/04	5	3	1.25
11/05/04	6	1	1.20
11/06/04	7	1	1.17
11/07/04	7	0	1.00

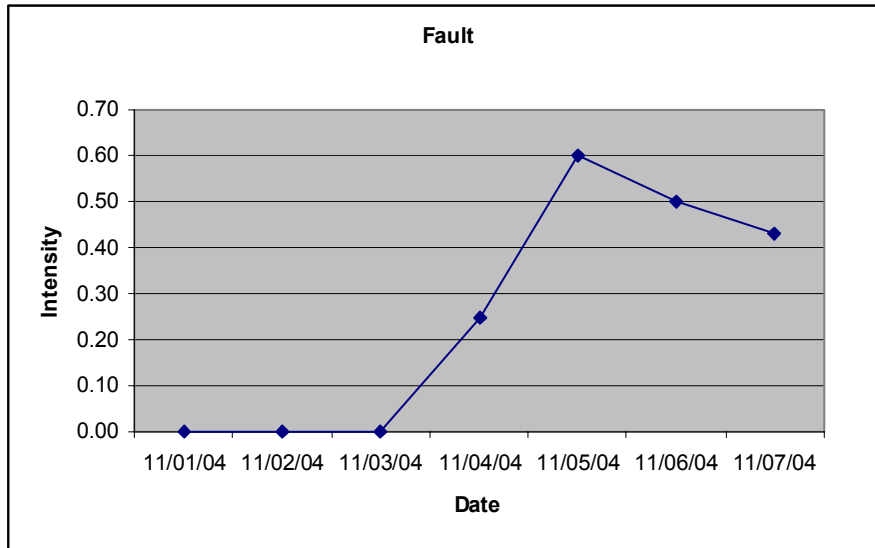
Failure intensity graph:



Fault data summary:

Fault data summary				
Fault date	Cumulative Faults	Faults in interval		Intensity
11/01/04	0	0	0	0.00
11/02/04	0	0	0	0.00
11/03/04	0	0	0	0.00
11/04/04	1	1	0.25	
11/05/04	3	2	0.60	
11/06/04	3	0	0.50	
11/07/04	3	0	0.43	

Chart:



We do not have a lot of data so far, but as we test the system more we will have more failure and fault data for a more detailed analysis.

3.3.2. Performance and Complexity

Performance is broken down into Time and Space performance and complexity is the major reason for low understandability, testability, maintainability. For the time performance, we setup the web service running at the server of gh-08.cdf.utoronto.ca (IP address: <http://128.100.31.98:8080/axis/services/OmniEditingService>). Then we run the test cases at home machine, the result is as follow.

```
d:\applications\Tomcat 5.0\webapps\axis\WEB-INF\classes>java
c408h003.omnieditor.TestOmniEditorService_Client -
lhttp://128.100.31.98:8080/axis/services/OmniEditingService
http://128.100.31.98:8080/axis/services/OmniEditingService
```

```
d:\applications\Tomcat 5.0\webapps\axis\WEB-INF\classes>java
c408h003.omnieditor.TestOmniEditorService_Client_download -
lhttp://128.100.31.98:8080/axis/services/OmniEditingService
http://128.100.31.98:8080/axis/services/OmniEditingService
i'm this cool...
```

```
d:\applications\Tomcat 5.0\webapps\axis\WEB-INF\classes>java
c408h003.omnieditor.TestOmniEditorService_Client_find -
lhttp://128.100.31.98:8080/axis/services/OmniEditingService
http://128.100.31.98:8080/axis/services/OmniEditingService
tempInt:4
```

```
real    0m7.900s
user    0m0.015s
sys     0m0.203s
```

Clearly, the running time is depend on the network bandwidth and the traffic on it. Below is the download testing program's time performance. For download a file, it takes 1.465 seconds. It's not bad for a network response.

```
$ time java c408h003.omnieditor.TestOmniEditorService_Client_download -lhttp://
128.100.31.98:8080/axis/services/OmniEditingService
http://128.100.31.98:8080/axis/services/OmniEditingService
i'm this cool...

real    0m1.465s
user    0m0.015s
sys     0m0.031s
```

The find and update take a little more (0.561s) than download. That should be caused by go over all file contents and comparing against the search target.

```
james_jia@james /cygdrive/d/applications/Tomcat 5.0/webapps/axis/WEB-INF/classes
$ time java c408h003.omnieditor.TestOmniEditorService_Client_find -lhttp://128.
100.31.98:8080/axis/services/OmniEditingService
http://128.100.31.98:8080/axis/services/OmniEditingService
tempInt:4

real    0m2.026s
user    0m0.030s
sys     0m0.141s

$ time java c408h003.omnieditor.TestUpdate_Client -
lhttp://localhost:8080/axis/services/OmniEditingService
http://localhost:8080/axis/services/OmniEditingService
null

real    0m2.153s
user    0m0.015s
sys     0m0.171s
```

Profiler instruments the code by book-keeping instructions. When the program runs, these instructions can be used to tell

- Which functions are called the most
- How is time distributed among different functions?
- From these data, one can pinpoint the bottleneck of the execution time

We tried to user java profile to run our web service, but it failed. The out put is following.

```
HPROF ERROR: thread local table NULL in method exit 009F8348
HPROF ERROR: thread local table NULL in method exit 009F8348
```

*HPROF ERROR: class ID already in use
Exception in thread "main" java.lang.NoClassDefFoundError: java
Dumping CPU usage in old prof format ... done.*

And for the complexity, we tried the maccab and halstead mentioned in the tutorial. But those two seem only work for c program. We got errors when running them against our *.java files. Since the program is not too complicated, we suppose that the complexity is not high.

4. Maintenance Plan

4.1. Customer Communication

We have created a yahoo newsgroup for communicating with our customers.

- Name of the yahoo.com newsgroup: **c408h003**
- URL: <http://groups.yahoo.com/group/c408h003>
- To post to the newsgroup by email, send to: c408h003@yahoogroups.com

We will also have a project website. The project website will contain:

- Documentation (including javadoc)
- File and tool downloads (e.g. wsdl2java)
- Bug Tracking Information
- General project updates, such as version changes.

4.2. Corrective

4.2.1. Bug Reporting

To report a bug, the customer can post/email to the newsgroup stating:

- Description of the problem
- The operation it appears in
- The command called, and any output or error messages that might be helpful

Upon receiving the bug report, our team member will give the customer a tracking number.

The customer can check the status of the bug on our project website.

The customer can keep constant communication with the group by emailing the newsgroup or posting to it.

4.2.2. Bug Solving and Tracking

- 24-hour guarantee: The bug will be solved in 24 hours, starting from when the group received the bug and the tracking number is assigned.
- Every week from November 08 to the due date of Phase C, two members are assigned the bug-checking and bug-solving task. The two members alternate each week with the other two. The first two members responsible will be Yang Jia (James) and Yuen-nung Chiao (Juno).
- Once the bug is solved, the customers will be notified. The customers can also check the project website for this information.

4.3. Adaptive

Since the project is done with Java and web service, the adaptability should be good. However if the customers made reasonable request on changing the system to fit their needs, they can send a Request for Feature report to the above newsgroup stating:

- The condition of the change request
- A description of what they want
- The preferred deadline for the feature

Our team will contact the customer to clarify the feature request if necessary, and a Feature tracking number will be assigned. Customers can check the status of the feature on the project website. Note that extra features usually have a lower priority than bugs so the 24-hour guarantee does not apply.

The feature requests will be handled by the two members not currently responsible for bug-solving in that week.

4.4. Perfective

Regression tests will be run after every change made to the system. As we test we will find some areas that we can improve, for example the performance.

If there are improvements added to the project (not from customer request), the customers will be notified with the version changes and the website will be updated with this information.

Perfective changes have not as high a priority as bugs. It will be taken care of by the two members not responsible for bug-solving in that week.

4.5. Preventive

With regression testing we might find some problems that have not yet manifested or caught by customers, if so the problem is treated as a bug, but has less priority than customer's bugs. These bugs are not reported to customers as they were found, but will be described on the project website if there are any version changes.

Since preventive changes are treated as bugs, they are handled by the two members responsible for the bugs that week.

5. Requirement Specification

The Javadoc for the functional requirements can also be found at c408h003's project website:
<http://seawolf.cdf.toronto.edu:8086/doc/i>

5.1. Functional Requirements:

5.1.1. *Upload*

Opens a channel with the editing state, such as the content of the local text buffer and the position of the cursor in the buffer

Inputs: `userId` - the user who want to upload a file

`fileName` - the name of the file

`fileContent` - the file contents of which the user want to share

Outputs: void

Precondition: The arguments are not null.

Postcondition: A file object with the editing states is created and saved to the buffer, editing states includes the content of the local text buffer and the position of the cursor. If the user is not registered yet, register the user.

5.1.2. *Download*

Sends the current editing state of the specified buffer to the user initiates this service.

Inputs: `userId` - the user who wants to download a file

`fileNames` - the file names of which the user want to get

Outputs: the current contents of each file in the buffer

Precondition: The files specified by filenames exist in the buffer

Postcondition: The contents of the files specified are sent to client

5.1.3. *Find*

Finds a string in the specified editing buffer

Inputs: `userId` - the user who want to search in the file

`text` - the string to be looked for in a file

`fileName` - the file name in which the user want to find the string

`forward` - true to search forward in the file, false otherwise.

`caseSensitive` - true for case-sensitive search, false otherwise

Outputs: the first occurrence position of the text in the file

Precondition: There is a working file buffer in the server.

Postcondition: the current occurrence be highlighted if there is at least one. Nothing if none exists.

5.1.4. *Register*

Gets the user id to start the web service.

Inputs: void

Outputs: An integer ID number assigned by the web service

Precondition: The user is not registered yet.

Postcondition: The user is registered with the web service.

5.1.5. Update

Gets the difference (changes made) between the user's version and the server's version.

Inputs: `userId` - the user's identification

`fileName` - the name of the file to be updated

Outputs: a string array stands for all actions perform on the file. Each string stands for one action.

Precondition: The version number of user's local file is different from that of the server's file.

Postcondition: The versions of the local and the server side is the same for that file.

5.1.6. *getFileNames*

Gets all files available on the web server.

Inputs: `userId` - the user's identification number

Outputs: a string array of all file names currently available.

Precondition: User must be registered

Postcondition: File list is not modified.

5.1.7. Logout

Logout the web service.

Inputs: `userId` - the user's identification number

Outputs: void

Precondition: The user is registered with the service.

Postcondition: The user is removed from web service.

5.2. Non-functional requirements

- Softgoal: Correctness
- Metric: the outputs of the functions
- Satisfaction criteria: the outputs of the functions are the same as expected

- Softgoal: Reliability
- Metric: replicability or the extent to which similar results can be reproduced over time using the same instrument
- Satisfaction criteria: the functionalities provided should be able to repeat over and over with the same results as long as the inputs are the same.

- Softgoal: Interoperability
- Metric: Data communication
- Satisfaction criteria: Reliable, timely data communication must occur between clients, as well as from client to host/access point and back.

- Softgoal: Extensibility

- Metric: Capability to add new editors to the system
- Satisfaction criteria: Provide an easy to learn and use interface for integrating new editors on to the current system

- Softgoal: Security
- Metric: Integrity of the buffer
- Satisfaction criteria: Through all the operations on the buffer, all clients should be able to operate on the same buffer at all the time.

- Softgoal: Usability
- Metric: The easiness to use the functionalities provided.
- Satisfaction criteria: The functionalities provide should be straightforward to use if the user has experience in Eclipse text editing. The learning time < 5 minutes.

- Softgoal: responsiveness [Operation]
- Metric: response time
- Satisfaction criteria: response time < 1 second

5.3. Feasibility

5.3.1. *Technical feasibility*

In this part, we will try to answer the following questions to show the design and planning of our project are technically feasible:

5.3.1.1. **Is building a web application for OmniEditor service a feasible choice?**

For the following reasons, our answer is yes:

Web application uses HTTP as communication protocol. It is more system and programming language independent than the protocols used by RPC calls.

Sometimes RPC represents a compatibility and security problem and firewalls and proxy servers will normally block this kind of traffic. However, HTTP is different, since it is supported by all web servers and browsers, and it can bypass the block of firewall.

Furthermore, we will use SOAP as the message format to request to and get response from web server. Since SOAP provides a way to communicate between applications running on different operating systems and it is supported by different technologies and programming languages, our service will provide good portability for clients on various systems.

5.3.1.2. Is our proposed solution to the shared editing service reliable and practical?

We will show our solution in greater detail in Architecture. However, the core of our solution is build on the idea that, in the shared editing service, each client can only edit / change the content of the document when he / she has the latest version of the document. We decide to stick to this rule because we believe that it is essential that no one can overwrite others' change he / she is not aware of, and changes can only be made when a client possess the latest information about the document being edited.

We borrowed this idea from the pattern that CVS used to solve current version systems problem. Since CVS is a mature technology and is believed to provide a reliable solution to a problem very similar to ours, as long as we make our shared editing service easy to use, our solution should be both reliable and practical.

5.3.1.3. Are all software components/tools we need to use available?

The key software components / tools we need to use to build our OmniEditor are listed below:

Tools	Description
Editor	Eclipse 3.0 for Linux (x86/Motif)
Java SDK	j2sdk1.4.2_05
Soap Engine	Apache Axis 1.1
Web Application Server	Apache Tomcat 5.5

All of these components are open source projects and their source codes are available online. We have already installed them on our CDF machines and they worked without problem.

In summary, as shown in the three aspects, we believe that our solution is technically feasible.

5.3.2. Operational feasibility

5.3.2.1. Is the Problem worth solving?

Suppose this is a product we are going to build in real world, this is a very important question to answer before we start.

Shared editing service can be very powerful if we provide enough functionality to multiple users such that everybody can share latest information. It is like a large whiteboard in a meeting room, and all participants shared their ideas and therefore can be more efficient and productive.

It is obvious that the basic functions we are going to implement for the shared service can not fulfill this purpose. However, we believe that it is still worth building as long as we provide enough scalability such that new functionalities can be easily added to this software. Therefore, our focus is not only to implement the basic functions but also to build a flexible skeleton that is easy to support new shared-editing functionalities.

5.3.3. *Schedule feasibility*

5.3.3.1. Can we meet the deadline to deliver our product?

In order to meet deadline of delivery of our product, we need to keep up with the following schedule:

5.3.3.2. Schedule for OOD, OOP

OOD and OOP	Deadline	Owner
OOD	Oct 10, 2004	All members
Publish the interface of all classes and public methods for team development	Oct 12, 2004	Jianlei Su
Implement Find function locally	Oct 14, 2004	Yang Jia
Implement Upload function	Oct 16, 2004	Jianlei Su
Implement Download function	Oct 18, 2004	Jianlei Su
Implement Find function	Oct 20, 2004	Jianlei Su
Unit Test Development	Oct 28, 2004	Yang Jia/Kelvin Chan
Functional/Non-functional Unit Testing	Nov 03, 2004	Yang Jia/Kevin Chan/Jianlei Su
Web Service Deployment Test	Nov 03, 2004	Yang Jia
Code Documentation	Nov 03, 2004	Yuen-nung Chiao
Project Phase B Write-up	Nov 07, 2004	Yuen-nung Chiao

5.3.3.3. Schedule for Phase C

OOD and OOP	Deadline	Owner
OOD	Nov 13, 2004	All members
Publish the interface of all classes and public methods for team development	Nov 15, 2004	Jianlei Su
Implement client-side (depend on which set of functions we choose)	Nov 23, 2004	Yang Jia / Jianlei Su / Kelvin Chan / Y.N. Chiao
Unit Test Development	Nov 25, 2004	Y.N. Chiao / Kelvin Chan

Unit Testing	Nov 30, 2004	Yuen-nung Chiao / Kelvin Chan
Code Documentation	Nov 30, 2004	By all members
Project Phase C Write-up	Dec 02, 2004	Y.N. Chiao

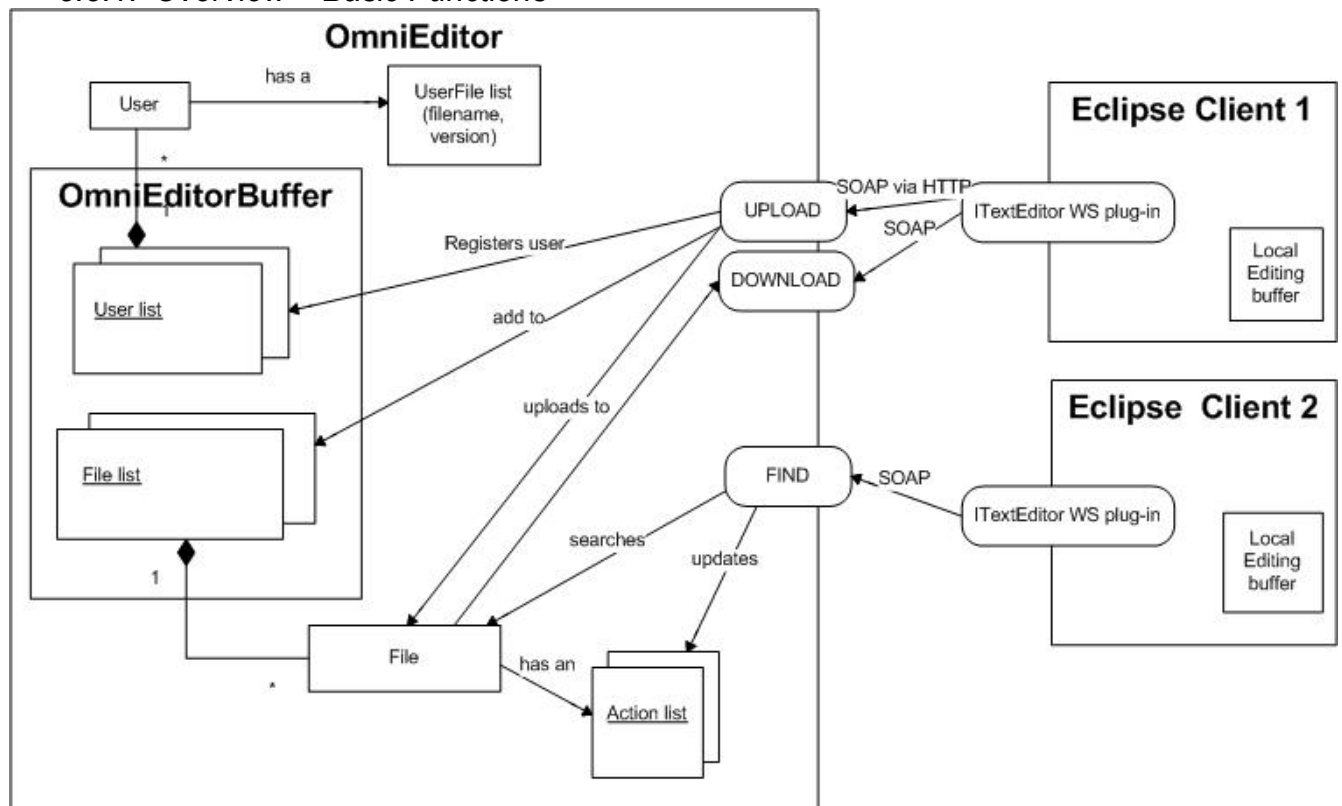
5.4. Other Feasibility Analysis

Other Feasibility analysis such as Economic feasibility analysis is not applicable to this course project, and therefore we will not discuss it here.

In summary, the discussion above shows that our product satisfies technical feasibility, operational feasibility. If we stick to our planed schedule, we can also achieve schedule feasibility for our project.

5.5. Architecture Design

5.5.1. Overview – Basic Functions



OmniEditorBuffer keeps a User list and File list for OmniEditor and provides functions to manipulate them. User list contains a list of Users currently registered to use the WS. File list contains a list of Files currently opened for editing; each File has its own current version number. The filenames must be unique.

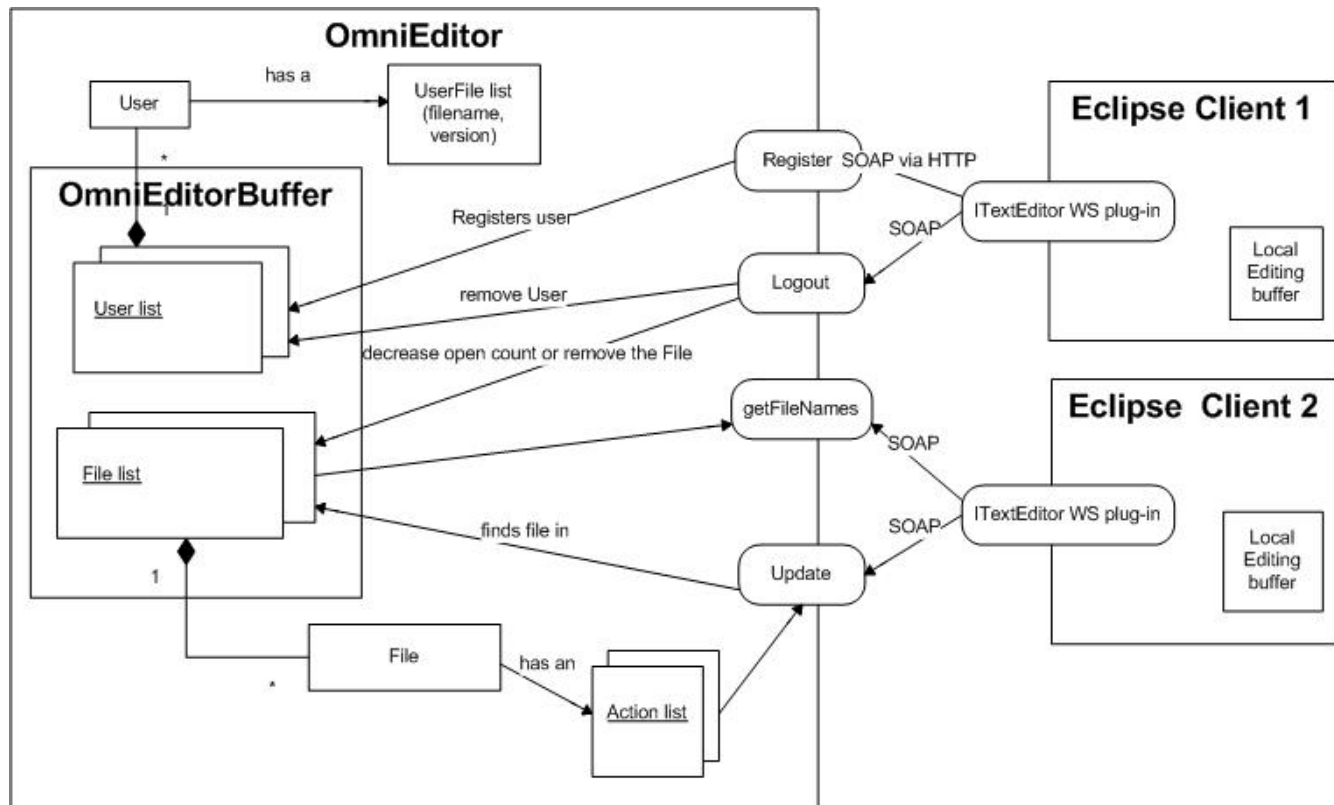
User keeps a list of UserFiles, which only contains the name of the files the User is editing (thus User can editing more than one File), and the User's version of the Files. User can only modify a File if his/her file version for that file is the same as the one kept in the File object.

Whenever Upload is called, the file buffer is loaded with the new content. If the calling client is not registered, Upload registers it and adds the new File to the File list.

When Download is called, the File content is sent to the client.

When Find is called, a FindAction is added to the list of Actions a File maintains to indicate an action of Find is done on the File. Find searches the File and returns the offset where the string is found.

5.5.2. Overview – Additional Functions



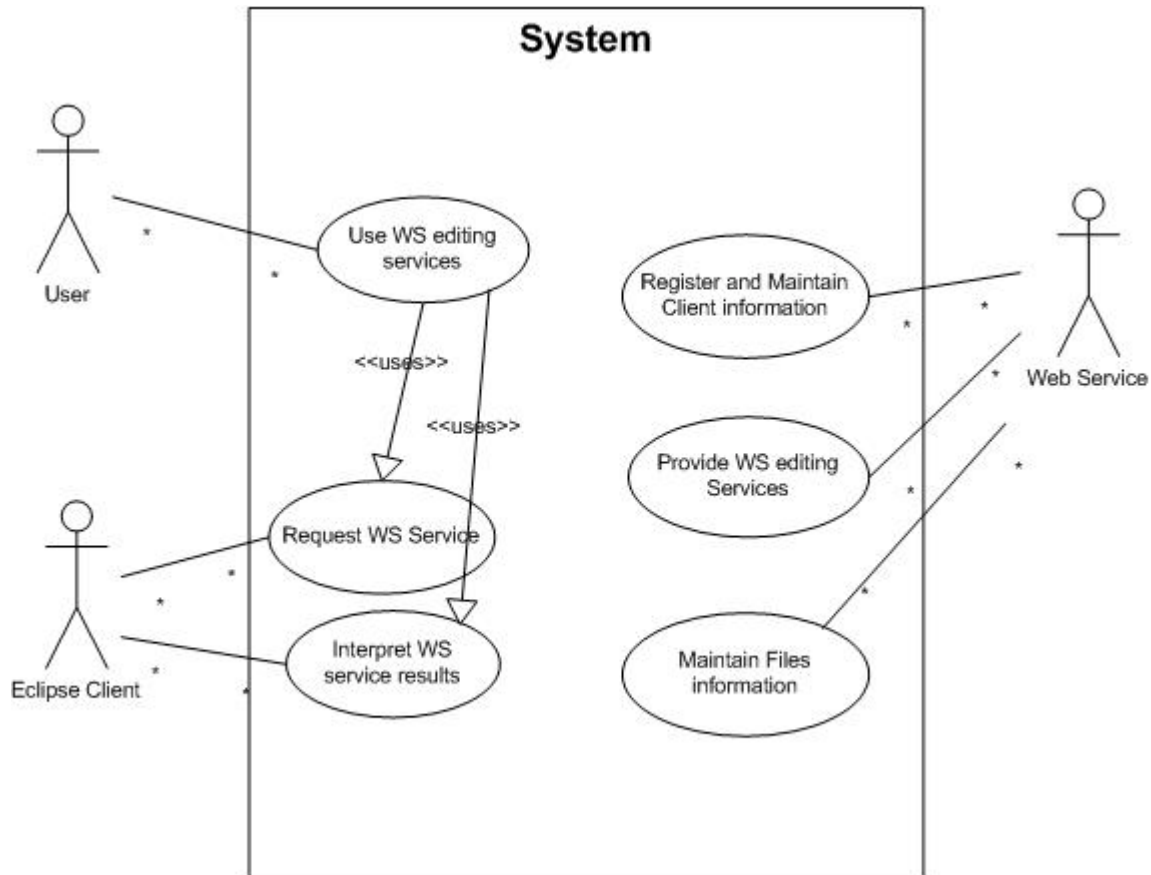
Register: Registers a User into the User list. A User ID is assigned and returned to the client by the WS.

Logout: Remove the User from the User list, and decrease the open count of the Files that this User was editing in the File list. If any File in the File list has an open count of zero, it means this File is no longer being edited by anyone and Logout removes it from the File list.

GetFileNames: Returns to the client a list of file names of the Files that are currently open in the server.

Update: Finds the specified File in the File list, and returns the list of Actions done on the File to the client, starting from the actions that the client has not yet downloaded (controlled by the File's version number and the client's own version number of that File.)

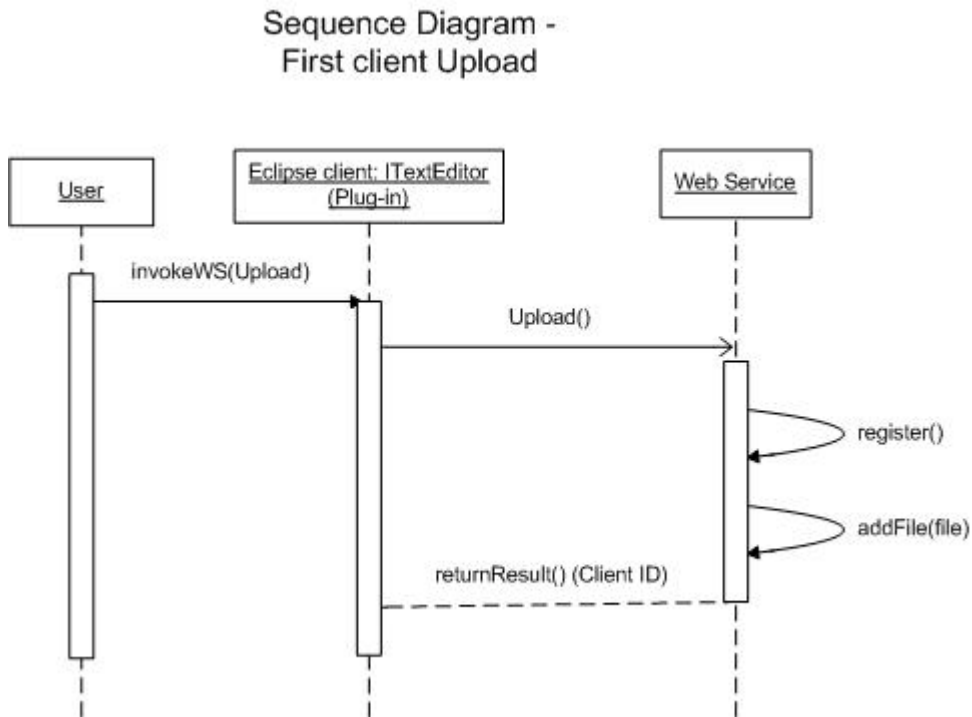
5.5.3. High Level Use Case



The user can invoke the editing services via the Eclipse client, which is in charge of requesting web services from the WS server and interpreting the returned results. The “WS editing services” includes UPLOAD/DOWNLOAD/FIND as specified in the OmniEditor project handout, and also helpful functions Register, Logout, getFileNames and Update. The WS Server maintains registered clients and their files, also a list of all open files in the server. It provides the WS editing services to the clients/users.

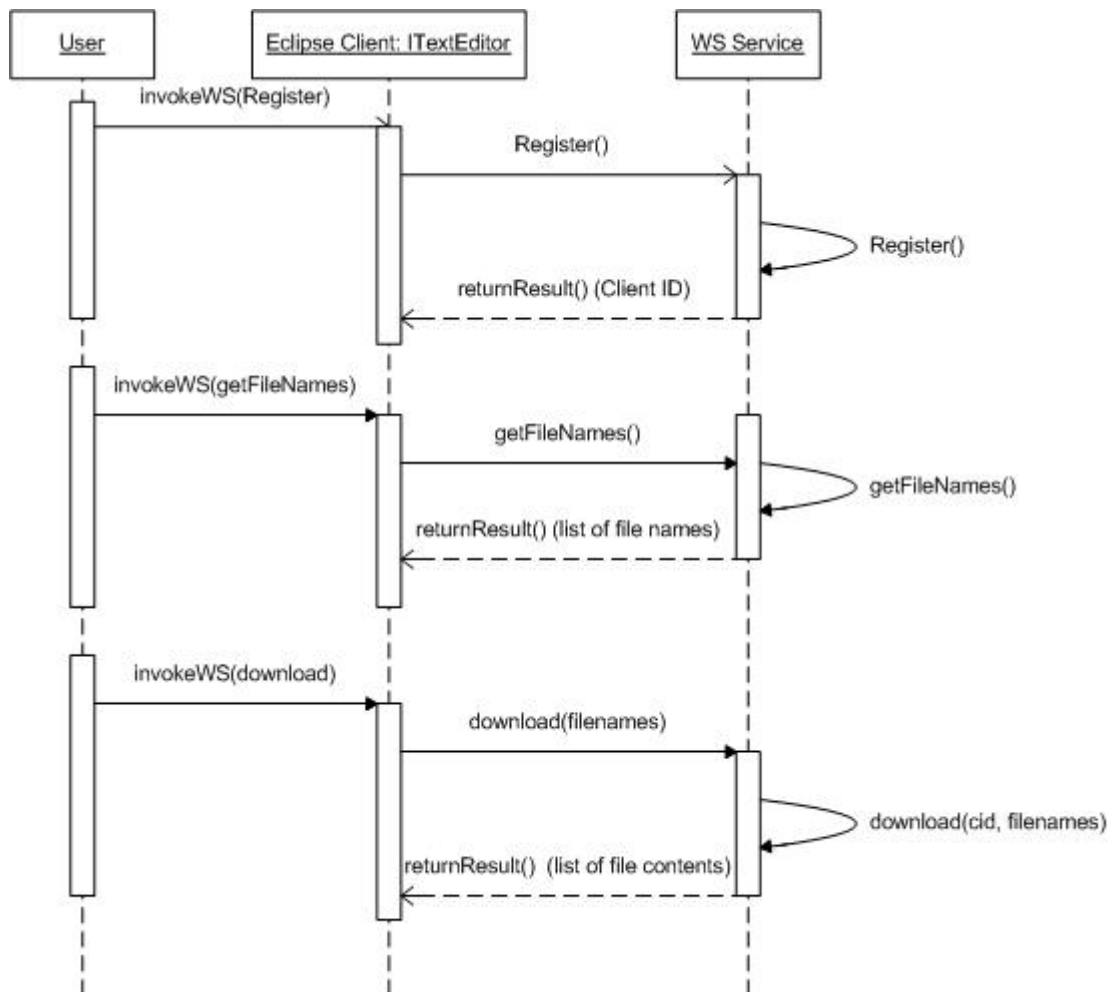
5.5.4. Sequence Diagrams (special cases)

5.5.4.1. First Client Upload (starting a new editing session)



When a user wants to start a cooperative editing session on a file, the user invokes the web service capabilities in the Eclipse client and indicates the file to upload. The web service server will register the client in its User list along with the file name it has uploaded. The server then creates and adds this new File to the server's list of open Files. It then returns the result to the client, which is the client ID that the WS assigned to this User. If the call fails, an error message is returned.

5.5.4.2. New Client joins an existing editing session (downloads)

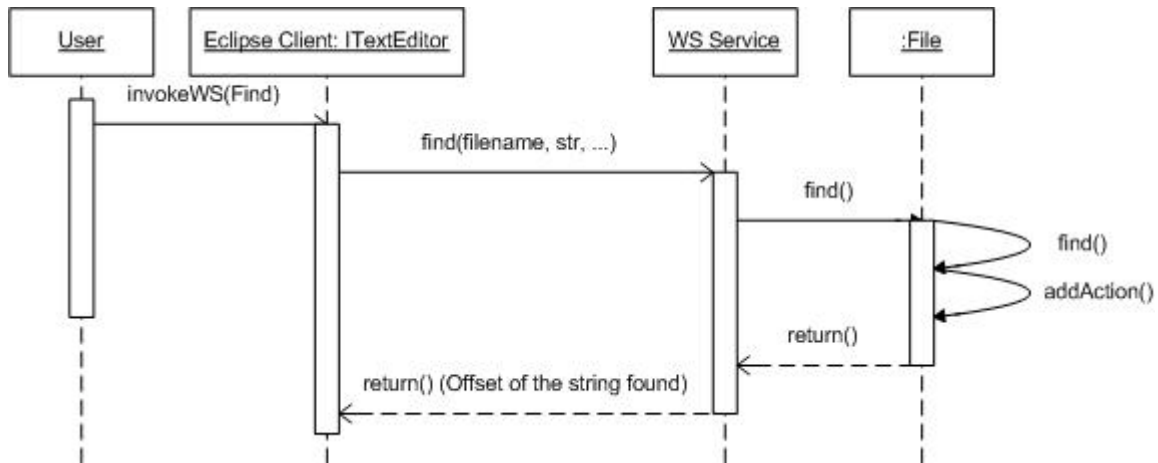


When a new user wants to join an existing editing session, the user first invokes the web service indicating that he/she wants to register. After the WS server receives the request from the Eclipse client, it will register the new client and return the client ID assigned to the user.

The user then use `getFileNames()` to ask for a list of currently opened files in the web server. The WS returns the list of names to the user.

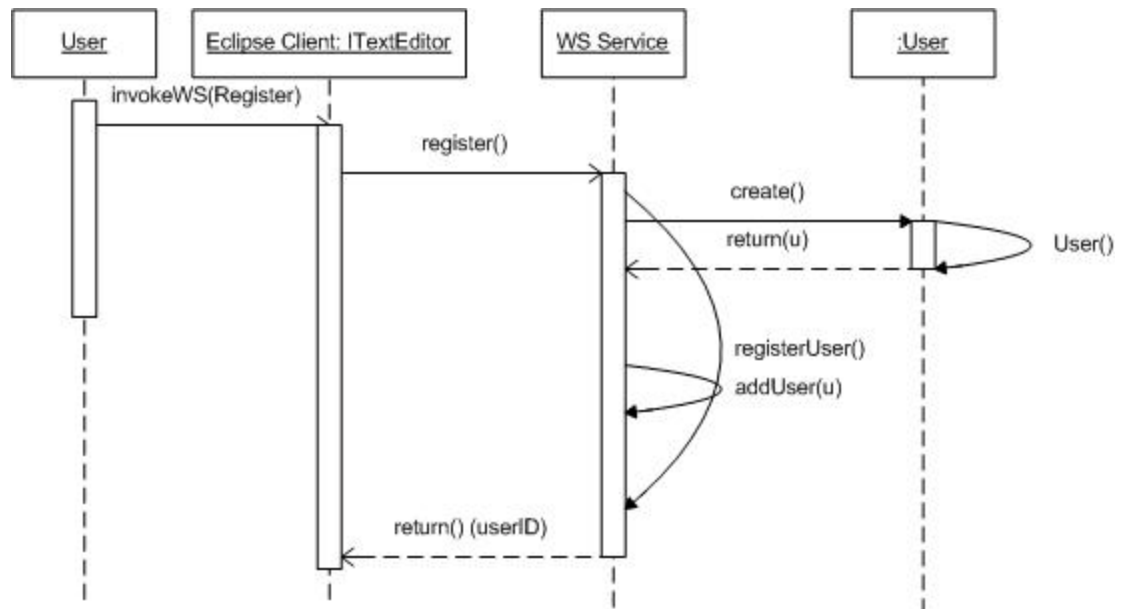
After getting the filenames, the user can choose which list of files to download by calling `download()` (giving a list of file names and its client ID to the WS). The WS returns the contents of these files to the user; in the server it also updates the information on what files this user is currently editing.

5.5.4.3. Find



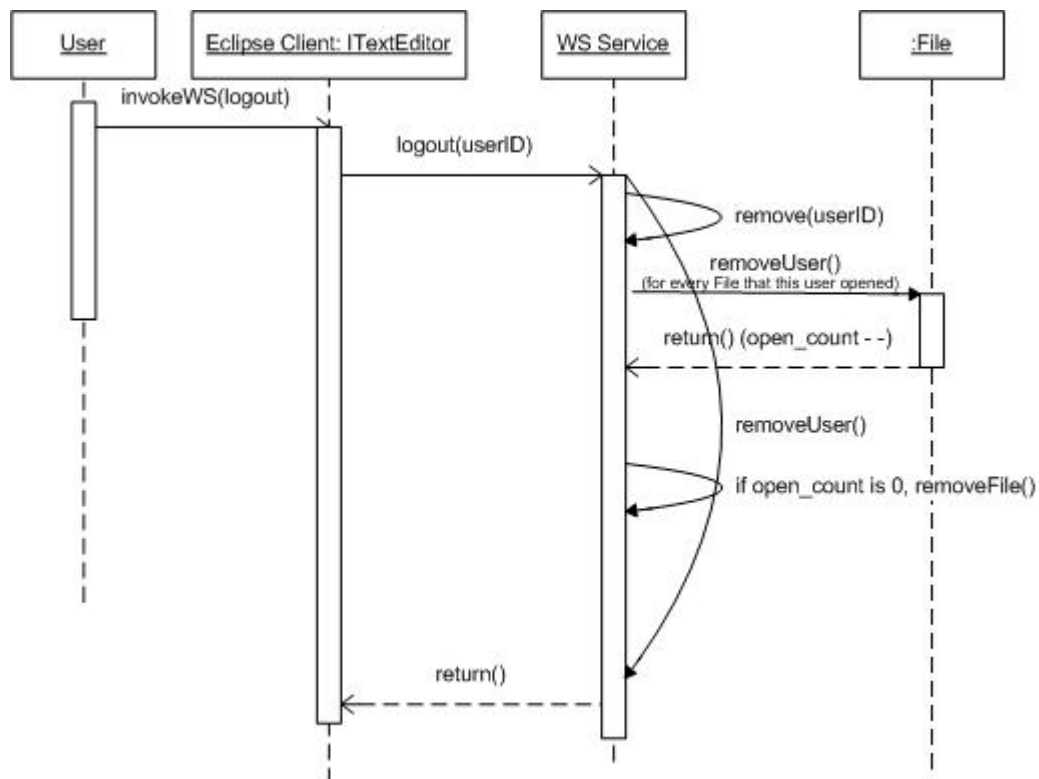
User invokes the Find command. Client sends requests Find() service from WS with correct arguments. The WS calls find() for the particular File object, the File object finds the string and returns the offset, and records this Find action made to itself. The offset is then returned to the client.

5.5.4.4. Register



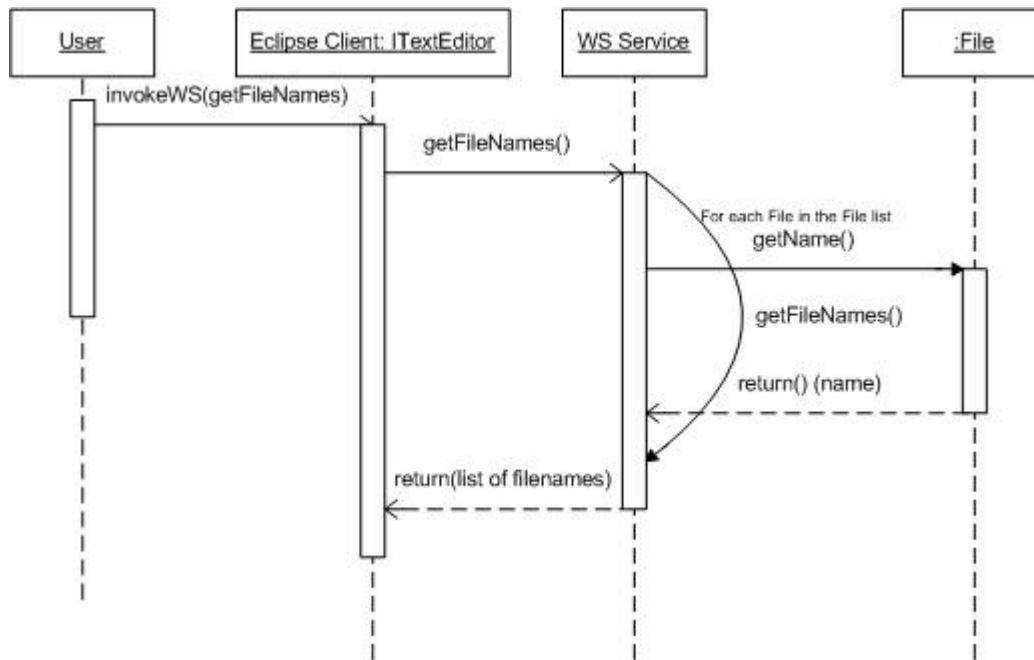
User invokes the Register command and client sends it. WS registers the new user by creating a new User object for the new user, and adds it to the list of Users that the WS maintains. WS then returns the new User ID to the client.

5.5.4.5. Logout



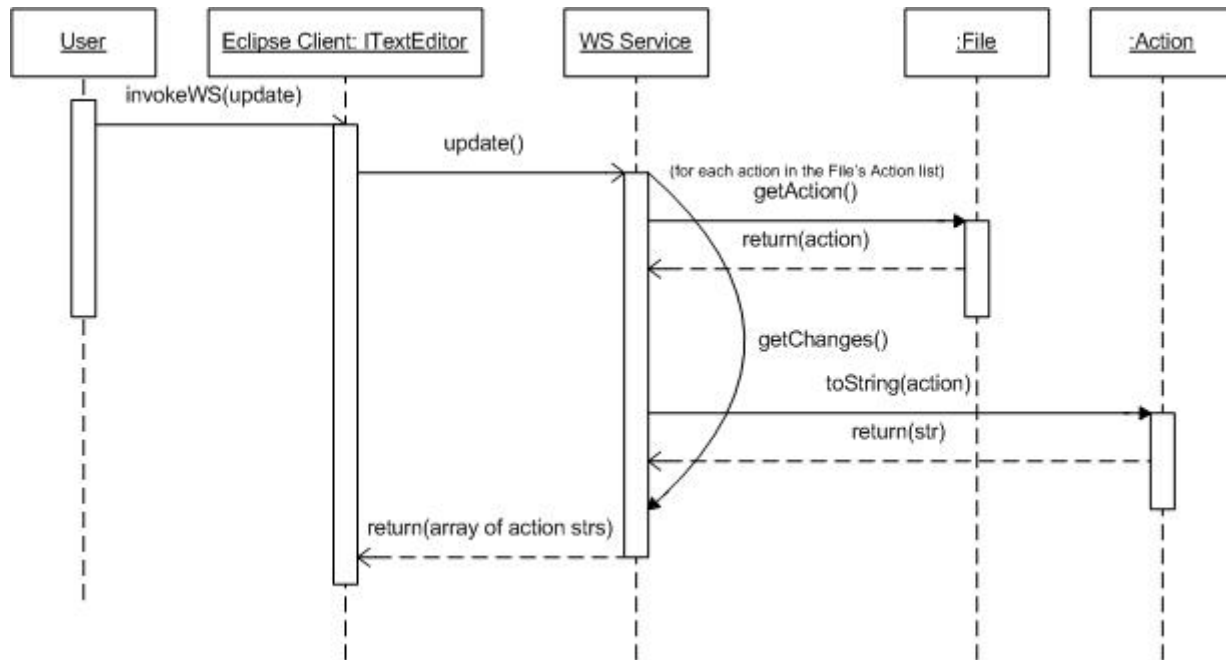
Upon Logout, WS calls its own `removeUser()`, which first removes the User from its User list. Then for each File that this User was editing, decrease the `open_count` for that File in the WS's File list. If the `open_count` is 0, that means this File is no longer being edited by anyone, and WS removes it from the File list.

5.5.4.6. getFileNames



WS calls its `getFileNames()`, which loops through the File list and for each File, gets the names of the File, store them in an array, and return the array of filenames to the client.

5.5.4.7. Update



WS calls its `getChanges()` to handle `update()` calls. For the specified File, go through the File's Action list (i.e. change history). Convert each Action object to a string, and store these action strings in an array. The array of action strings is then returned to the client.

6. Project Plan

6.1. Task Allocation

Task allocation will mostly be on a voluntary basis, done first at the beginning of each phase, and continues throughout the phase as each member progresses and issues are raised. If there are unclaimed tasks, it will be assigned by voting.

6.2. Gantt chart

Please refer to Appendix A for the Gantt chart of task allocations and schedules.

7. Team Organization

7.1. Team Management

We decided to take a flexible approach in our team management. After several meetings we decided it is best for our team to not select an overall team leader, but each member will be consulted as expert on the area they are best skilled at (according to skills and preferences below). For each phase, each member will become expert on the area he/she is assigned to explore, and he/she will be the chief consultant on that subject. Generally for Phase B:

Jianlei Su – Design, chief programmer

Yang Jia – Design, programmer, High-level Test Manager, WS Deployment

Kelvin Chan – Design, programmer, Junit Test Manager

Y.N. Juno Chiao – Design, maintenance/integration plan, report update/organization

7.2. Skills and Preferences

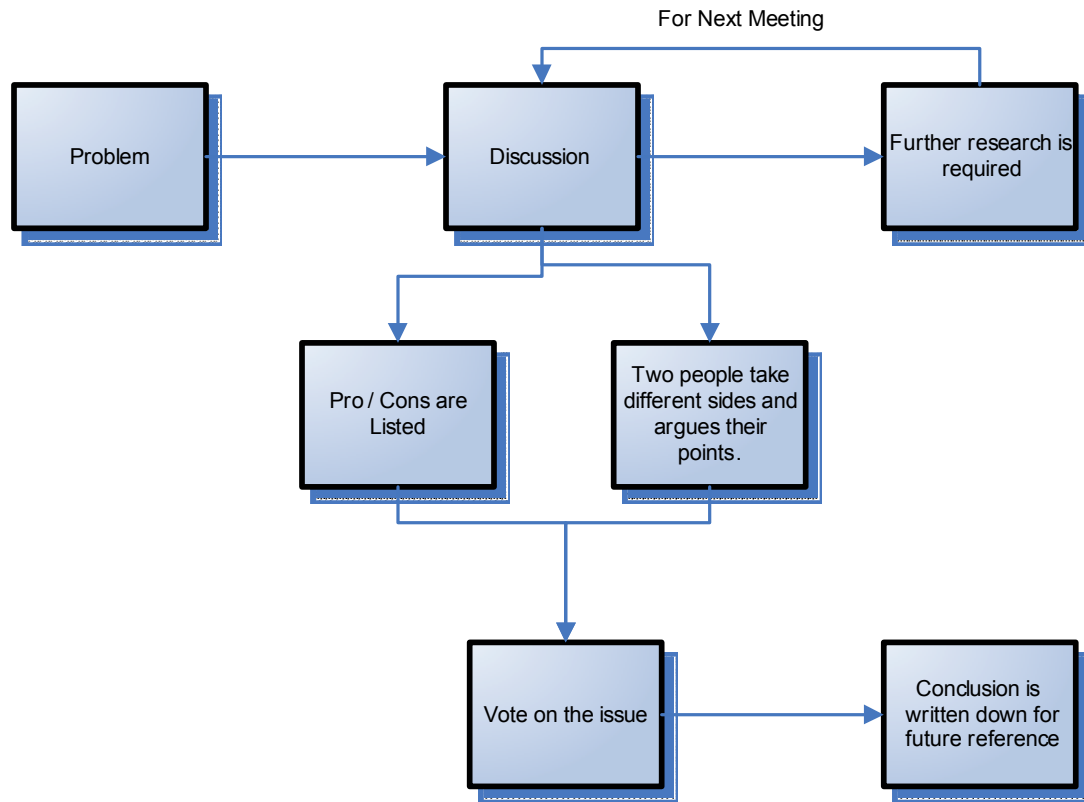
Every team member will participate in the design of the Web Service (phase B) and the client side (Phase C), the following is a list of skills that we believe each person is most skilled at. This will help to determine which task will be allocated to which member.

- Jianlei Su – design and architecture, programming, debugging
- Yuen-nung Chiao– design and architecture, documentation, debugging
- Yang Jia – programming, testing, debugging
- Kelvin Chan– documentations, debugging, testing

7.3. Team Meeting Schedule

- Every Thursday before and after class: short discussion on what to prepare for the Saturday team meeting.
- Every Saturday: In-person team meetings from 2 till completion of discussion. Minutes taken by different team member each meeting.
- Online meetings through MSN during the week when necessary.

Decision Making Strategy



7.4. Methods of Communication:

- Weekly meetings – Saturdays at 2 p.m.
- Email
- MSN Messenger – During the week for online meetings and exchange of information.
- Newsgroup – Yahoo newsgroup was created specifically for this project, where message and files are uploaded for every member to see.

8. Risk Analysis

8.1. Personnel Shortfall

- *Working as a group of 4.* Our project plan and team organization, task allocation and schedule are developed based on a team of four members. Adding one more member might have negative impact on our group.
- *Lost of team member (e.g. due to illness).* Unfortunate accidents may always occur; the only thing that can be done would be preparing for this possibility. *Insufficient planning*
 - Inadequate time allocated for tasks.
 - Under estimation of project difficulty or complexity.
 - Insufficient testing time allocated.

8.1.1. Impact

- *Addition of members:* Adding one more member to the group at this time will cause more harm than good since it will most likely result in more arguments and confusion on the task and structures that have been developed. The whole project plan and team organization has to be restructured and we will lose valuable time.
- *Lost of team member:* If the other members are informed of this near the due date and they are not familiar with the sick member's work and progress, the team might not be able to finish the project deliverables in time.

8.1.2. Prevention

- If some other team breaks up and need to disperse their members into other teams, our team will try to talk to the professor to not take another team member if possible (this has already happened, we did not take another member). If it is inevitable, we will call a meeting as soon as the new team member is decided and reorganize the team structure and project plan. For example, if the new member is good at software development, we will put him/her on the software development part of the project and thus have more manpower in other areas such as testing.
- Each member has the obligation to keep each others informed on the progress and work that we are doing. This will be done through email and especially weekly meetings. If the one team member cannot finish their work before the due date due to illness/accident/etc, a group meeting will be called in person or via msn to decide who will take on the responsibility

8.2. Unrealistic Schedule/Budget

There is the assumption that the professor has planned the project specifically for this term course thus everything should be completed on time and within the budget. But if we are not able to progress according to our project plan schedule due to the members' busy schedules:

8.2.1. Impact

The deliverables of the particular phase might not be completely finished, or it will be of poor quality resulting in bad marks, which is the profit we get for this project.

8.2.2. Prevention

We hold group meetings every weekend to make sure everyone's on schedule. If any team members are concerned about other member's progress and work quality, he/she is

responsible to voice their concern publicly on the team's yahoo course group or on MSN, where the team keeps communication outside of class and regular group meetings.

8.3. Wrong Functionality

Given that our team has discussed the requirements many times, there is a very minimal possibility of misunderstand any part of the requirements. If there are any doubts there have been notes taken and diagrams drawn of the requirements which can be quickly reviewed.

8.3.1. Impact

Our customers in phase C will run into trouble when they find that the services do not behave according to the specifications. Bugs will be generated which decrease our final marks, and extra time and effort will need to be spent on solving the problem.

8.3.2. Prevention

Frequent team discussion will reveal the discrepancies among each member's understanding of the requirements. Once discrepancy is found the team should discuss what correct functionality is and confirm it with the professor or the TAs. Each member is responsible of visiting the course newsgroup and website so each member will be updated on the latest news on the project. During development each member should post/commit the changes they made to the private group newsgroup and CVS so everyone can see the current development of the project. This way if there are any issues it can be quickly identified and dealt with.

8.4. Gold-Plating

Too many unnecessary functions are developed in order to make the project look impressive.

8.4.1. Impact

We will spend too much time on unnecessary functions and lose time on testing on the necessary ones. The WS interface might seem too complicated to use for other teams since the more functions there are the more they have to learn about them. This may result in less team choosing our services and also may contain more hidden bugs.

8.4.2. Prevention

While we discuss the design of this project we as a group constantly remind each other to keep things simple and not to do too much. We plan each task carefully such that it will be versatile yet not too complicated as to potentially create major issues.

8.5. Requirement Volatility

There might be an issue of redesigning or reworking of tasks, which maybe caused from unclear requirements or changing requirements.

8.5.1. Impact

Time will be lost on trying to redesign things that are already done, when this time can be useful for testing or debugging.

8.5.2. Prevention

To minimize the possibility of requirement adjustments, clear well defined documents will be created to avoid any possibilities of this issue occurring. In the case that requirements must be changed the architecture of the system has been designed to be loosely connected and thus changes will be minimal in terms of the entire project.

8.6. Bad External Tasks

8.6.1. Impact

If our team has failed to select a web-service that is relatively bug free and simple to implement then it may cause problems in our development of the omni-editor.

8.6.2. Prevention

To minimize the risk of this event each web-service will have to be looked at thoroughly and detailed analyses have to be done. There are websites set up by each team with their documentations. We also need to talk to those team members to see if they seem trustworthy enough to uphold their various promises of customer support.

8.7. Capability Shortfalls

The technologies that have been selected may have short comings which are not known until it is used.

8.7.1. Impact

As the problems manifest, members will spend hours of time trying to debug or solve the problem. This time and effort is wasted.

8.7.2. Prevention

The environment and technologies have been previously reviewed by the members of the team and tested for the functionalities that we need to use. If there are more problems regarding the technologies, we will contact the TAs and the professor for help, and post the problem on the newsgroup. Since many other groups use the same technology, we might be able to resolve the issue quickly.

9. Appendix A: Task Allocation

ID	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	Phase A	20 days	Fri 9/17/04	Fri 10/8/04		
11	Phase B	20 days	Mon 10/11/04	Sat 11/6/04	1	
12	Overall Design Plan	3 days	Mon 10/11/04	Wed 10/13/04		Kelvin,James,Ray,Juno
13	Class Skeleton Design	5.5 days	Tue 10/12/04	Tue 10/19/04	12	James,Juno,Kelvin,Ray
14	Group 3 (1,2,9) Implementation	12.25 days	Thu 10/14/04	Mon 11/1/04	13	
15	OmniEditor Class	6.5 days	Fri 10/22/04	Mon 11/1/04	19	
16	Programming	1 day	Fri 10/22/04	Mon 10/25/04		Ray
17	Unit Testing	4 days	Mon 10/25/04	Fri 10/29/04	16	James
18	Documentation	1.5 days	Fri 10/29/04	Mon 11/1/04	17	James,Ray
19	OmniEditor Buffer Class	3.5 days	Mon 10/18/04	Fri 10/22/04	23,27	
20	Programming	1 day	Mon 10/18/04	Tue 10/19/04		Ray
21	Unit Testing	1 day	Tue 10/19/04	Wed 10/20/04	20	Kelvin
22	Documentation	1.5 days	Wed 10/20/04	Fri 10/22/04	21	Kelvin,Ray
23	OmniEditor User Class	2.25 days	Thu 10/14/04	Mon 10/18/04		
24	Programming	0.5 days	Thu 10/14/04	Thu 10/14/04		Ray
25	Unit Testing	1 day	Fri 10/15/04	Fri 10/15/04	24	Kelvin
26	Documentation	0.75 days	Mon 10/18/04	Mon 10/18/04	25	Kelvin,Ray
27	OmniEditor File Class	2.25 days	Thu 10/14/04	Mon 10/18/04		
28	Programming	0.5 days	Thu 10/14/04	Thu 10/14/04		Ray
29	Unit Testing	1 day	Fri 10/15/04	Fri 10/15/04	28	Kelvin
30	Documentation	0.75 days	Mon 10/18/04	Mon 10/18/04	29	Kelvin,Ray
31	OmniEditor User File Class	2.25 days	Thu 10/14/04	Mon 10/18/04		
32	Programming	0.5 days	Thu 10/14/04	Thu 10/14/04		Ray
33	Unit Testing	1 day	Fri 10/15/04	Fri 10/15/04	32	Kelvin
34	Documentation	0.75 days	Mon 10/18/04	Mon 10/18/04	33	Kelvin,Ray
35	OmniEditor Action / FindActi	2.25 days	Thu 10/14/04	Mon 10/18/04		
36	Programming	0.5 days	Thu 10/14/04	Thu 10/14/04		Ray
37	Unit Testing	1 day	Fri 10/15/04	Fri 10/15/04	36	Kelvin
38	Documentation	0.75 days	Mon 10/18/04	Mon 10/18/04	37	Kelvin,Ray
39	Integration Testing / Bug Fixing	1 day	Mon 11/1/04	Tue 11/2/04	14	Kelvin,James,Ray
40	Update Documentation	7 days	Mon 10/11/04	Tue 10/19/04		Juno
41	Report Review	2 days	Thu 11/4/04	Sat 11/6/04	40	
42	Phase C	20 days	Mon 11/8/04	Fri 12/3/04	11	
43	System Analysis	4 days	Mon 11/8/04	Thu 11/11/04		
44	Requirements gathering	2 days	Mon 11/8/04	Tue 11/9/04		
45	Omni Editor Design	3 days	Fri 11/12/04	Tue 11/16/04	44,43	
46	Implementation	8 days	Wed 11/17/04	Fri 11/26/04	45	
47	Testing & Bug Fixing	3 days	Mon 11/29/04	Wed 12/1/04	46	
48	Report Review	2 days	Thu 12/2/04	Fri 12/3/04	47	

