



THE AUSTRALIAN NATIONAL UNIVERSITY

**FACULTY OF ENGINEERING AND INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE**

Development of WebScope3

Xiaobin Wang

(u4266538)

November 2007

This is the final report for the COMP6703 eScience Project (Semester 2, 2007)

TABLE OF CONTENTS

ACKNOWLEDGMENTS

ABSTRACT

1. Introduction.....	6
1.1 Overview.....	6
1.1.1 Background.....	6
1.1.2 Purpose.....	8
1.1.3 Statement of Scope.....	8
1.1.4 Target Results and Project Customers.....	9
1.1.4.1 Target Outcomes.....	9
1.1.4.2 Outputs.....	9
1.1.4.3 Deliverables.....	9
1.1.4.4 Customers.....	9
1.1.4.5 Stakeholders.....	10
1.2 Introduction To WebScope.....	10
1.3 Introduction To WebScope With Cooe.....	12
1.4 Key technologies involved.....	14
1.5 Report Brief.....	15
2. Requirement Analysis.....	16
2.1 Software Requirements.....	16
2.2 Operating System.....	16
2.3 Languages.....	16
2.4 Client Requirements.....	17
3. Scheduling.....	19
3.1 Planned Timetable.....	19
3.2 Actual Progress.....	20
3.3 Gantt Charts of the Two Timetables.....	21
4. Modeling.....	22
4.1 Structure of WebScope3.....	22
4.2 High Level Design.....	22
4.2.1 System Context.....	22
4.2.2 Domains.....	24
4.3 Detailed Design.....	25
4.3.1 Communication Between Packages.....	25
4.3.2 Database Design.....	25
4.3.3 WebScope3 Domain.....	26
4.3.4 dataAccessDomain Domain.....	28
4.3.5 dataServerDomain Domain.....	29
4.3.6 graphicsDomain Domain.....	30

4.3.7 Mapping Domain.....	31
4.3.8 Applet Domain.....	31
5. Implementation.....	33
6. Cognitive Walkthrough and Heuristic Evaluation.....	36
6.1 Cognitive Walkthrough.....	36
6.2 Heuristic Evaluation.....	36
7. Testing.....	38
7.1 Unit Test.....	38
7.2 Module Test.....	38
7.3 Acceptance Test.....	42
8. Design Patterns.....	43
8.1 WebScope3 Domain.....	43
8.2 dataAccessDomain Domain.....	43
8.3 dataServerDomain Domain.....	44
8.4 graphicsDomain Domain.....	45
8.5 applet Domain.....	47
8.6 Comparison between the WebScope3 and EScope4 systems.....	47
8.6.1 Transformation from EScope4 to WebScope3.....	48
8.7 Conclusion.....	50
9. Future Work.....	51
10. Summary of Contributions and Conclusion.....	52
10.1 Summary of Contributions.....	52
10.2 Personal Statement.....	52
11. Reference.....	54
Appendix A: Cognitive Walkthrough Details.....	56
Appendix B: Heuristic Evaluation.....	64
Appendix C: Progress on the Servlet-based WebScope System.....	70
Appendix D: Project Log.....	73
Appendix E: Installation and Compilation Guide.....	79

Acknowledgements

During the entire project, I have come across numerous problems and troubles. However, I am never afraid of them because I can always receive help from my supervisor and client, Dr. Henry Gardner. Henry is always ready to answer my questions whenever I'm stuck, and give me encouragement whenever I make some progress. His comprehensive knowledge and expertise is really a great help to me. I think I have learned both how to do a project and how to work hard to achieve more from him. I greatly appreciate everything Henry has done for me.

Dr. Raju Karia is another great help for me. He has introduced many of the latest Java techniques to me. I have learned a lot from him.

I should also thank Mr. Ajith M. Jose, Mr. Le Ma, and Mr. Zhongshan Tan for their hard working. The current WebScope3 project is the result of our joint work. No one can achieve the final success on his own.

Abstract

This report outlines the development of the WebScope3 application for the eScience project course COMP6703, at the Australian National University.

The core idea behind this project is retrieving nuclear fusion datasets from an MDSPlus based data server using a web browser. The use of the Java object/relational package “Hibernate” to manage and cache the retrieved datasets is a highlight of this project.

In this project what I have done is as follows:

- I have implemented the system using “Cooee”. Cooee is an independent branch of the Echo2, EchoPointNG and Echo2 Extras source code to construct AJAX-based user interfaces for Java programs. It is an ongoing project to further develop the Echo code base into a highly robust web UI framework.
- I have combined the former projects done by Mr. Le Ma and Mr. ZhongShan Tan.
- I have added the following new features to the WebScope system:
 - Metadata Query feature
 - Java Applet graph-display mode.
 - Dynamic Metadata Table

1 Introduction

1.1 Overview

1.1.1 Background

Currently nuclear fusion research is being performed at various locations of the world. Millions of nuclear fusion datasets are generated as a result of various nuclear fusion experiments happening at different research organizations. It is very important for the researchers to share the datasets for mutual research benefits. The safe storage and retrieval of fusion datasets is currently done using a data storage and management system known as “MDSPlus” [23].

For research purposes, several students supervised by Dr. Henry Gardner have worked on “Scope projects” over the past two years. By the end of Semester 1 2007, there were two versions of Scope: “EScope” and “WebScope”. The latter is considered to be a web-based version of the former, with the intention of enhancing the universality of its usage.

eScience Data Grid:

In general terms, a Data Grid can be thought of as a loose federation of networked data stores which is supported by grid computing [7].

This concept has evolved because with the development of scientific and engineering applications, the need to access large amounts of distributed data has become more and more obvious.

EScope:

Researchers from different parts of the world can connect to a MDSPlus and retrieve the required datasets for their own research purposes. Special, client-side software applications like “EScope” are used to retrieve datasets from MDSPlus. The client-side applications plot the retrieved datasets as graphs or as statistical tables.

However, there are many limitations to this way of retrieving datasets from the MDSPlus server.

- No caching
- No searching
- Lack of flexibility in schemes
- Cannot compare or combine different MDSPlus data stores

- Client-side software needs to be installed. No present web-based clients.

Noticing the deficiencies stated above, “WebScope” has been developed.

WebScope:

The project “WebScope” is a project to resolve the existing issues with the retrieval of datasets from the MDSPlus server using EScope. Another highlight of “WebScope” project is the use of Java object relational mapping solution “Hibernate” to solve the caching issues with “EScope” data retrieval system.

At the time of the commencement of this project, there were two versions of WebScope available. Their genealogy is shown in the figure below (Figure 1.1):

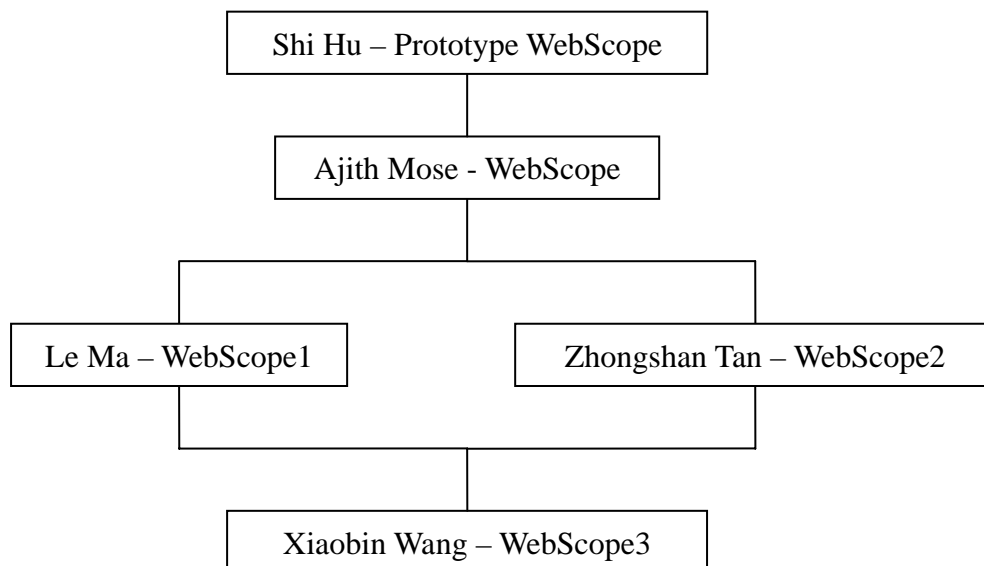


Figure 1.1 The Genealogy of WebScope

As the current two versions of WebScope application [Le Ma 2006] [Zhongshan Tan 2006] both have their own advantages, the need to combine the two versions and make an integrated version of WebScope application is obvious. That is, we should use AJAX to develop the GUI to improve the responsiveness of the application and add the dynamic metadata feature to the application to increase the functionality.

Having taken the reasoning above into account, Dr. Henry Gradner and Dr. Raju Karia has decided to introduce that implementing the “WebScope” with Cooe. Cooe is a Java implemetation of AJAX. I shall briefly describe both of the concepts below:

Ajax, shorthand for Asynchronous JavaScript and XML (Extensible Markup Language), is a web development technique for creating interactive web applications.

The intent of Ajax is to make web pages respond to the exchange of small amounts of data between the browser and the server, rather than reloading the entire web page each time the user makes a change. This is meant to increase the web page's interactivity, speed, and usability [2].

Cooee is a branch of the Echo2, EchoPointNG and Echo2 Extras [Nextapp 2005] source code. It is an ongoing project to further develop this code base into a highly robust web UI framework. The project also aims to keep as much API compatibility with the Echo code as possible to aid projects wishing to move from Echo2 to Cooee [17].

1.1.2 Purpose

The main purposes of the current project are as follows:

- Combine the two versions of the WebScope project developed by Mr. Le Ma and Mr. Zhongshan Tan respectively, which means that the new Web Scope system should be able to generate dynamic metadata, allow users to contribute metadata to the database, and make it possible for the web clients and web servers to interact via Cooee.

- Add the following new features to the Web Scope system:
 - Allow users to query metadata by keywords
 - Display a list of metadata when a user logs on
 - Amend the dynamic metadata feature of Le Ma's version and display dynamic metadata tables that a user has added
 - Display user log information and metadata information in the main screen.
 - Provide user with two modes to plot graph: the Image Map (AJAX) mode and the Java Applet mode.

- Review the software architecture of WebScope and define the transformations needed to convert EScope to WebScope. Perhaps improve the existing WebScope software architecture.

1.1.3 Statement of scope

The scope of the current WebScope project is not limited to adding features on the basis of the current system. Some modification to the whole structure of the system may be necessary.

1.1.4 Target results and project customers

1.1.4.1 Target outcomes

The “Development of Web Scope” project has five target outcomes:

- Migration from the existing Echo2 framework to Cooee.
- Adding the metadata and user-access log database tables to the Cooee version of the WebScope application. This will make it possible for the application to record user access log and will let the users contribute static metadata to the database.
- Let the user “dynamically” contribute a customized table to the database and display the table to the user in real time.
- Let the user query metadata by different keywords.
- Let the user switch between two graph-display modes: the Image Map (AJAX) mode and the Java Applet mode.

1.1.4.2 Outputs

The target outputs for this “Development of WebScope3” project are the working “WebScope” application and the final project report.

1.1.4.3 Deliverables

Deliverables are a superset of the outputs obtained as a result of the project.

The project deliverables are listed below:

- Source code of the Web Scope with Cooee application
- Final report of the system
- All the other documents related to the project

1.1.4.4 Customers

The project customers who are vital in ensuring the success of achieving target outcomes are nuclear fusion researchers or scientists from different parts of the globe.

1.1.4.5 Stakeholders

The stakeholders of this project are described below.

StakeHolder	Name	Contact Details
Supervisor & Client	Dr. Henry Gardner	Henry@cs.anu.edu.au
Supervisor	Dr. Raju Karia	rjkaria@smartchat.net.au
Developer	Xiaobin Wang	tommywang1981@gmail.com

1.2 Introduction to WebScope

“WebScope” is the system proposed by Dr. Henry Gardner and Dr. Raju Karia and realised by several students in the eScience program at ANU. Figure 1.2 gives an overview of the WebScope.

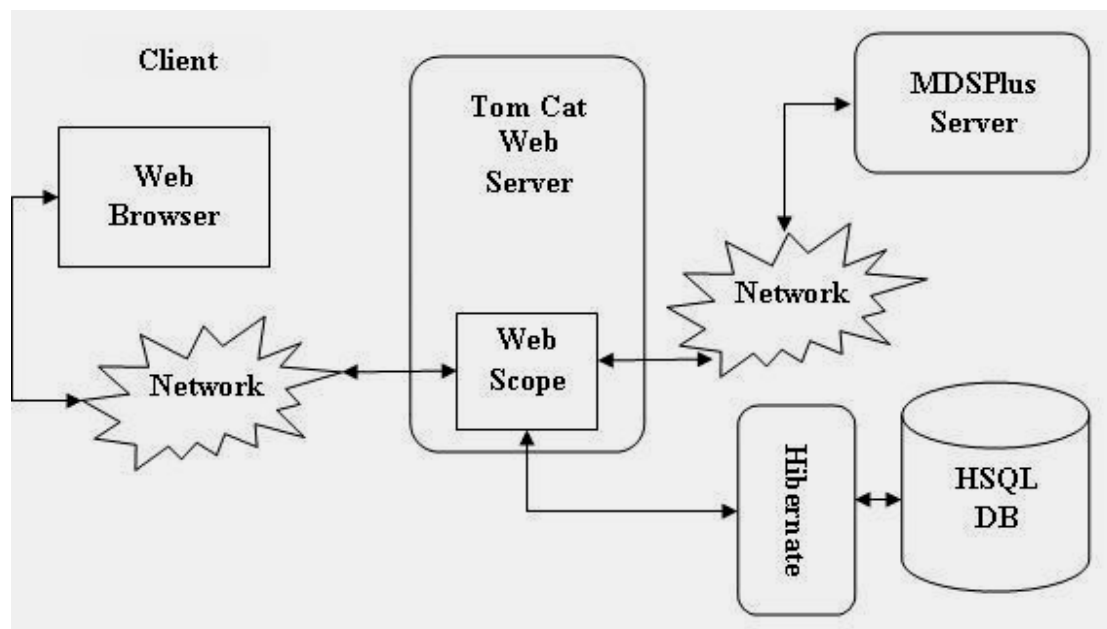


Figure 1.2 High level structure of WebScope system

WebScope runs on a web server and waits for the client requests coming through web browsers. It maintains a relational database called “HSQLDB” for storing the retrieved datasets from the MDSPlus server in the figure, but this could be any

relational database. All database interactions (storage and retrieval of datasets) are handled using the Hibernate package, which is an object/relational persistence and query service for Java.

WebScope works in the following way:

When a client request arrives at the “WebScope”, it returns a web GUI to the client.

- 1) The web-based MDSPlus data retriever application, “WebScope” runs on a Web Server (Apache Tomcat) and accepts the client requests coming through web browsers.
- 2) Upon receiving a client request the “Web Scope” checks for the client requested data in the attached HSQL database. If the requested data set is available in the database it is retrieved and sent to the client Web browser. If the requested data set is not available in the database, it is retrieved from the MDSPlus server and sent to the client Web browser. A copy of the retrieved data set is stored in the data base for later use, so that the later requests for the same data sets can be accomplished faster.
- 3) Using applets the retrieved data sets are plotted as graphs at the client Web browser.

The benefits of using WebScope are given below

- Users need not install any specific software to retrieve and view the MDSPlus data. They just needed to connect to the Web Server using a Java Applet- enabled web browser.
- Users are not restricted to the data sets in the local relational database. If requested data sets are not available in the database they are retrieved from the MDSPlus server and shown to the client.
- The use of Hibernate makes the caching faster.
- The utilization of the resources at the client side is decreased dramatically compared with the full EScope application.

1.3 Introduction to WebScope with Cooee

Compared with WebScope, WebScope with Cooee (which will be quoted as “WebScope3” in the following context) does not change much in terms of software architecture and the structure of the system. The main difference between “WebScope” and “WebScope3” is that WebScope uses Java Servlet to create web pages, while “WebScope3” uses Java Language in Cooee framework to develop web-based applications. The technical change can provide users with a more interactive, responsive, user-friendly, functional, stable, and beautiful GUI. In addition, the “Dynamic Metadata” feature does not actually work in the WebScope application in that dynamic tables cannot be stored and retrieved from database. It’s now completed in WebScope3. The change from “WebScope” to “WebScope3” is shown below:

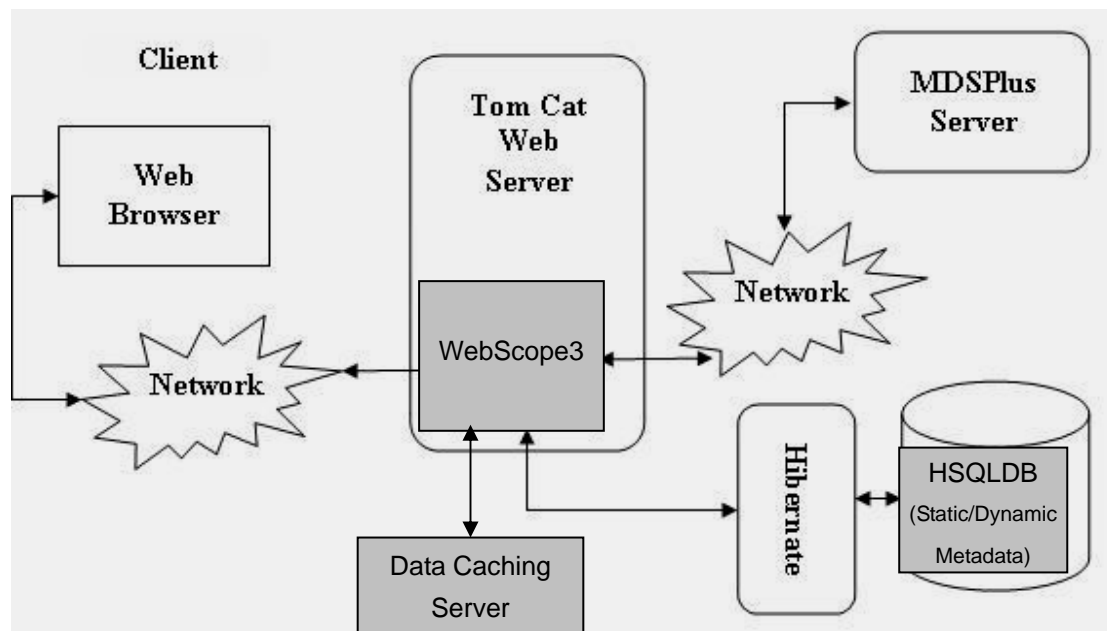


Figure 1.3 High level structure of WebScope3 system

The main benefits of using “WebScope3” are given below

- The interaction between the clients and servers are greatly improved because of the Cooee technology. We do not need to reload the whole page when users request a small change to a part of it. Because of the Ajax-based rendering engine, only the requested part of a page is updated per users’ requests.
- In the period of development, the developer does not have to think in terms of

"page-based" applications and is able to develop applications using the conventional object-oriented and event-driven paradigm for user interface development.

- Users can not only read experiment data from the database, but also contribute static/dynamic metadata to the server, which makes the application more interactive, flexible and functional. For the dynamic metadata feature, users can customize their own tables according to their requirements. New Java files and Hibernate mapping files are generated for the customized tables and then the Java files are compiled and the Java classes are loaded in runtime. Afterwards, the table content is saved into database. This gives users more control on the system.
- Users have two modes to plot the graph data now: the Image Map mode and the Java Applet mode. The former is quicker and simpler and the latter is more functional and interactive. It's up to the users to decide which mode suits them best. The two modes make the WebScope3 system more flexible.

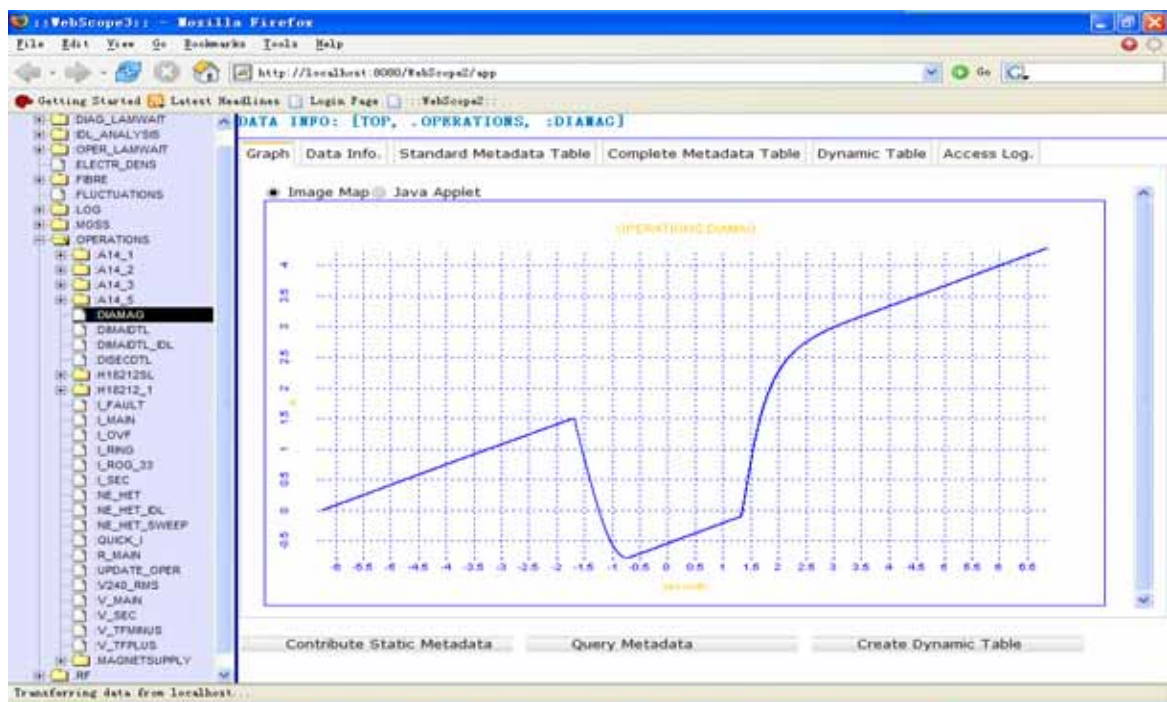
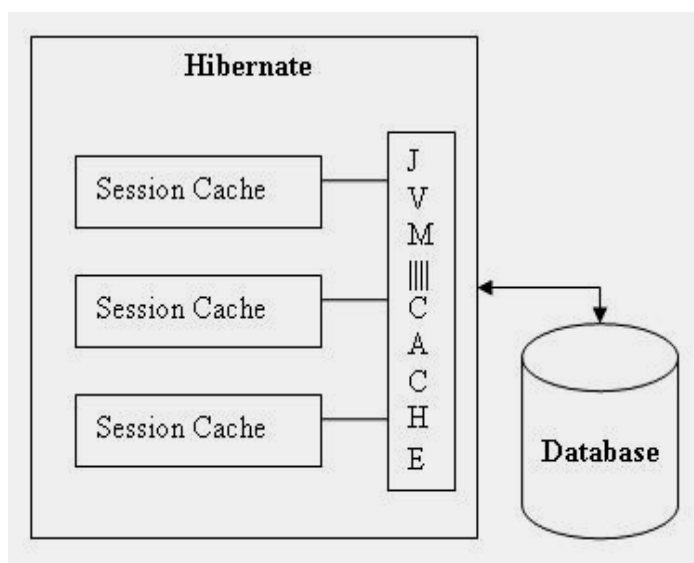


Figure 1.4 The interface of the WebScope3 system

1.4 Hibernate

The key technologies involved in this project are Cooe and Hibernate. The former one has been briefly discussed in Section 1.1.1. The following paragraph explains what Hibernate is.

Hibernate is a Java-based object/relational mapping solution. It can dynamically create database tables based on the information provided. It is much easier to retrieve and store data using the Hibernate package than the existing ones. The dual layer caching mechanism of Hibernate makes the data caching faster.



The main features of the Hibernate package described in its web site are listed below [14]:

Hibernate is a powerful, ultra-high performance object/relational persistence and query service for Java. Hibernate lets you develop persistent classes following common Java idiom - including association, inheritance, polymorphism, composition and the Java collections framework. The Hibernate Query Language, designed as a minimal object-oriented extension to SQL, provides an elegant bridge between the object and relational worlds. Hibernate also allows you to express queries using native SQL or Java-based Criteria and Example queries. Hibernate is now the most popular object/relational mapping solution for Java. The most important feature of Hibernate for this proposal is that Hibernate uses HDLCA (Hibernate Dual-Layer Cache Architecture). It maintains two levels of cache. One is session-level cache and the other is JVM-level cache. The session-level cache serves one client or one session at one time, it resolves circular/shared references and repeated requests for the same

instance in a particular session. The JVM-level cache resolves repeated requests for different sessions using the same instance of JVM.

1.5 Report Overview

This report is divided into 11 major sections.

- Section 1, Introduction, gives an overview of the project, as well as the expected outcomes, outputs and the major deliverables of the project. Through this section the customers and major stakeholders of the project are identified. It also gives a brief introduction to Cooee.
- Section 2, Requirement Analysis, describes detailed user requirements, softwares and other tools needed to accomplish this project.
- Section 3, Scheduling, describes the various phases involved in this project and the number of days allotted for each phase.
- Section 4, Modelling, also known as Designing, uses UML model to design the structure of this software. It also defines the database structure.
- Section 5, Implementation, describes the ideas and methods used to achieve the functionalities of “WebScope” application with Cooee.
- Section 6, Cognitive Workthrough, evaluates the usability of the WebScope3 system.
- Section 7, Testing, tests the whole system to ensure that it’s working properly.
- Section 8, Design Pattern Analysis, analyses the various domains of the WebScope3 system and then applies the suitable patterns to each domain.
- Section 9, Future Work, gives suggestions on further development.
- Section 10, Conclusion, draws a conclusion from this project.
- Section 11, Reference List, shows the various references used for accomplishing the “WebScope” application with Cooee.

2. Requirement Analysis

2.1 Software Requirements

The main software and tools used for the development of “WebScope with Cooee” are given below:

- Apache TomCat 5.5
- Hibernate3.0
- JDK1.6
- HSQLDB
- MDSPlus server
- FireFox & Internet Explorer
- Eclipse 3.3.0
- Cooee Framework 1.0.2

2.2 Operating system

The WebScope3 application is platform-independent. All we need to get this application working is a browser with Java plug-in.

2.3 Languages

- Programming Language: Java combined with Ajax and OpenJFX
- Database Querying Language: HQL (Hibernate Query Language)

2.4 Client Requirements

No.	Req. Name	Description	Priority
R1	Migrating from the existing Echo2 framework to Karora Cooee	“WebScope3” should be migrated from the existing Echo2 framework to Karora Cooee.	High
R2	Using Web Browser	“WebScope3” should allow users to connect to the MDSPlus server and retrieve required datasets via web browsers.	High
R3	Using Hibernate	“WebScope3” should use the benefits of Java object relational mapping solution “Hibernate” to efficiently cache the datasets retrieved from the MDSPlus server.	High
R4	Plotting Graph	“WebScope3” should be able to plot graphs based on the retrieved MDSPlus datasets.	High
R5	Data download	“WebScope3” should allow users to download specific data in the form of text file, so that users can view and analyses data off-line	Medium
R6	Platform Independence	“WebScope3” should be able to work on all major platforms.	High
R7	Getting and displaying user details	User details should be obtained when users register. This information should be displayed to the users themselves when they log on.	High
R8	Tracking user activities	“WebScope3” should have a mechanism to track every action of users. The track log should also be displayed to users in real time.	Medium
R9	Contributing static metadata	“WebScope3” should provide users with a mechanism to fill in the fields of a fixed metadata table and contribute the metadata to the database.	High
R10	Displaying static metadata	“WebScope3” should allow users to view all contributed static metadata	High

No.	Req. Name	Description	Priority
R11	Querying static metadata	“WebScope3” should allow users to query all contributed static metadata by different keywords.	Medium
R12	Contributing dynamic metadata	“WebScope3” should allow users to dynamically create customized metadata tables and contribute them to database.	High
R13	Displaying dynamic metadata	“WebScope3” should allow users to view all contributed dynamic metadata	High
R14	Providing two types of graph-displaying methods	“WebScope3” should provide users with both the Image Map and the Applet methods to display graphs.	High

Table 2.1 Major requirements of “WebScope3”

3. Scheduling

3.1 Planed timetable

The initial planned timetable is as below: (Table 3.1)

Phase	Date	Expected Duration(days)	Tasks	Notes
1	23 rd Jul ~ 12 th Aug	21	<ul style="list-style-type: none"> ✓ Understanding requirements ✓ Studying the new techniques ✓ Studying the current Web Scope system 	<ul style="list-style-type: none"> ● Choose and meet supervisor ● Understand requirements ● Decide tools and install software on the computer ● Study old reports
2	13 th Aug ~ 2 nd Sep	21	Modeling	<ul style="list-style-type: none"> ● Analyze existing project ● Modeling and design architecture for the new project ● Learn various software
3	3 rd Sep ~ 7 th Oct	35	Implementation	<ul style="list-style-type: none"> ● Do coding, Testing and Debugging
4	8 th Oct ~ 21 st Oct	14	Documentation and final report creation	<ul style="list-style-type: none"> ● Complete all documents ● Write the final report
5	22 nd Oct ~ 28 th Oct	7	Preparation for final presentation	<ul style="list-style-type: none"> ● Edit slides for final persentation

Table 3.1 Planned Timetable for “WebScope3” project

3.2 Actual progress

The actual timetable is as below:

Phase	Date	Expected Duration(days)	Tasks	Notes
1	23 rd Jul ~ 6 th Aug	14	<ul style="list-style-type: none"> ✓ Understanding requirements ✓ Studying the new techniques ✓ Studying the current WebScope system 	<ul style="list-style-type: none"> ● Choose and meet supervisor ● Understand requirements ● Decide tools and install software on the computer ● Study old reports ● Study old codes
2	7 th Aug ~ 20 th Aug	14	Modeling	<ul style="list-style-type: none"> ● Analyze existing project ● Modeling and design architecture for the new project ● Learn various software ● Perform some experiment on the existing WebScope system
3	21 st Aug ~ 7 th Sep	49	Implementation	<ul style="list-style-type: none"> ● Do coding, Testing and Debugging
4	8 th Oct ~ 21 st Oct	14	Documentation and final report creation	<ul style="list-style-type: none"> ● Complete all documents ● Write the final report
5	22 nd Oct ~ 28 th Oct	7	Preparation for final presentation	<ul style="list-style-type: none"> ● Edit slides for final presentation

Table 3.2 Actual Timetable for “WebScope3” project

Please refer to Appendix E for details of the actual progress per week.

3.3 Gantt charts of the two timetables

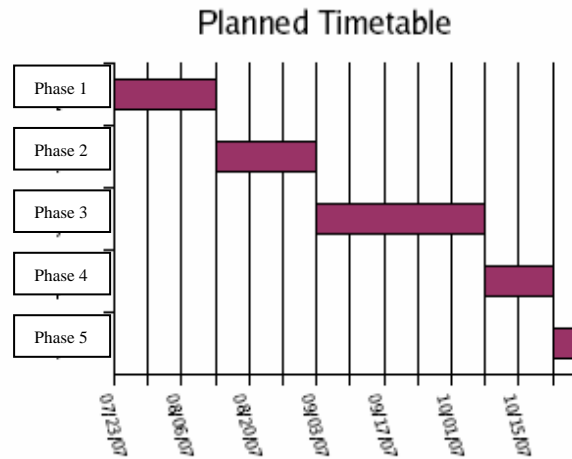


Figure 3.1 Planned Timetable Gantt Chart

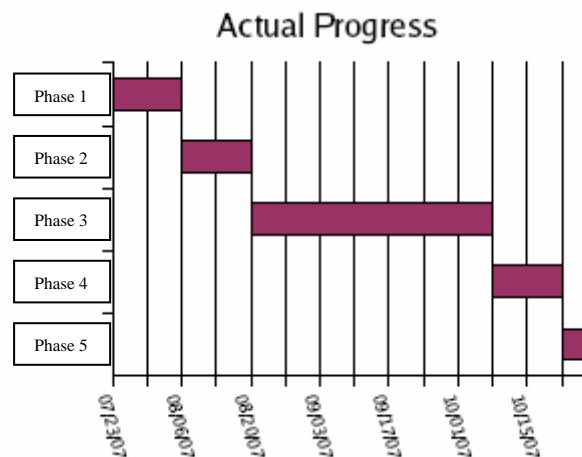


Figure 3.2 Actual Progress Gantt Chart

From the Gantt charts above, we can see that the first two phases of the actual progress took 1 week shorter respectively than planned, but the Implementation phase took two weeks longer. This is because I devoted more than 40 hours per week on the project, and with the help of Dr. Henry Gardner and Dr. Raju Karia, the Modeling phase went quite smoothly. Therefore, we had more time to add more features and functions to the WebScope3 system, which resulted in the longer Implementation phase. In addition, more than 2 weeks were spent on experimenting with new technologies such as the JavaFX scripting language. Although these technologies have not been applied in the WebScope3 system, the experience we have gained is quite valuable.

4. Modeling

4.1 Structure of WebScope3

The structure of WebScope3 is shown in Figure 1.3:

The structure of the WebScope3 system is largely the same as that of the WebScope2 system.

4.2 High Level Design

4.2.1 System Context

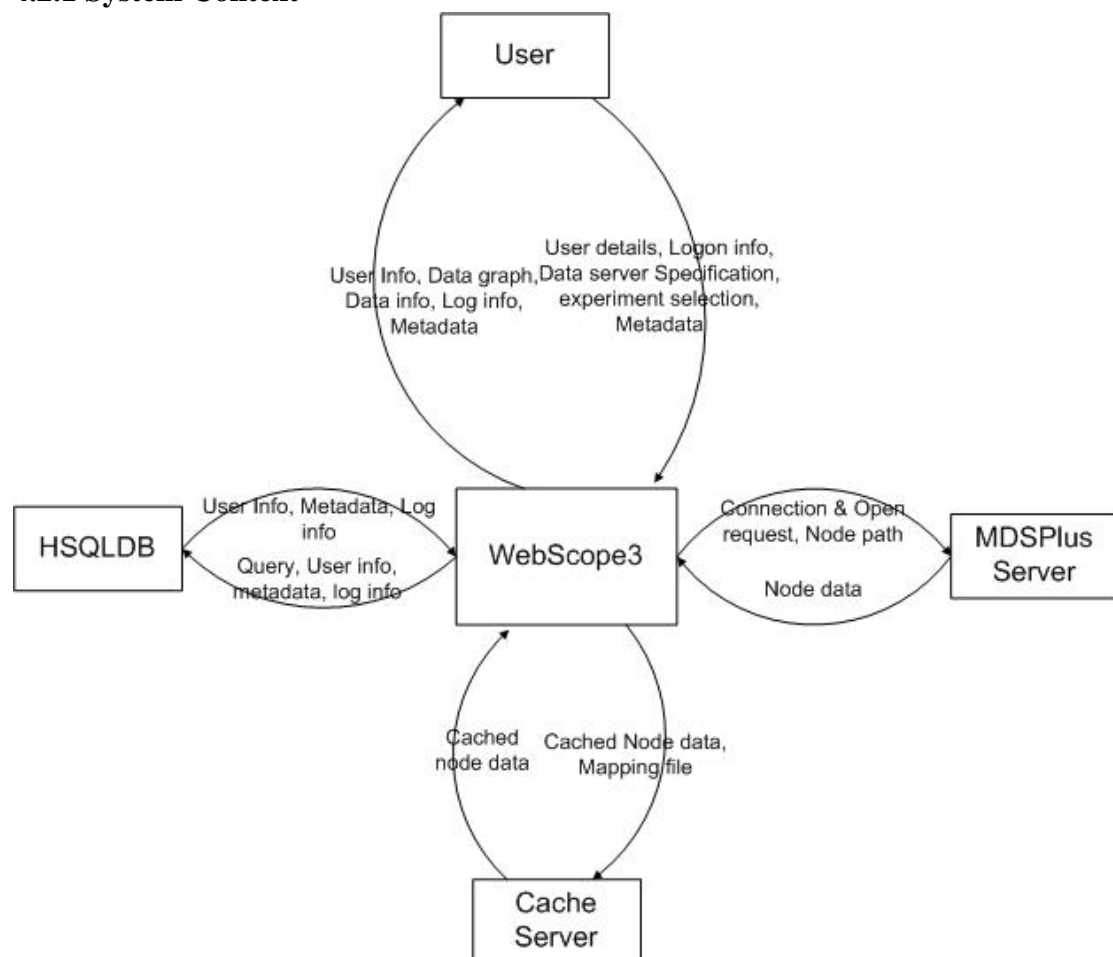


Figure 4.2 Context Diagram of “WebScope3”

There are four existing actors interacting with “WebScope3”.

- User:

1. A user inputs user login information (user email and password) at the login screen. After verification, the system sends back user information to the user.

2. A static metadata table will be shown to the user. The user specifies which data server to connect with server name, server port, experiment name and shot name. Then, the system sends data of the specified experiment, in the form of tree.

3. The user clicks to select a tree node that he/she wants to view. The system sends two type of expression of the tree node. One is the graph which is converted from binary code of the specified tree node. The other is data information such as numbers of point in the tree node and the text file which enables the user to download and analyze the data. The user can also contribute static/dynamic metadata to the database. In addition, the user can query metadata with different keywords.

- MDSPlus:

The WebScope3 system needs to connect to a MDSPlus server to work. A particular experiment will be opened and a node path is sent to the server to request specific data. The MDSPlus server will send back the data required.

- Caching Server

1. The WebScope3 system sends data to the caching server in two possible formats: binary file and text file. The caching server stores the data.

2. When some data is required, the system sends a request to caching server to retrieve cached data. The caching server then returns the data to the system.

- HSQLDB

1. The WebScope3 system sends user information, metadata and log information to HSQLDB, where these kinds of information are saved.

2. When a certain kind of information is required, the WebScope3 system sends a request to the database to retrieve user information, metadata or log information. The database then returns the required information to the system.

4.2.2 Domains

The WebScope3 system consists of four main domains, which are described below:

Domain Name	Main Responsibility
WebScope3	Provides Ajax-based Web User Interface to users.
databaseAccessDomain	Provides an interface for the WebScope3 system to interact with the Hibernate mapping solution to read and write information from the HSQLDB database.
dataServerDomain	Provides an interface for the WebScope3 system to retrieve datasets from the MDSPlus server.
Graphics	Provides a series of classes to draw the data graph.
Web Server	Provides a container to run the WebScope3 system and the associated database.
Java	Provides the programming platform.

Table 4.2 Domain Description

The relationships between domains have been illustrated in the figure below (Figure 4.3):

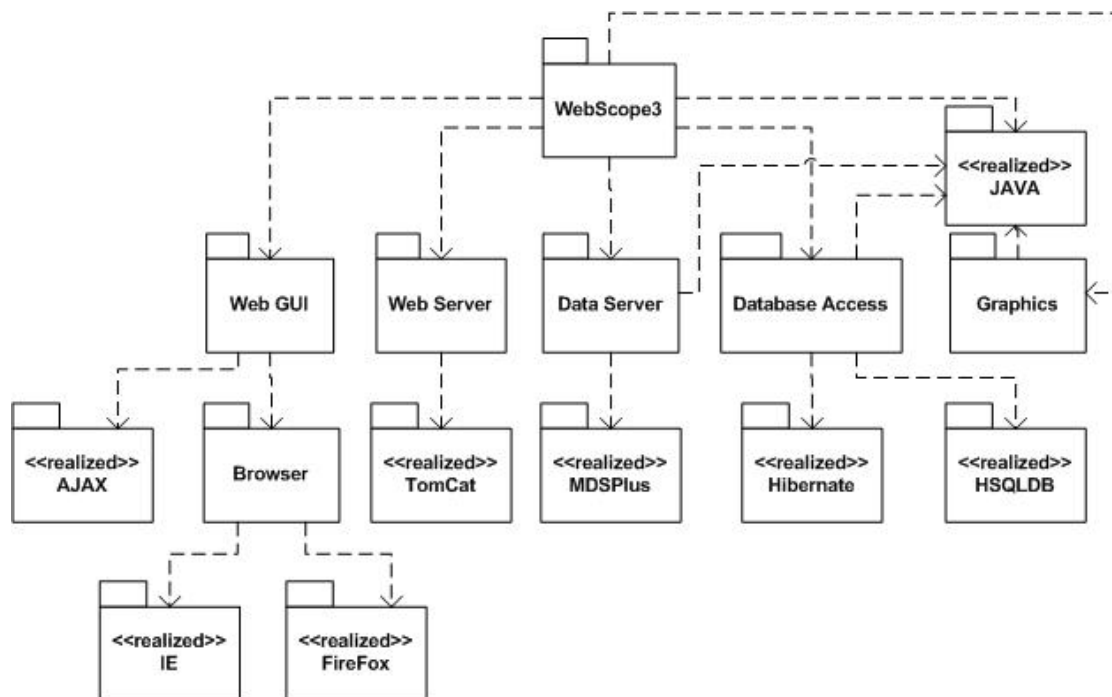


Figure 4.3 Domain Chart

4.3 Detailed Design

4.3.1 Communication between packages

The figure below (Figure 4.4) illustrates the communication between packages in the WebScope3 system:

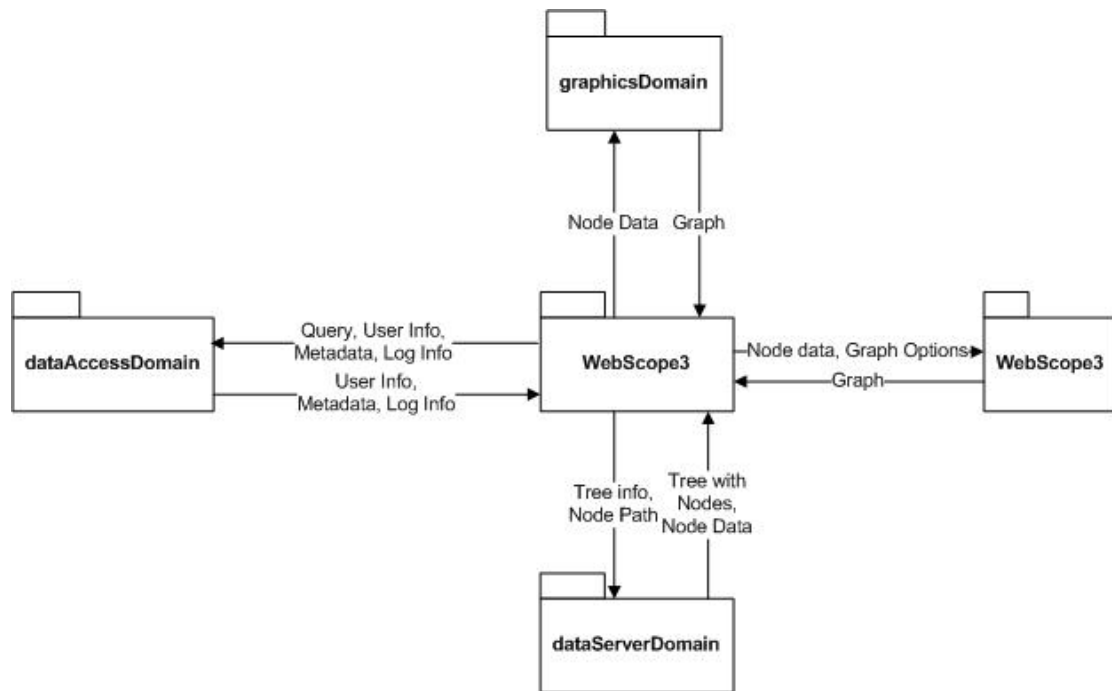


Figure 4.4 Communication between packages

4.3.2 Database Design

The database design for WebScope3 is based on the designs for Mr. Le Ma's WebScope and Mr. Zhongshan Tan's WebScope2. To meet the new requirements, I have combined their ideas and worked out a new database design. The major change I have made to their design is that the NODE and NODEPATH tables have been removed because of the new graph plotting method. The detailed database design for WebScope3 is shown below:

Table Name	Explanation
USERS	Stores the personal details of users.
LOGINFO	Keeps a log of user activities.
METADATAINFO	Stores the static metadata information contributed by users. Metadata information is related to the experiment data to which the metadata information is added.
DYNAMICTABLEINFO	Keeps the dynamic metadata table information. Note that only the table names, column numbers and column names are stored. The column values are stored in the dynamic tables. This design is to facilitate the dynamic creation of java classes and Hibernate mapping files.

Table 4.1 Database Design

All interactions with the database are handled with the help of java object relational/mapping solution “Hibernate”. Java persistent classes were created to represent the database tables and the details of those classes were entered into the Hibernate mapping file (“Mapping.hbm.xml”). Hibernate dynamically creates the tables corresponding to the information provided in the mapping file, and performs various actions. The Appendix C shows the mapping file used by Hibernate package.

All the tables in the database use an Id field as the primary key and the values to these fields are dynamically generated by the Hibernate package.

The METADATAINFO table and LOGINFO table both have foreign keys referring to the USERS table.

4.3.3 WebScope3 Domain

The WebScope3 domain provides the Ajax-based Web User Interface to users. The structure of this domain is exactly the same as that of the WebScope2 domain in the WebScope2 system developed by Mr. Zhongshan Tan. However, the PlotDataServer class has been greatly changed to provide the newly required features.

The figure below (Figure 4.5) illustrates the classes in the WebScope3 domain and their relationships:

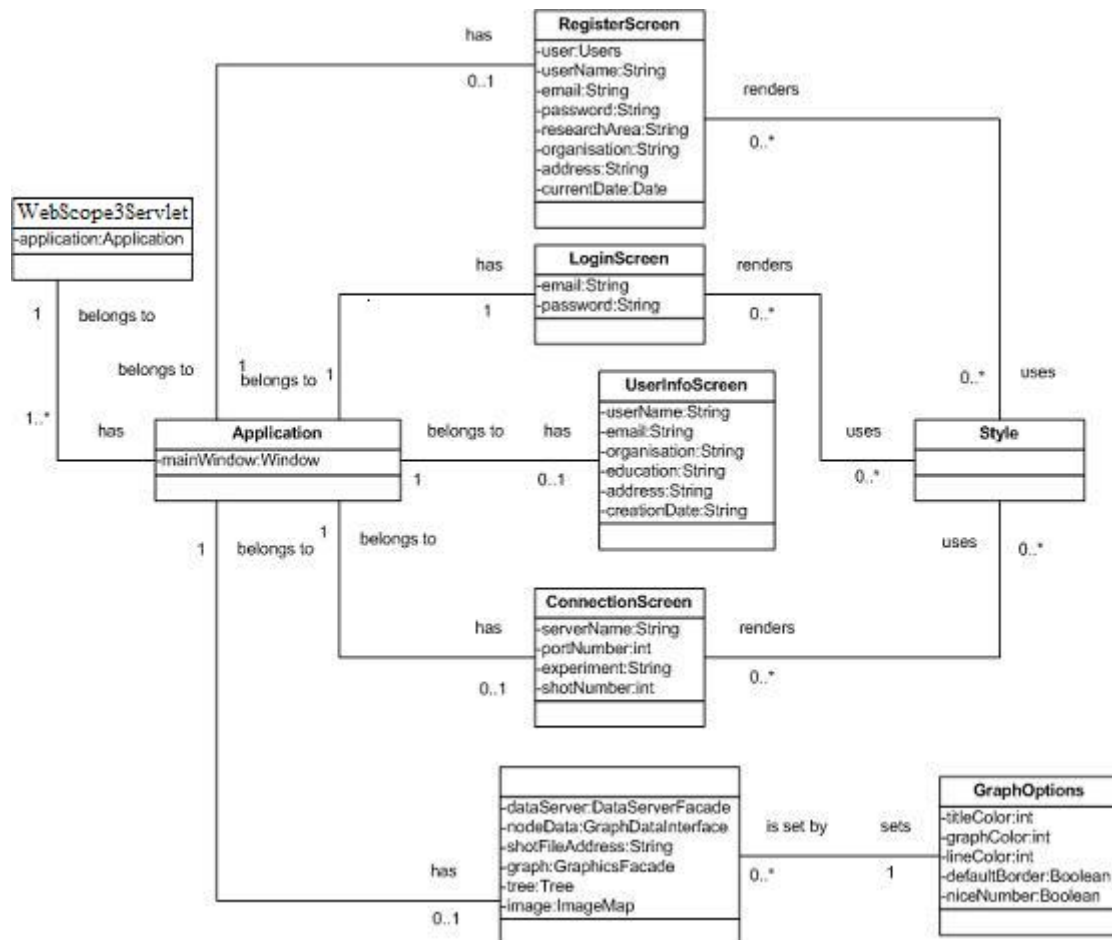


Figure 4.5 WebScope3 domain

The state machine of the WebScope3 domain is shown in the figure below (Figure 4.6):

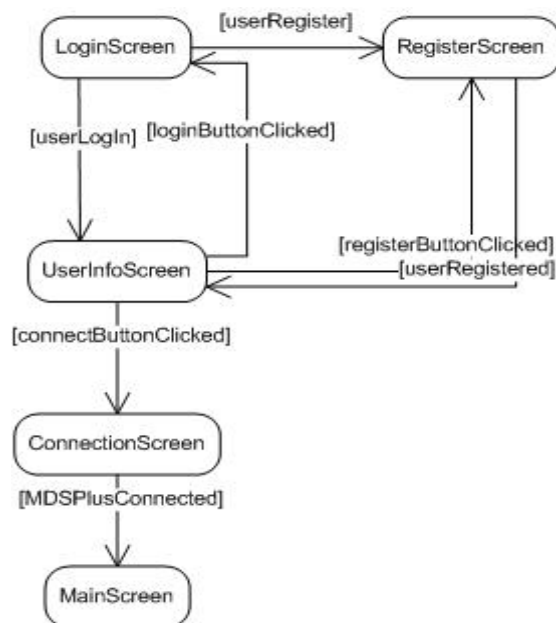


Figure 4.6 webscope3 Domain State Chart

4.3.4 dataAccessDomain Domain

In addition to the original User class in the dataAccessDomain domain of the WebScope2 system developed by Mr. Zhongshan Tan, 3 extra static classes, the DynamicTableInfo class, the LogInfo class, and the MetaDataInfo class, have been added.

The DynamicTableInfo class maps the DynamicTables table in the HSQLDB database via Hibernate. It stores the names of the dynamic tables, their number of columns and the names of the columns. However, the values of the columns are not stored here, but in the dynamically generated tables in the database. Therefore, the DynamicTableInfo class can be regarded as the manager or record of the dynamic tables.

The LogInfo class maps the LogInfo table in the HSQLDB database via Hibernate. It keeps a track of the access log of every user. Therefore, it has a Users foreign key.

The MetaDataInfo class maps the MetaDataInfo table in the HSQLDB database via Hibernate. It stores the static metadata information that users contribute to the database. It also has a Users foreign key together with a LogInfo foreign key.

The figure below (Figure 4.7) illustrates the classes in the dataAccessDomain domain and their relationships:

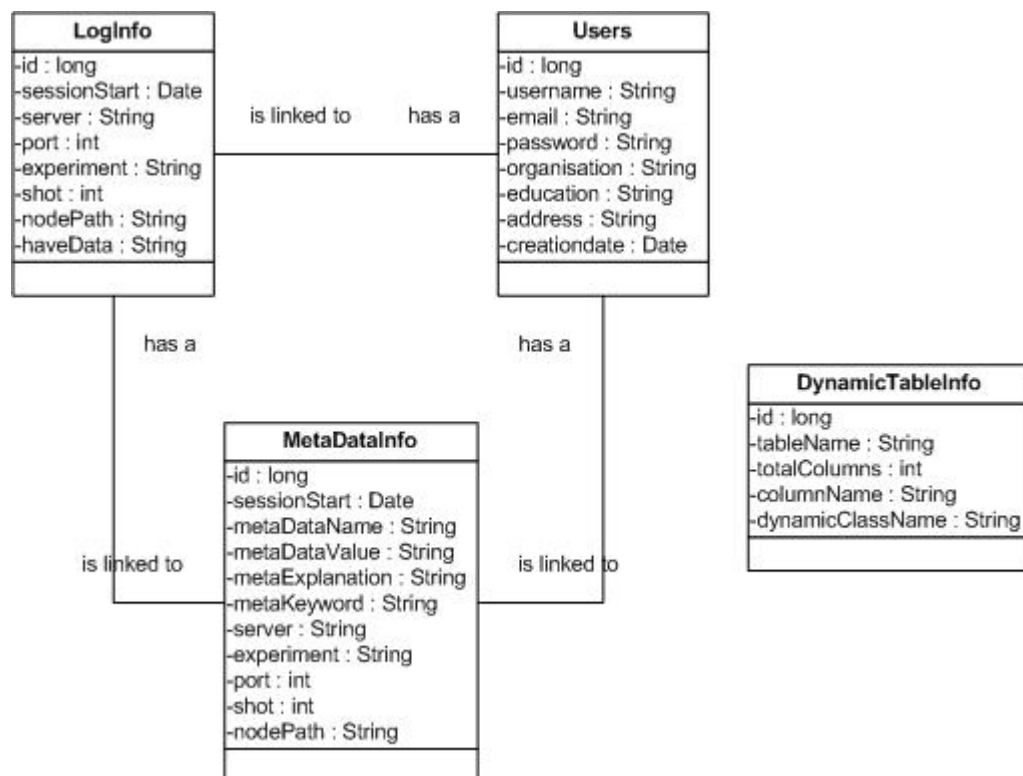


Figure 4.7 dataAccessDomain Domain

Moreover, classes are dynamically added to this domain at runtime if dynamic tables are contributed. The names of the classes and their attributes are consistent with the information stored in the DynamicTables table in HSQLDB database.

4.3.5 dataServerDomain Domain

dataServerDomain domain here is inherited from the dataServerDomain domain of the WebScope2 system. It provides an interface for the WebScope3 system to retrieve datasets from the MDSPlus server.

Most classes in DataServer domain in WebScope2 system reuse dataServerDomain in EScope4 which is developed by Dr. Henry Gardner. Moreover, some additional classes are included in the domain. They are DataServerCache class, CacheThread class and TextFileMaker class.

- DataServerCache class is the class for caching data into local server rather other retrieving data from MDSPlus server every time.
- CacheThread class is the class in which a respective thread runs, which is on the purpose of enhance the performance and usability. Data can be downloaded while the system plot graph to user.
- TextFileMaker class converts binary file to text file, which enables users to download text file format of data to view and analysis off-line. Also, it has the caching function; any downloaded text file is cached for the later use.

The figure below (Figure 4.8) illustrates the classes in the dataServerDomain domain and their relationships:

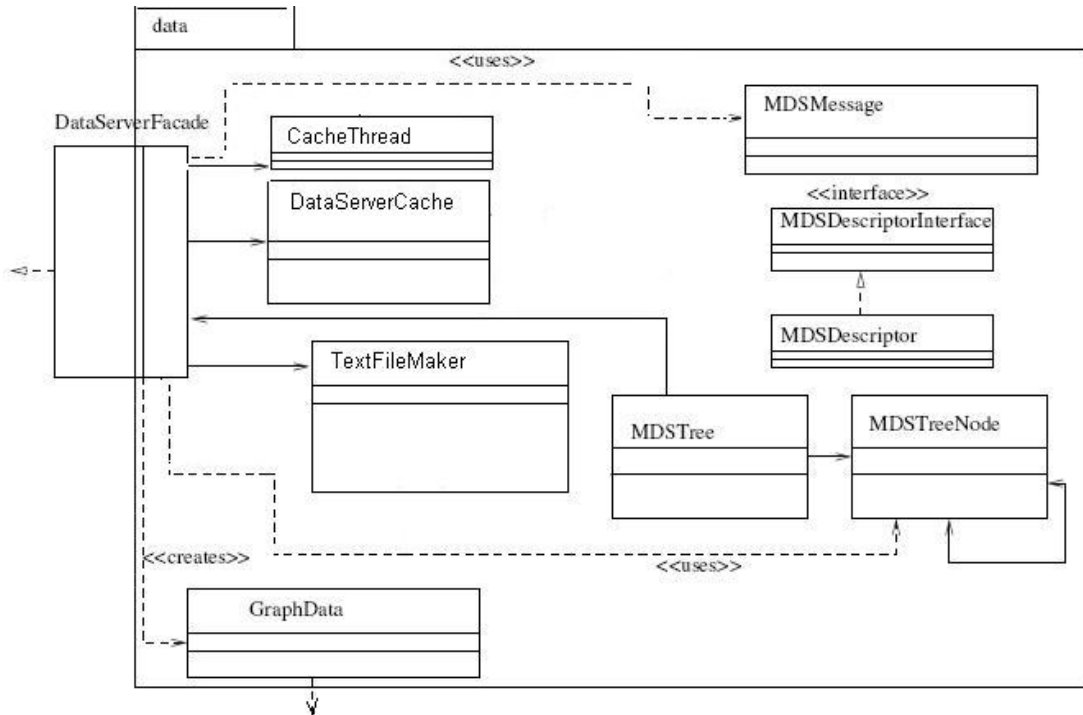


Figure 4.8 dataServerDomain Domain [1]

4.3.6 graphicsDomain Domain

The graphicsDomain Domain is currently inherited from the graphicsDomain domain of the WebScope2 system. However, this domain is being re-written. It provides a series of classes to draw the data graph.

The figure below (Figure 4.9) illustrates the classes in the current Graphics domain and their relationships:

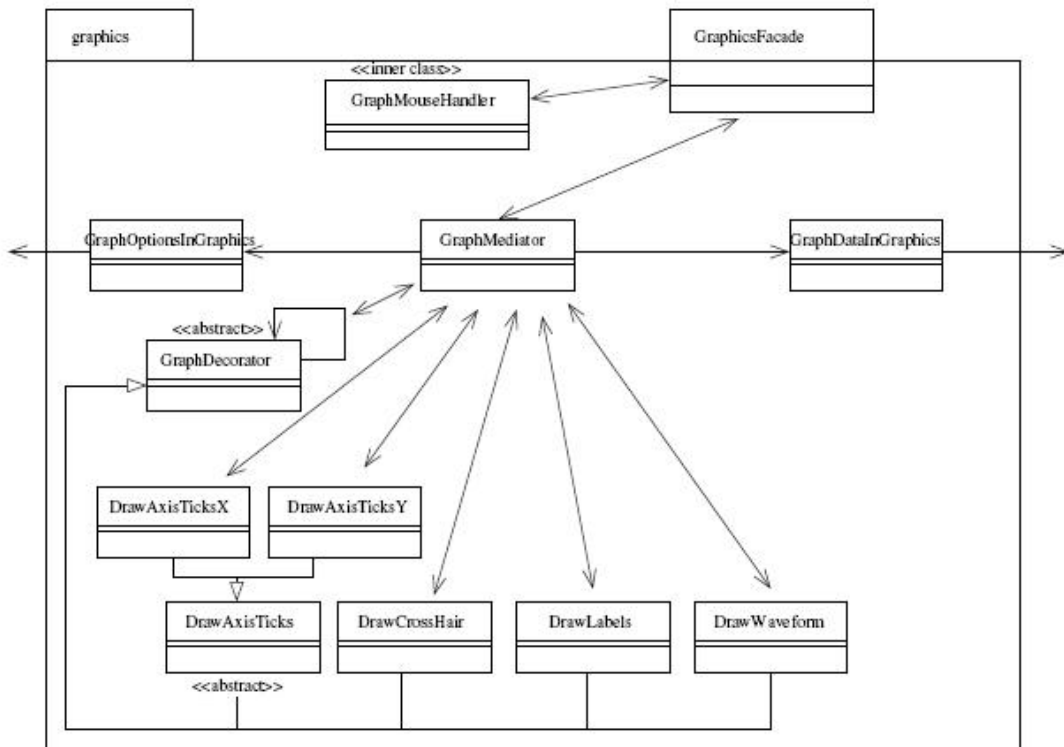


Figure 4.9 graphicsDomain Domain [1]

4.3.7 Mapping Domain

In addition to the two mapping files hibernate.cfg.xml and UserInfoMapping.hbm.xml which are inherited from the WebScope2 system, one more static mapping file MetadataStuffMapping.hbm.xml is added. It maps the MetaDataInfo class, the LogInfo class and the DynamicTableInfo class in dataAccessDomain domain to the corresponding HSQLDB table, which are the MetaDataInfo table, the LogInfo table and the DynamicTables table respectively.

Moreover, more mapping files are dynamically added to this domain at runtime if dynamic tables are contributed to the database.

4.3.8 Applet Domain

This domain is responsible for plotting the dataset in the Java Applet mode of the WebScope3 system. It's not loaded unless the Java Applet mode is enabled. It consists of four classes: the GraphOptions class, the GraphPanel class, the RetrieveMainFrame class, and the SimpleTableModel class. The figure below (Figure 4.10) illustrates the relationship between the 4 classes.

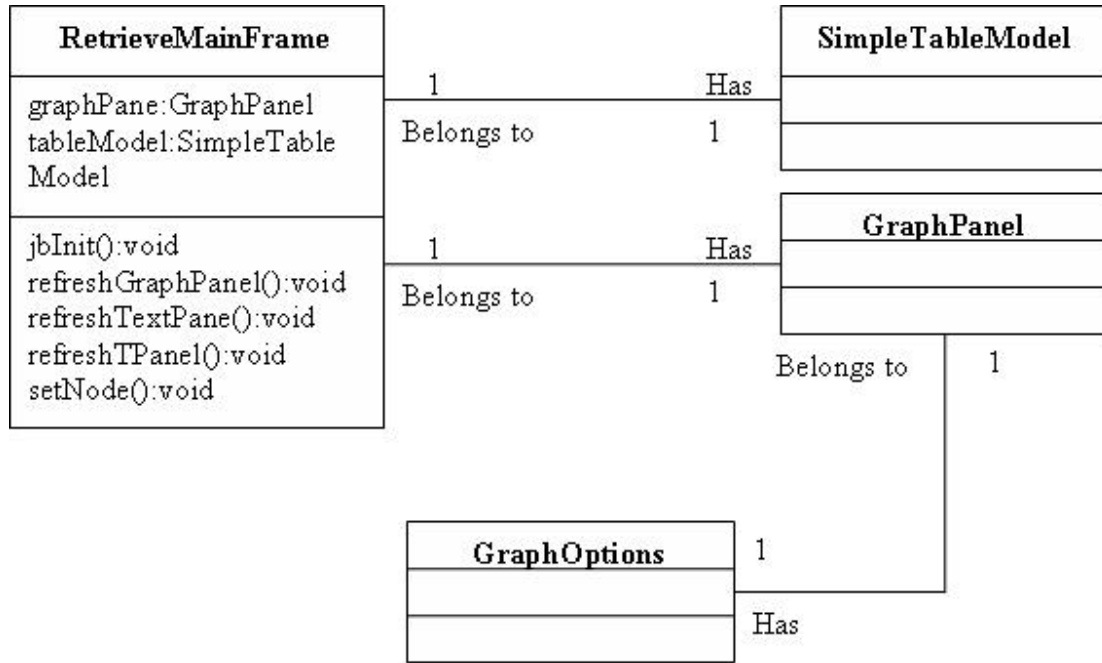


Figure 4.10 Applet Domain

5. Implementation

Since the number of technologies used for the development of “Web Scope” is high, the whole Implementation phase was divided into six sub phases. The details of these sub phases are given below:

Sub Phase Name	Activities Involved
1. Migrating from Echo2 to Cooe	Implement the WebScope3 system using the Cooe framework instead of the Echo2 framework.
2. Adding MetaDataInfo, LogInfo, and DynamicTableInfo tables	Add the classes MetaDataInfo, LogInfo, and DynamicTableInfo to the databaseAccessDomain of the WebScope2 system, and add the new mapping file MetadataStuffMapping.hbm.xml which provides the related Hibernate mapping information.
3. Tracking user activities and displaying real-time user activity log	Keep a record of the nodes a user has accessed and show this record to the user in the form of a LogInfo table. The LogInfo table is refreshed every time a tree node is clicked on.
4. Enabling users to contribute static metadata to the HSQLDB database. Displaying static metadata	Static metadata includes four parts: Metadata name, Metadata value, Metadata Explanation, Metadata keyword. A static metadata is bound to a tree node and the user who has contributed to the metadata, which means that the metadata table includes the tree node information and the user name. Two types of static metadata tables are displayed to users in the main screen of the WebScope3 system, the Standard Metadata Table and the Complete Metadata Table. A simplified metadata table is displayed to users on the user information screen, too.
5. Enabling users to query static metadata by keywords	Users are able to query static metadata by different keywords. Metadata can be queried by server, by experiment, by shot, by node path, by metadata name, by metadata value, or by user.
6. Enabling users to contribute dynamic metadata to the HSQLDB database. Displaying dynamic metadata	Dynamic metadata does not have a standard format. Users can create their own tables and contribute them to the database. This is realized in the way below: <ol style="list-style-type: none"> 1. Generate Java files and add related Hibernate mapping files based on users' inputs. Table names, column numbers and column names are recorded in the form of a DynamicTableInfo table. 2. Compile the Java files to produce class files

	<p>3. Load the class files dynamically and contribute the dynamic tables to the database.</p> <p>4. Load the class files dynamically to retrieve and display the table information.</p>
7. Providing users with two graph displaying methods: Image Map and Applet	Users are able to choose which method can best meet their needs. As the Applet method is relatively slow to load and provides a more powerful way to display the graph, it is up to the users to decide which method to use.
8. Testing	<ul style="list-style-type: none"> ● Test the software by various possible use ● Test the software under all possible conditions (such as the MDSPlus is not working, the connection is interrupted while interacting)

Table 5.1 Sub-phases and activities involved

Generally speaking, the whole implementation process went smoothly. However, I did meet some problems during this process.

First of all, migrating from the existing Echo2 framework to the Cooe framework meant that I needed to update every class and change the “include” information. In addition, I had to change the form files and resource information, or the WebScope3 system would simply not load without throwing out any exception. This was a bit confusing as I got no information from debugging. Fortunately, I came across the form and resource files by accident and the issue was resolved after some modification.

Secondly, adding the MetaDataInfo, LogInfo, and DynamicTableInfo tables to the WebScope3 system is not as simple as copying Le Ma’s table files to Zhongshan Tan’s source folder. As some of the variables and information in Le Ma’s tables were never used in Zhongshan Tan’s version, I made some necessary changes to the tables to fit them to the WebScope3 system.

The dynamic metadata part of the WebScope3 system took me a lot of time as this feature was not really implemented in any of the previous version of WebScope. I had to completely re-write this feature in Zhongshan’s version. During this process, the following problems puzzled me a lot:

- Hibernate does not load a single mapping file twice even if you tell it to do this. To workaround this problem, I had to make a mapping file for every dynamic table.
- We should save an instance of a class object using the *save* method of the *Session* class instead of saving the class object directly.

In addition, it is not difficult to write a Java Applet and make it plot the graph for a dataset. However, it took me quite a long time to send objects from the WebScope3 system to the applet. The problem was that the HTTPConnection provided by the getActiveConnection() method of the WebRenderServlet class in the Cooe package did not work as expected. Whenever an object was written into the outputstream retrieved from this connection, the whole system froze. After spending about two weeks testing the various parameters and combinations of other methods, I finally worked around this issue by using a socket connection instead.

I also needed to take into account the cooperation between the two graph-displaying modes (The Image Map mode and the JAVA Applet mode). This is because the most currently selected node should be plotted on the screen in the proper mode when a new graph node is selected from the tree or users click on the radio button to switch between modes. To make this work, I chose to update the data for both modes in the background whenever a new graph node is clicked.

A lot of testing was involved as almost every possible exception had been thrown out during the developing process of the dynamic metadata feature. For details, please refer to Chapter 7.

6. Cognitive Walkthrough and Heuristic Evaluation

6.1 Cognitive Walkthrough

“The cognitive walkthrough is a formalized way of imagining people’s thoughts and actions when they use an interface for the first time.” [8] The cognitive walkthrough includes three major steps:

- Set up tasks and detailed task sequence
- Go through the tasks and check the points raised by Clayton Lewis and John Rieman [8]
- Conclude the walkthrough.

The purpose of this walkthrough is to ensure the usability of the features in a program and to find potential usability problems which may prevent novice users from using the program properly.

The WebScope3 system gets a very high mark in the cognitive walkthrough. Therefore, we can conclude that our system is easy to use from the perspective of a first-time user. This is because I paid quite a lot of attention to web page layout when designing the pages and tried to put everything where they were supposed to be. Dr. Henry Gardner also provided me with many valuable suggestions and advice, which helped a lot in making the WebScope3 application easy to use and understand.

The detailed information about this cognitive walkthrough has been included in Appendix A (See P58/59).

6.2 Heuristic Evaluation

Heuristic evaluation [22] is a usability engineering method for finding the usability problems in a user interface design so that they can be attended to as part of an iterative design process.

In general, more than one evaluator is required in the heuristic evaluation process as the usability problems that a single individual can find are quite limited. However, as we do not have enough hands to perform a comprehensive heuristic evaluation, I myself acted as the only evaluator.

In this evaluation, 10 heuristics are used to check the usability. The following problems have been found:

- Users do not have full control and freedom in some part of the system (See Appendix B, Page 63)
- No help documentation has been provided to users (See Appendix B, Page 67).

The first problem has been largely resolved. However, the second problem remains unresolved due to the limitation of time. I've listed this problem in the Future Work section (Chapter 9).

In comparison with the cognitive walkthrough, I believe that a heuristic evaluation can help to find more usability problems because it takes both expert users and novice users into account. In addition, as no special use cases are used, every possible operations are performed, which means that a heuristic evaluation can cover more parts of the system.

For details, please refer to Appendix B.

7. Testing

Testing is always important as it can help to ensure the functionality and usability of a system. Three testing methods are carried out in this case, which are unit test, module test and acceptance test respectively.

7.1 Unit Test

Unit test is a method used to check whether or not the individual units of a system are working properly. A unit is the smallest testable part of a system, which is a method in the WebScope3 system.

Unit tests were carried out whenever a new method was added to the WebScope3 system. All possible types of inputs to a method were tested to ensure that the method was able to return the expected outputs in all situations.

Thanks to the continuous unit test on the *thousands of methods*, the system can work fine without any visible problems now.

7.2 Module Test

Module test is a method used to check whether or not a collection of individual units of a system can work together properly. The collection of units can be a class or a package. Here, I describe a module test which was preferred on packages.

The WebScope3 system consists of 8 packages, which are listed below:

- The databaseAccessDomain package
- The mapping package
- The dataServerDomain package
- The graphicsDomain package
- The sharedDataInterfaces package
- The sharedInterfaces package
- The webscope3 package
- The applet package

Among the packages above, the sharedDataInterfaces package and the sharedInterfaces package are used for design pattern purpose and only provide abstract classes and interfaces. Therefore, there is no need to perform module test on

these two packages. In addition, the mapping package does not include any java files. It's used to work with the databaseAccessDomain package to communicate with Hibernate. Therefore, these two packages are tested as a single package.

The details of the module test have been included in the table below (Table 7.1~Table 7.5):

databaseAccessDomain & mapping Test Plan	Test Result
<p>Plan 1: add a Users table</p> <p>Step 1: Instantiate the Users class in dataAccessDomian; Step 2: Evaluate the Users instance and then save it back to the Users table in the database.</p> <p>Expected output: The Users instance should appear in the database.</p>	<p>After performing the 2 steps and querying the Users table in the database, it's confirmed that the Users instance has been successfully saved in the database.</p> <p>Result: Test passed</p>
<p>Plan 2: add a MetadataInfo table</p> <p>Step 1: Instantiate the MetadataInfo class in dataAccessDomian; Step 2: Evaluate the MetadataInfo instance and then save it back to the MetadataInfo table in the database.</p> <p>Expected output: The MetadataInfo instance should appear in the database.</p>	<p>After performing the 2 steps and querying the MetadataInfo table in the database, it's confirmed that the MetadataInfo instance has been successfully saved in the database.</p> <p>Result: Test passed</p>
<p>Plan 3: add a LogInfo table</p> <p>Step 1: Instantiate the LogInfo class in dataAccessDomian; Step 2: Evaluate the LogInfo instance and then save it back to the LogInfo table in the database.</p> <p>Expected output: The LogInfo instance should appear in the database.</p>	<p>After performing the 2 steps and querying the LogInfo table in the database, it's confirmed that the LogInfo instance has been successfully saved in the database.</p> <p>Result: Test passed</p>
<p>Plan 4: add a DynamicTableInfo table</p>	<p>After performing the 2 steps and querying the DynamicTableInfo table</p>

<p>Step 1: Instantiate the DynamicTableInfo class in dataAccessDomain;</p> <p>Step 2: Evaluate the DynamicTableInfo instance and then save it back to the DynamicTableInfo table in the database.</p> <p>Expected output: The DynamicTableInfo instance should appear in the database.</p>	<p>in the database, it's confirmed that the DynamicTableInfo instance has been successfully saved in the database.</p> <p>Result: Test passed</p>
---	--

Table 7.1 databaseAccessDomain Module Test

dataServerDomain Test Plan	Test Result
<p>Plan 1: Retrieve a data tree from the MDSPlus server</p> <p>Step 1: Instantiate the dataServerFacade class.</p> <p>Step 2: Retrieve a data tree from the MDSPlus server.</p> <p>Step 3: Instantiate a Cooee tree and show the data tree in the main screen of the WebScope3 system.</p> <p>Expected output: A data tree should appear in the main screen and the names of the tree nodes should be exactly the same as those in the MDSPlus server.</p>	<p>After performing the 3 steps, a data tree is shown in the left pane of the main screen. The names of the tree nodes are consistent with those in the MDSPlus server.</p> <p>Result: Test passed</p>
<p>Plan 2: Retrieve the data of a tree node from the MDSPlus Server</p> <p>Step 1: Instantiate the dataServerFacade class.</p> <p>Step 2: Retrieve the data of a tree node from the MDSPlus server.</p> <p>Step 3: Plot the data in the graph pane of the main screen.</p> <p>Expected output: A correct graph should be plotted which illustrates the retrieved tree node.</p>	<p>After performing the 3 steps, a graph is plotted in the graph pane of the main screen and the graph is correct.</p> <p>Result: Test passed</p>

Table 7.2 dataServerDomain Module Test

graphicsDomain Test Plan	Test Result
<p>Plan: Plot and return a graph image</p> <p>Step 1: Instantiate the GraphicsFacade class Step 2: Retrieve a tree node from the MDSPlus server and set the proper graph data using the GraphicsFacade instance. Step 3: Plot and return the graph image.</p> <p>Expected output: The proper graph should be returned and shown in the graph pane of the main screen.</p>	<p>After performing the 3 steps, the expected graph appears in the graph pane of the main screen.</p> <p>Result: Test passed</p>

Table 7.3 graphicsDomain Module Test

applet Test Plan	Test Result
<p>Plan: Load the applet and plot a tree node</p> <p>Step 1: Load the Java Applet in the main screen of the WebScope3 system.</p> <p>Step 2: Retrieve a tree node from the MDSPlus server and transfer the data to the applet using a Socket connection.</p> <p>Step 3: Plot the tree node.</p> <p>Expected output: The applet should get the data and plot the proper tree node.</p>	<p>After performing the 3 steps, the applet is loaded correctly and the proper tree node is plotted.</p> <p>Result: Test passed</p>

Table 7.4 applet Module Test

webscope3 Test Plan	Test Result
<p>Testing the webscope3 domain includes too many tasks, most of which are duplicated with those mentioned above. Therefore, the detailed process of testing this domain has been omitted.</p>	<p>After various tests, the webscope3 domain is proved to be working properly.</p>

Table 7.5 webscope3 Module Test

7.3 Acceptance Test

Acceptance test is a method used to check whether or not a system is acceptable to users and all the required features have been implemented. As I have a very clear requirement list, it's relatively easy to perform this test (Thanks to Dr. Henry Gardner!). The details have been included in the table below (Table 7.2):

No.	Req. Name	Priority	Implemented
R1	Migrating from the existing Echo2 framework to Karora Cooee	High	Yes
R2	Using Web Browser	High	Yes
R3	Using Hibernate	High	Yes
R4	Plotting Graph	High	Yes
R5	Data download	Medium	Yes
R6	Platform Independence	High	Yes
R7	Getting and displaying user details	High	Yes
R8	Tracking user activities	Medium	Yes
R9	Contributing static metadata	High	Yes
R10	Displaying static metadata	High	Yes
R11	Querying static metadata	Medium	Yes
R12	Contributing dynamic metadata	High	Yes
R13	Displaying dynamic metadata	High	Yes
R14	Providing two types of graph-displaying methods	High	Yes

Table 7.2 Acceptance Test

8. Design Patterns

“In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design.” [21]

A design pattern is not a piece of code or an actual implementation of functions, but an abstract description or template used to improve the modularization of software. Although programs can work fine without implementing any design pattern, it is important to introduce design patterns to every software project because they can make the structure of software clearer and make it easier for programmers to maintain or update the software in the future.

However, this does not mean that the more design patterns we use, the better our system will be. We should analyze the software and find out the purpose of each domain first before deciding which design patterns we will apply.

Here I will analyze the packages/domains of the WebScope3 system individually and describe which design patterns should be used.

8.1 WebScope3 Domain

The WebScope3 domain is responsible for user interaction. It's the bridge between users and the database. Please refer to 4.3.3 for details of the structure of this domain. Most of the classes included in this domain are related to the various screens of the GUI, such as the ConnectionScreen class, the LoginScreen Class, the PlotDataScreen class, the RegisterScreen class, and the UserInfoScreen class. The Application class and the WebScope3Servlet class are required by the Cooee framework, and the structure of the current layout is compulsory. I decide to leave the WebScope3 Domain as it is, otherwise the Cooee framework will not be able to work properly.

8.2 dataAccessDomain Domain

The dataAccessDomain domain is responsible for mapping the tables in the relational database via Hibernate. For details of this domain, please refer to 4.3.4. As only four classes are included in this domain and all of them need to communicate with classes in other domains, applying design patterns to this domain may make it less efficient. Therefore, no design patterns have been used on the dataAccessDomain domain.

8.3 dataServerDomain Domain

The dataServerDomain domain is responsible for the communication between the WebScope3 system and the MDSPlus server. Therefore, it is good to make this domain a black box, which means that the only visible parts of this domain should be two interfaces, one for the WebScope3 system and one for the MDSPlus server, from the perspective of the other domains. For this purpose, the Façade design pattern has been applied on this domain. The image below (Figure 8.1) illustrates the structure of the Façade pattern:

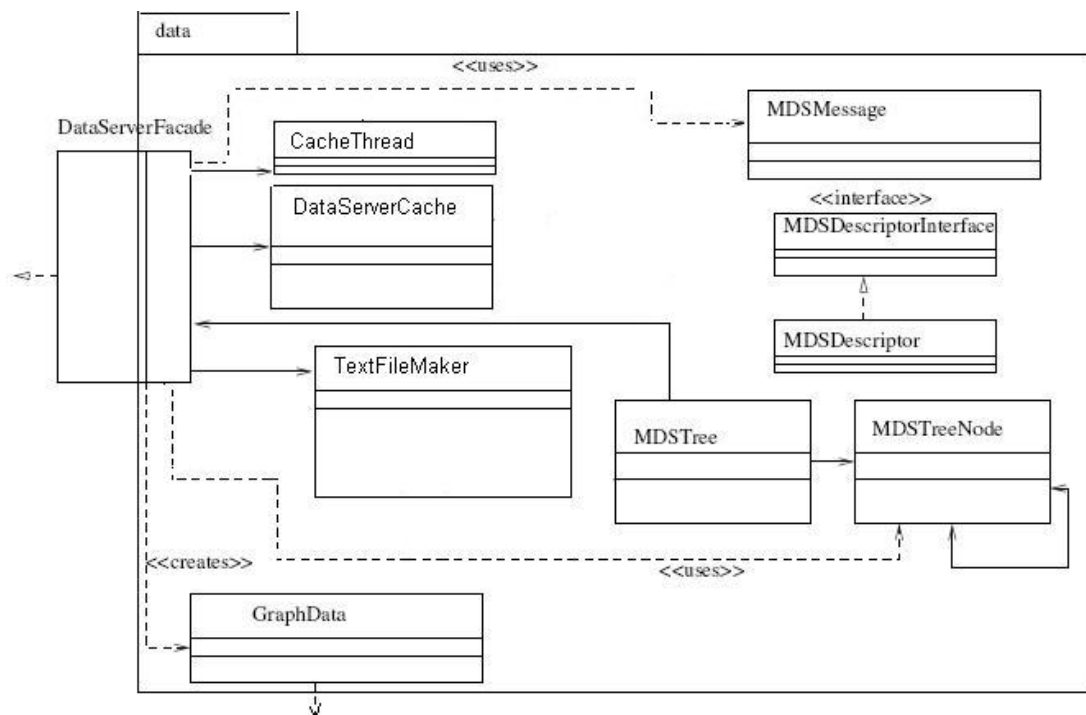


Figure 8.1 Façade Pattern [1]

To implement the Façade design pattern, we should complete the following tasks:

- Create a Façade class for the dataServerDomain domain
- Associate the Façade class with an interface, which contains the signatures of all the methods which represent the services the dataServerDomain domain can provide
- The dataServerDomain domain can only be visited or called via the Façade class. All the other classes in this domain are supposed to be invisible to other domains.

In order to meet the requirements above, the DataServerFacade class was created and it works as the Façade class. This class implements the interface DataServerFacadeInterface, which is located in the sharedInterfaces package.

The DataServerFacadeInterface interface contains the methods below:

- public void connect(String serverAddrCPort) throws IOException;
- public void disconnect() throws IOException;
- public void open(String experiment, int shot) throws IOException;
- public void close() throws IOException;
- public void constructTree(Tree tree) throws IOException;
- public void getDataDownload(ContentPane contentPane, GraphDataInterface graphData, String nodePath);
- public String getSPES();
- public boolean isConnected();
- public boolean isOpen();
- public String getExperiment();
- public int getShot();
- public GraphDataInterface getPlotData(String path);

The methods above are all the services the dataServerDomain domain provides, and they are all realized in the DataServerFacade class. Every access to the dataServerDomain domain is attempted via the DataServerFacade class now.

8.4 graphicsDomain Domain

The graphicsDomain domain is responsible for plotting the dataset in the ImageMap mode. For details of this domain, please refer to 4.3.6. The interactions between this domain and the other domains are relatively simple: getting the dataset from other domains, and returning an image file which contains the data graph.

Therefore, the Façade design pattern is applied on this domain. The GraphicsFacade class works as the Façade class and it implements the GraphicsFacadeInterface interface, which contains the methods below:

- public void setGraphData(GraphDataInterface data);
- public AwtImageReference generateImage(int width, int height);
- public void setGraphOptions(GraphOptionsInterface gO);
- public void applyGraphOptions();

All the services the other domains can expect from the graphicsDomain domain are the 4 methods above. All the other parts of this domain are invisible to them.

According to the designing of the dataset graph layout, four different tasks need to be finished before a single image can be generated, which are as follows:

- Draw axis ticks
- Draw centered message
- Draw labels
- Draw wave form

To chain these classes together and make it easier to add new features in the future, the Decorator pattern is applied here.

In object-oriented programming, the decorator pattern is a design pattern that allows new/additional behavior to be added to an existing method of an object dynamically [21].

The abstract class GraphDecorator is created and it works as the abstract decorator. All the four classes DrawAxisTicks, DrawCentredMessage, DrawLabels, and DrawWaveform extend the GraphDecorator class. If some new features are added in the future, the only thing we need to do is to create a new class implementing the new feature and make it extend the GraphDecorator class.

In addition, as there are many parallel classes in this domain and the communication between classes are quite frequent, it is necessary for us to regulate the communication. Otherwise, this domain will be difficult to read or maintain. To address this potential problem, the Mediator pattern is applied, too.

The mediator pattern is a software design pattern that provides a unified interface to a set of interfaces in a subsystem [21].

The image below (Figure 8.2) illustrates the structure of the Decorator and Mediator patterns:

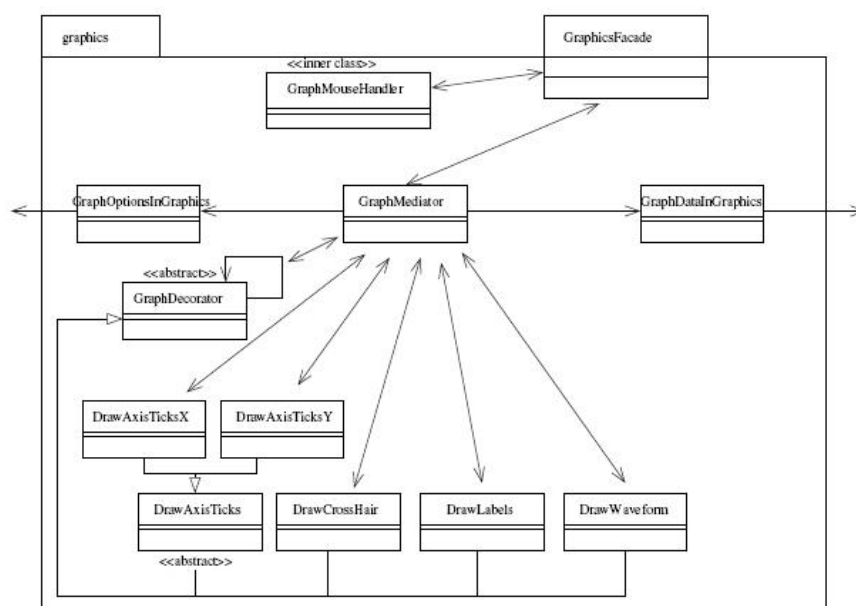


Figure 8.2 Decorator and Mediator Patterns [1]

The GraphMediator class is created and it works as the mediator. The communication among the classes in the graphicsDomain domain is realized via the GraphMediator class. This way, it's easier and more efficient to manage the communication between classes in this domain.

8.5 applet Domain

The applet domain is not loaded on the server side. It's only called when the Java Applet mode is activated on the client side, and this domain can only communicate with the other parts of the WebScope3 system via a socket connection. Therefore, it is not necessary for us to apply the Façade pattern on it.

In addition, as this domain is very simple, currently no other design patterns have been considered for this domain.

8.6 Comparison between the WebScope3 and EScope4 systems

Actually, the structure of the WebScope3 system is quite like that of the EScope4 system. The table below describes the corresponding parts of these two systems:

WebScope3	EScope4
dataServerDomain	dataServerDomain
graphicsDomain	graphicsDomain
Webscope3	guiDomain
databaseAccessDomain	
Mapping	
Applet	

Table 8.1 Comparison between WebScope3 and EScope4

The EScope4 system does not have counterparts for the databaseAccessDomain and mapping domains because it does not need to interact with Hibernate. The applet domain is not needed in this system, either.

Due to the similarity in the structures, almost the same design patterns are applied on

these two systems. However, there are still some differences. For example, the Façade pattern is applied on the guiDomain domain of the EScope4 system. However, no patterns are applied on its WebScope3 counterpart, which is the webscope3 domain. This is because of the requirement of the Cooee framework.

8.6.1 Transformation from EScope4 to WebScope3

When designing the WebScope3 system, the EScope4 system is used as the reference due to the reasons below:

- Both of the systems are used to explore nuclear fusion data.
- The structures of the two systems can be very similar due to the similarity in functionality.

The probable differences between these two systems are as follows:

- The EScope4 system uses a Java application as its GUI, while the GUI of the WebScope3 system is based on web pages.
- The WebScope4 system requires communicating with Hibernate and an additional cache server is used to ensure the transmission efficiency, while the EScope4 system does not need these.

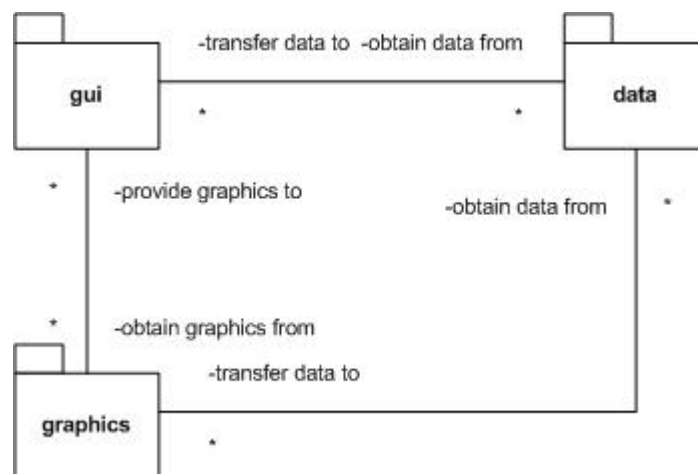


Figure 8.3 Structure of the EScope4 system

Therefore, the following steps are used to transform the structure of the EScope4 system to the WebScope3 system:

Step 1: Add some new domains

As the WebScope4 system requires communicating with Hibernate, a databaseAccess domain should be added to map the Hibernate tables. This domain only interacts with

the gui domain as it only collects input information from the GUI and return stored information as required.

In addition, a mapping domain should be added to store the corresponding Hibernate mapping files. This domain only interacts with the databaseAccess domain.

As we need to provide users with a Java applet method to display graph data, a Java applet domain is also necessary and it only interacts with the gui domain.

This way, the initial structure of the WebScope3 system should be like this (Figure 8.4):

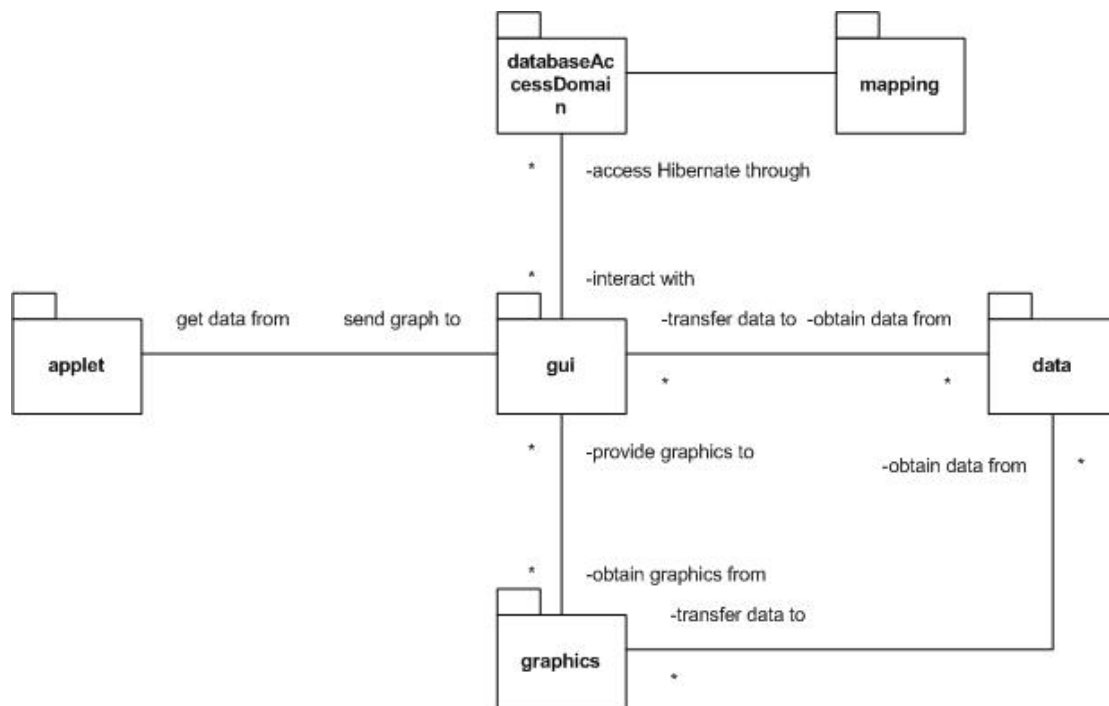


Figure 8.4 Initial structure of the WebScope3 system

Step 2: Modify the existing domains

The gui domain should be replaced with the webscope3 domain to provide users with a web-based GUI. Both the structure of this domain and the classes inside the domain are modified as per the requirements of the Cooe framework.

We can keep the structure of the graphics domain. However, almost all of the classes inside this domain should be modified as we are now providing graphics to a web application based on Cooe.

The dataServerDomain can be kept as it was. However, some new classes should be added to provide the additional cache server.

The new structure of the WebScope3 system should be like this (Figure 8.5):

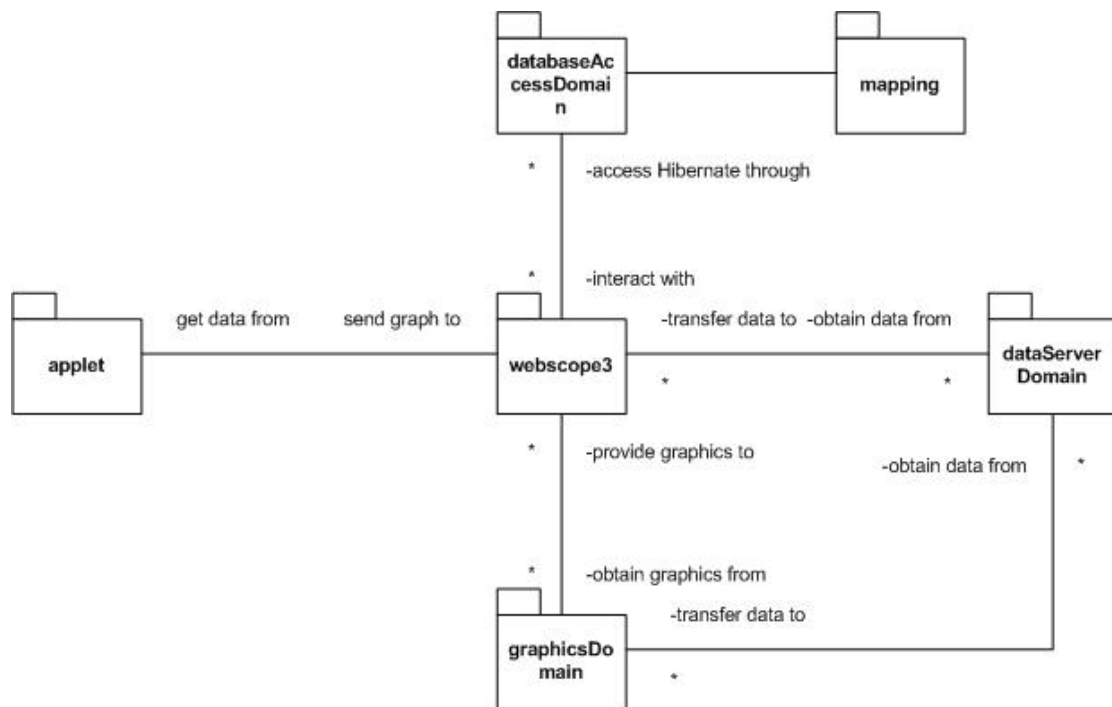


Figure 8.5 Updated structure of the WebScope3 system

And now, the transformation has been completed.

8.7 Conclusion

After analyzing the domains one by one and choosing the proper design patterns for each domain, the WebScope3 system is now more flexible, maintainable, efficient, and readable. Although it takes me some time to complete the design pattern analysis, it is worthwhile as this analysis can help to make the future work easier.

With the development of the WebScope system, more and more design patterns will be introduced in the future, which can help to guarantee the health and correctness of the system.

9. Future Work

The following work may be done in the future to make the WebScope3 system better:

- Currently, applet is used as an alternative graph-displaying method. However, as the Java Applet technology is not as popular as it was, we may introduce some more up-to-date technology, such as the JavaFX scripting technology. This way, the graph-displaying part of the WebScope3 system will be more beautiful, responsive and interactive.
- We may introduce the user priority concept to the WebScope3 system. Currently, every user possesses the same priority, which may not be ideal for some organizations. Some user management/user priority features may be added in the future.
- We are saving user details in plain text, which is definitely not safe. Some encrypting technology should be used to save this information in encrypted text to make the system more secure.
- Users cannot delete metadata (both static metadata and dynamic metadata) from the database. This feature could be added in next version of the WebScope system.
- The HSQLDB database is not efficient enough. Actually, it does not really save tables in the database, but keeps a list of scripts so that tables are “written” to the database every time the HSQLDB database starts up. Apparently, with the increasing of tables in the database, it will be slower and slower for the HSQLDB to start up. Some other database should be used to avoid this from happening.
- The WebScope3 system is not yet as powerful as the Escope system. Therefore, some more work should be done to improve the functionality, such as multiple graphs.
- Some help topics may be added to the WebScope system.

10. Summary of Contributions and Conclusion

10.1 Summary of Contributions

The following contributions have been made to the WebScope systems:

- I have combined the WebScope1 and WebScope2 systems, which involved adding the following features to the AJAX-based WebScope2 system:
 - Static metadata contribution
 - Static metadata displaying
 - User activity log
- I have migrated from the Echo2 framework to the Cooe framework.
- I have added the new features below: (See Chapter 5)
 - Metadata Query
 - Users can now switch between the Image Map and the Java Applet graph-display modes
 - A working dynamic metadata contribution/displaying feature
- I have studied the JavaFX script and concluded that it does not quite fit the requirements. Therefore, we need to continue to use Applet. (See Appendix D)
- A Cognitive Walkthrough and a Heuristic Evaluation have been performed to ensure the usability of the WebScope3 system. (See Chapter 6)
- Unit tests, module tests and acceptance tests have been performed to ensure the functionality and availability of the WebScope3 system. (See Chapter 7)
- The metadata query and metadata displaying table features have also been added to the older Java Servlet-based WebScope1 system.

In summary, WebScope3 provides a web-based interface to a fusion eScience data grid which includes dynamic metadata. As well as providing new functionality for the support of distributed exploration and management of data from nuclear fusion experiments, WebScope3 can serve as a prototype “Grid Portal” for other eScience applications.

10.2 Personal Statement

During the developing process of the WebScope3 system, I have met various problems and learned many useful techniques.

First of all, I have learned how to efficiently understand the code of others. To achieve this, it is better for me to ignore the details at the very beginning and get a general idea of what each class is doing.

Secondly, I have learned how to get useful information from all available resources. Whenever I'm stuck, I am able to get help from my supervisors and the Internet. Asking for help is never a shame.

I have also realized the importance of making plans and keeping a record of actual progress, which can help me clearly know what I'm doing and what I should do.

11. Reference

- [1] Henry Gardner, Gabriele Manduchi, Design Patterns for e-Science, Springer Verlag, 2007, ISBN 978-3-540-68088-8.
- [2] Dave Crane, Eric Pascarello with Darren James, AJAX IN ACTION, Dreamtech Press, 2006, ISBN 81-7222-657-6.
- [3] David Flanagan, Java Examples in a Nutshell, O'REILLY, 2001, ISBN 7-5083-0665-4.
- [4] Ajith Mannanakunnel Jose, Development of Web Scope, the final report for the COMP6703 eScience Project (Semester2, 2005) course, <http://escience/project/05S2/AjithJose/FinalReport.doc> . Last accessed 20 October 2007.
- [5] Le Ma, Development of Web Scope, the final report for the COMP6703 eScience Project (Semester2, 2006) course, http://escience/project/06S2/report/LeMa_report.pdf . Last accessed 20 October 2007.
- [6] Zhongshan Tan, Use of Echo2 in Web Scope, the final report for the COMP6703 eScience Project (Semester2, 2006) course, http://escience/project/06S2/report/ZhongshanTan_report.pdf . Last accessed 20 October 2007.
- [7] Henry J. Gardner, Raju Karia, Gabriele Manduchi, A Web-Based, Dynamic Metadata Interface to MDSplus, To be published.
- [8] Clayton Lewis, John Rieman, Task-Centered User Interface Design, 1993.
- [9] Marilyn Hughes Blackmon, Peter G. Polson, Muneo Kitajima, Clayton Lewis, Cognitive Walkthrough for the Web.
- [10] The java doc for EchoPointNG, <http://docs.rakeshv.org/java/echopointng/allclasses-noframe.html> . Last accessed 20 October 2007.
- [11] Apache Ant 1.7.0 Manual, <http://ant.apache.org/manual/> . Last accessed 20 October 2007.
- [12] Apache Tomcat Manual, <http://tomcat.apache.org/> . Last accessed 20 October 2007.

- [13] The java doc for the Karora Cooee project,
<http://www.karora.org/projects/cooee/apidocs/> . Last accessed 20 October 2007.
- [14] The Hibernate official website, <http://www.hibernate.org/> . Last accessed 20 October 2007.
- [15] The HSQLDB official website, <http://hsqldb.org/> . Last accessed 20 October 2007.
- [16] Java Servlet Technology, <http://java.sun.com/products/servlet/index.jsp> . Last accessed 20 October 2007.
- [17] The Karora official website, <http://www.karora.org/> . Last accessed 20 October 2007.
- [18] The Echo2 official website, <http://www.nextapp.com/platform/echo2/echo/> . Last accessed 20 October 2007.
- [19] Servlet Essentials, <http://www.novocode.com/doc/servlet-essentials/> . Last accessed 20 October 2007.
- [20] The Chinese version of the Java Servlet tutorial,
http://www.bc-cn.net/Article/web/jsp/jc/200409/72_3.html . Last accessed 20 October 2007.
- [21] The Wikipedia home page, <http://www.wikipedia.org/> . Last accessed 20 October 2007.
- [22] How to Conduct a Heuristic Evaluation,
http://www.useit.com/papers/heuristic/heuristic_evaluation.html . Last accessed 20 October 2007.
- [23] MDSPLUS home page, <http://www.mdsplus.org/> . Last accessed 20 October 2007.

Appendix A: Cognitive Walkthrough details

A.1 Tasks

A.1.1 Task details

The detailed tasks involved in this walkthrough have been listed in the table below (Table 6.1):

Task Name	Task Description	Action Sequence
Task1: Display the graph for a random experiment on the screen	Plotting graphs for the experiment data retrieved from the MDSPlus server is one of the main features of the WebScope3 system. This task is intended to test the usability of the graph plotting feature.	<ol style="list-style-type: none">1. Launch a web browser and open the page http://HOSTNAME:8080/WebScope3/app2. Input the email address of an existing user account and the corresponding password.3. Click the “Login” button.4. Click the “Go to connection page” at the user information page.5. Specify the server name, the port number, the experiment name and the shot number at the connection page.6. Click the “Connect to MDSPlus Server” button.7. Expand the experiment tree in the left pane at the main page. Click on one of the experiments. If the experiment contains experiment data, the graph will then be plotted in the right pane.
Task 2: Contribute static metadata to the database	The WebScope3 system allows users to interact with the database by contributing static metadata. The static metadata is bound to the experiment data. Thus, users can add their	<ol style="list-style-type: none">1. Launch a web browser and open the page http://HOSTNAME:8080/WebScope3/app2. Input the email address of an existing user account and the corresponding password.3. Click the “Login” button.

	<p>comments and notes with this feature. This task is intended to test the usability of the static metadata feature.</p>	<ol style="list-style-type: none"> 4. Click the “Go to connection page” at the user information page. 5. Specify the server name, the port number, the experiment name and the shot number at the connection page. 6. Click the “Connect to MDSPlus Server” button. 7. Expand the experiment tree in the left pane at the main page. Click on one of the experiments. 8. In the right pane, click the “Contribute static metadata” button under the experiment graph. 9. In the pop-up window pane, fill in the Metadata Name, Metadata Value, Metadata Explanation, and Metadata Keyword fields. 10. Click the “Submit Metadata” button in the window pane, and the metadata will be stored in the database as long as all the provided information is valid. Warning information will be given if some of the information is invalid.
<p>Task 3: Contribute dynamic tables to the database</p>	<p>If the static metadata cannot meet the requirements of users, they can also create customized metadata table and contribute these information to the database with the dynamic metadata table feature. This task is intended to test the dynamic table feature.</p>	<ol style="list-style-type: none"> 1. Launch a web browser and open the page http://HOSTNAME:8080/WebScope3/app 2. Input the email address of an existing user account and the corresponding password. 3. Click the “Login” button. 4. Click the “Go to connection page” at the user information page. 5. Specify the server name, the port number, the experiment name and the shot number at the connection page. 6. Click the “Connect to MDSPlus Server” button. 7. In the right pane of the main page, click the “Create dynamic table” button under the experiment graph. 8. In the “Create Dynamic Table” window pane, specify the Table Name

		<p>and Number of Columns. Then, click the “Submit” button.</p> <p>9. In the “Save Dynamic Table Column Names” window pane, specify the names of each column. Then, click the “Submit Column Name” button.</p> <p>10. In the “Save Dynamic Table Column Values” window pane, specify the values of each column. Then, click the “Submit Column Value” button.</p> <p>11. In the “Success” window pane, click the “Finish” button to complete this process.</p> <p>Note that warning information will be given if some of the fields are incorrectly filled in.</p>
<p>Task 4: Query a static metadata</p>	<p>Although a list of contributed static metadata is given, it will be better for users to query the static metadata using the query feature. The static metadata can be queried by server, by experiment, by shot, by node path, by metadata name, by metadata value, or by user. This task is intended to test the usability of the query feature.</p>	<ol style="list-style-type: none"> 1. Launch a web browser and open the page http://HOSTNAME:8080/WebScope3/app 2. Input the email address of an existing user account and the corresponding password. 3. Click the “Login” button. 4. Click the “Go to connection page” at the user information page. 5. Specify the server name, the port number, the experiment name and the shot number at the connection page. 6. Click the “Connect to MDSPlus Server” button. 7. In the right pane of the main page, click the “Query Metadata” button. 8. In the “Query” window pane, fill in the Keyword field and select a querying method in the “Query by” dropdown list. 9. Click the “Query” button. 10. Click the “OK” button in the “Query Results window pane” to complete this process.

Table A.1 Cognitive Walkthrough Tasks

A.1.2 Potential users

The potential users are physical scientists who do research in the nuclear fusion field. They may not be computer experts. However, they will understand the interface and features easily.

A.2 Walkthrough Results

The four questions that we are using to walkthrough the action sequences of each task are listed in the table below (Table 6.2):

Q1	Will users be trying to produce whatever effect the action has?
Q2	Will users see the control (button, menu, switch, etc.) for the action?
Q3	Once users find the control, will they recognize that it produces the effect they want?
Q4	After the action is taken, will users understand the feedback they get, so they can go on the next action with confidence?

Table A.2 Walkthrough Questions

The actual walkthrough of the action sequences listed in Table 6.1 and the results have been listed in the table below (Table 6.3):

Task ID	Sequence ID	Q1	Q2	Q3	Q4
1	1	Yes	Yes	Yes	Yes
1	2	Yes	Yes	Yes	Yes
1	3	Yes	Yes	Yes	Yes
1	4	Very likely	Very likely	Yes	Yes
1	5	Yes	Yes	Yes	Yes
1	6	Yes	Yes	Yes	Yes

1	7	Yes	Yes	Yes	Yes
2	1	Yes	Yes	Yes	Yes
2	2	Yes	Yes	Yes	Yes
2	3	Yes	Yes	Yes	Yes
2	4	Very likely	Very likely	Yes	Yes
2	5	Yes	Yes	Yes	Yes
2	6	Yes	Yes	Yes	Yes
2	7	Very likely	Very likely	Yes	Yes
2	8	Yes	Yes	Yes	Yes
2	9	Yes	Yes	Yes	Yes
2	10	Yes	Yes	Yes	Yes
3	1	Yes	Yes	Yes	Yes
3	2	Yes	Yes	Yes	Yes
3	3	Yes	Yes	Yes	Yes
3	4	Very likely	Very likely	Yes	Yes
3	5	Yes	Yes	Yes	Yes
3	6	Yes	Yes	Yes	Yes
3	7	Very likely	Very likely	Yes	Yes
3	8	Yes	Yes	Yes	Yes
3	9	Very likely	Yes	Very likely	Very likely
3	10	Very likely	Yes	Very likely	Very likely
3	11	Yes	Yes	Yes	Yes
4	1	Yes	Yes	Yes	Yes
4	2	Yes	Yes	Yes	Yes

4	3	Yes	Yes	Yes	Yes
4	4	Very likely	Very likely	Yes	Yes
4	5	Yes	Yes	Yes	Yes
4	6	Yes	Yes	Yes	Yes
4	7	Very likely	Very likely	Yes	Yes
4	8	Yes	Yes	Yes	Yes
4	9	Yes	Yes	Yes	Yes
4	10	Yes	Yes	Yes	Yes

Table A.3 Action Sequence Walkthrough

A.3 Summary

Most of the actions needed to complete the tasks above are obvious and straightforward, which can be concluded from the very high mark we've got from the walkthrough. However, sometimes it may not be that easy to find the buttons/controls for a certain control if the user is not very familiar with the interface. To resolve these issues, some changes have been made to the layout of the WebScope3 system. Some examples are given below:

The main screen of the WebScope3 system was too long and we needed to scroll down the screen to find features such as "contribute static metadata", "query metadata", and "create dynamic table". Although this may be a convenient and direct way to show the users which information is needed for each feature, it may cause problems for users who are not familiar with the program layout to find these features (See Figure A.1). To resolve this potential problem, the three features mentioned above are no longer put on the screen in the original way. Instead, three buttons have been added to the screen to trigger these features. This way, we do not need to scroll down the screen to find any feature and everything fits in the main screen now (See Figure A.2).

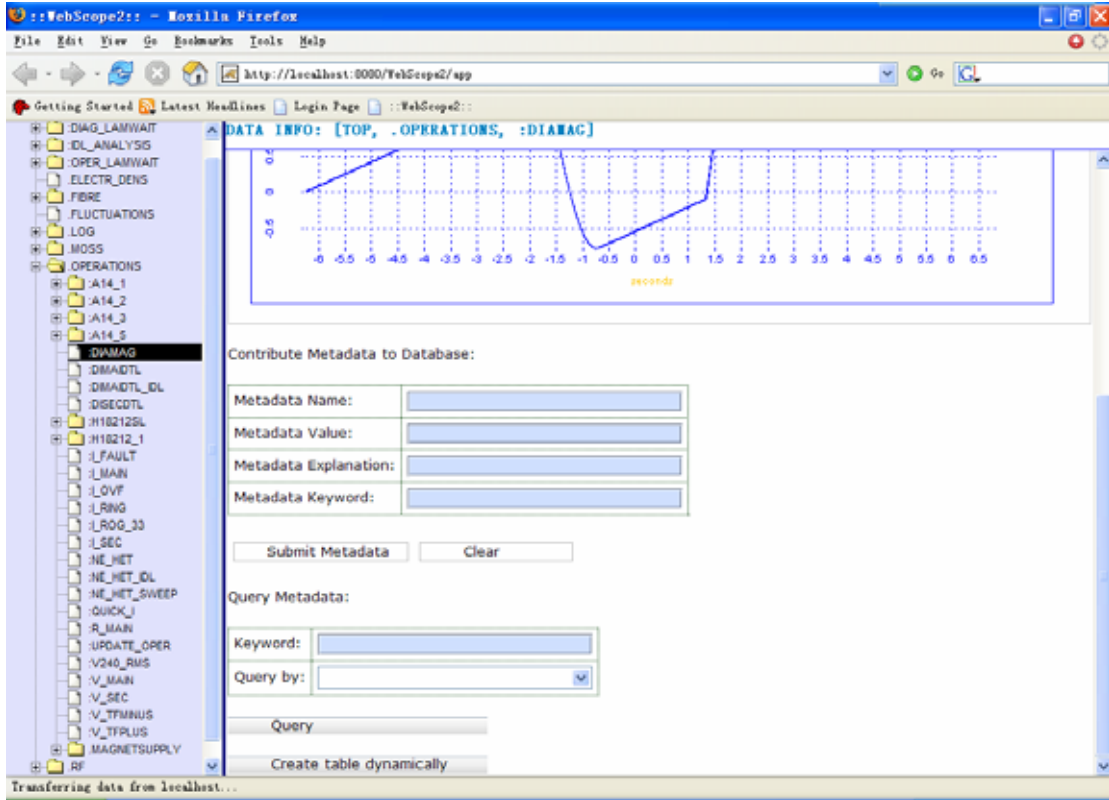


Figure A.1. Original Main Screen Layout

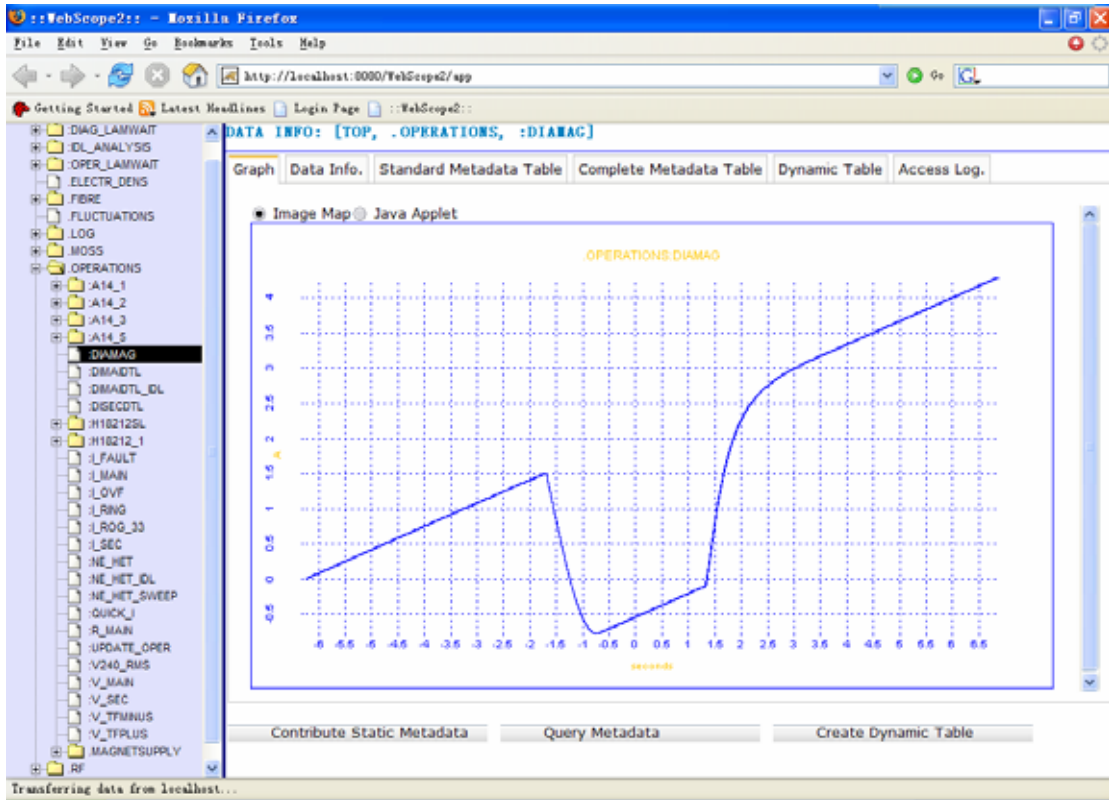


Figure A.2. New Main Screen Layout

As some additional features will be added to the WebScope system in the future, it is important that we pay more attention to the layout of the application. To further investigate the usability problems, I have also performed a Heuristic Evaluation on the WebScope3 system. Detailed information has been included in Appendix B.

Appendix B: Heuristic Evaluation

B.1 Ten Usability Heuristics

These are ten general principles for user interface design [22].

- **Visibility of system status**
The status of the system should always be visible, which means that users should be informed of what's going on and what information is required to proceed.
- **Match between system and the real world**
System should follow real-world conventions and work in the style that is familiar to users. This involves the establishment of a good mental model.
- **User control and freedom**
Users should have control on the system under all conditions. They should be able to undo/redo their actions, too.
- **Consistency and standards**
The same words and terms should be used to describe the same features or functions.
- **Error prevention**
The system should be design carefully to prevent potential errors or careless mistakes. Confirmations should be presented to users before they commit an action.
- **Recognition rather than recall**
The system should not require users to remember the state of the system or certain information provided in another part of the system. Every piece of required information should be retrievable whenever users need them.
- **Flexibility and efficiency of use**
Some advanced options or features, such as shortcut or batched actions, should be provided to expert users to improve the efficiency of the system. This way, the system will be suitable for both novice users and expert users.
- **Aesthetic and minimalist design**
Everything included in a screen or a dialogue box should be closely relevant to the actions users are performing. The irrelevant information in a screen or a dialogue box may distract the users' attention and cause confusion.

- Help users recognize, diagnose, and recover from errors
Error messages should be clear and informative. It will always be good to provide users with a possible solution to the problem they've encountered.
- Help and documentation
Help topics should be provided to cover the common problems which may occur and FAQ's.

The principles above are used as the check list during this Heuristic Evaluation.

B.2 Evaluation Results

As mentioned above, I will use the ten usability heuristics to evaluate the WebScope3 system.

B.2.1 Visibility of system status

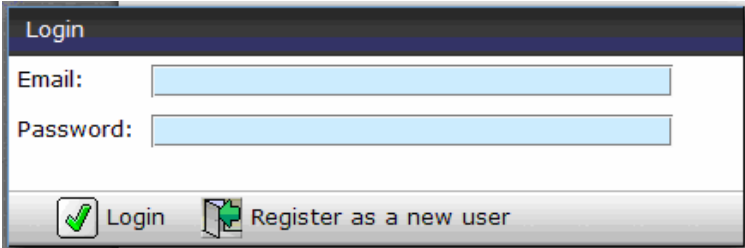
Every screen of the WebScope3 system provides users with the necessary information about what the screen is for and what information is required to proceed to next screen.

Every button and text box is accompanied with corresponding explanation. The titles of the dialogue boxes can clearly show their purposes.

Therefore, I don't think the system has any visibility issues.

B.2.2 Match between system and the real world

The layout of the screens in the WebScope3 system is designed in the convention adopted by most of the popular websites. Thus, users will not find it odd to work in this system. For example, the data tree is put to the left of the main screen, which is a common way used in most operating systems (See Figure A.2). Another example is that the login box in the login screen should also be quite familiar to users as similar login boxes can be seen everywhere (See Figure B.1).



The image shows a screenshot of a web-based login form. The form has a dark blue header with the word "Login" in white. Below the header, there are two input fields: "Email:" followed by a light blue text box, and "Password:" followed by another light blue text box. At the bottom of the form, there are two buttons: "Login" with a green checkmark icon and "Register as a new user" with a green plus icon.

Figure B.1. Login Dialogue Box

Therefore, the system matches the real-world very well.

B.2.3 User control and freedom

I do find some problems about user control and freedom in the WebScope3 system:

- Users do not have the option to log out and log in with another account once they reach the main screen.
- Users cannot cancel the action if they click the “Create dynamic table” button. They have to complete a new dynamic table (See Figure B.2).

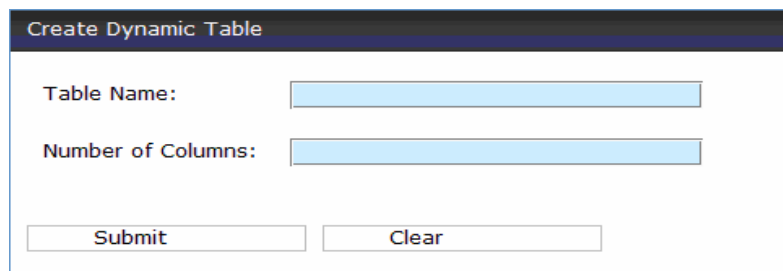
The image shows a dialog box titled "Create Dynamic Table". It has a dark blue header bar with the title in white. Below the header, there are two text input fields. The first is labeled "Table Name:" and the second is labeled "Number of Columns:". Both input fields have a light blue background. At the bottom of the dialog, there are two buttons: "Submit" on the left and "Clear" on the right. The dialog box has a thin border and a drop shadow.

Figure B.2. Create Dynamic Table Dialogue Box

- Once users have contributed some metadata to the database, they do not have the option to undo this action. That is, they cannot delete metadata from the database.

The problems above may cause some potential usability issues. Currently, the second problem has been fixed.

B.2.4 Consistency and standards

The same words/terms are used to describe the same features in the WebScope3 system. Therefore, we can say that the WebScope3 system is consistent.

B.2.5 Error prevention

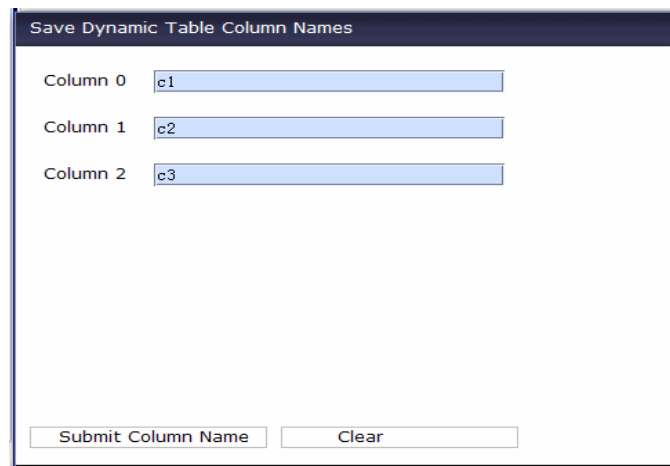
When the WebScope3 system was designed, considerations were taken to prevent potential problems caused by careless mistakes. For example, the system is divided into 4 consecutive screens: the login screen, the user information screen, the database connection screen and the main screen. Users will not be able to proceed to the next screen unless they provide the correct information, which can be considered as an interlocking mechanism. Many potential errors have been prevented with this

mechanism.

B.2.6 Recognition rather than recall

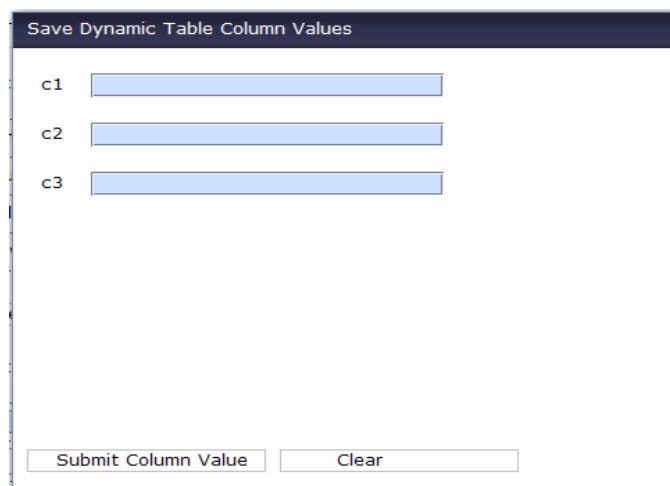
The WebScope3 system does not require users to remember any information. Instead, all the information they are looking for can be found somewhere in the screen.

For example, when a new dynamic table is created, users are required to input the name of the columns in the new table. Then, users are required to input the values of each column. These two inputs happen in two different dialogue boxes and it is possible that the users may have forgotten what the name for each column is. To prevent this issue, the WebScope3 system shows the column names to users when they are required to input the column values (See Figure B.3 & Figure B.4).



The dialog box has a title bar that reads "Save Dynamic Table Column Names". Below the title bar, there are three rows of input fields. The first row is labeled "Column 0" and contains the text "c1". The second row is labeled "Column 1" and contains the text "c2". The third row is labeled "Column 2" and contains the text "c3". At the bottom of the dialog box, there are two buttons: "Submit Column Name" and "Clear".

Figure B.3. Save Dynamic Table Column Names Dialogue Box



The dialog box has a title bar that reads "Save Dynamic Table Column Values". Below the title bar, there are three rows of input fields. The first row is labeled "c1", the second row is labeled "c2", and the third row is labeled "c3". At the bottom of the dialog box, there are two buttons: "Submit Column Value" and "Clear".

Figure B.4. Save Dynamic Table Column Values Dialogue Box

B.2.7 Flexibility and efficiency of use

As the interface and features of the WebScope3 system is relatively simple, no additional accelerator is provided to expert users. However, it will be import to take this into account in the future when the WebScope system gets more and more complicated.

B.2.8 Aesthetic and minimalist design

As is shown in the figures above, no unnecessary information is included in the dialogue boxes or screens. Therefore, the WebScope3 system meets the requirement of this principle.

B.2.9 Help users recognize, diagnose, and recover from errors

The error messages thrown out when users make mistakes are quite informative. Examples are given in the figures below:

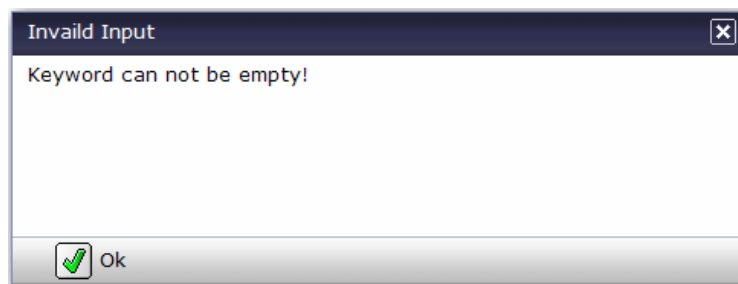


Figure B.5. Empty Query Keyword Error Message

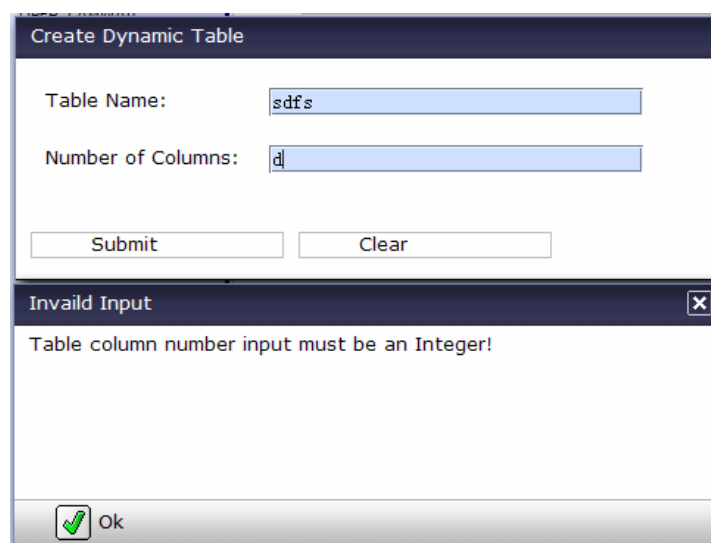


Figure B.6. Invalid Dynamic Table Column Number Error Message

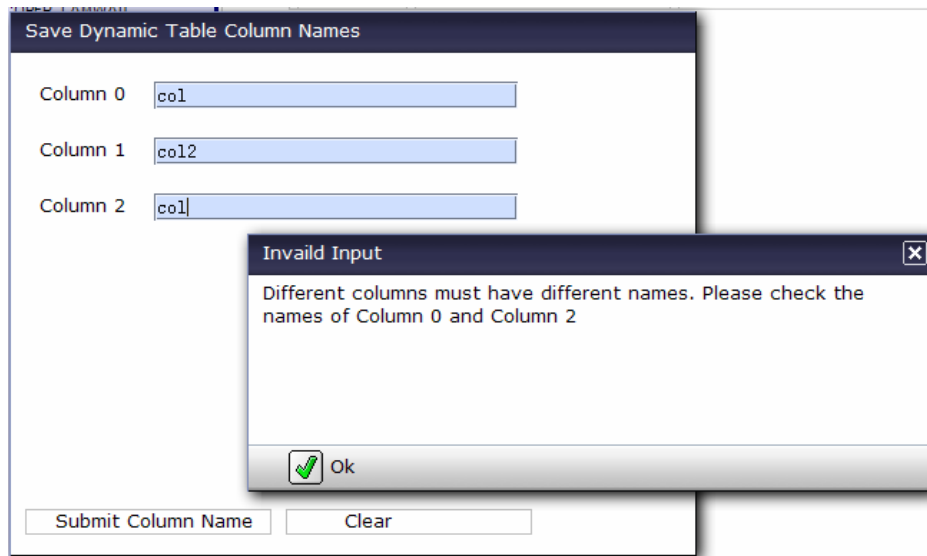


Figure B.7. Invalid Column Name Error Message

B.2.10 Help and documentation

Currently, no help topics or documentation are provided to users. We may add these in the future to further improve the usability of the system.

B.3 Conclusion

The WebScope3 system can meet the requirements of most of the principles. However, there are still some problems and the usability can be further improved.

- Users do not have full control and freedom in some part of the system.
- No help documentation has been provided to users.

The first problem has been resolved. However, the second one requires some future work.

Appendix C: Progress on the Servlet-based WebScope system

In addition to the Cooee-based WebScope3 system, some additional work has been done on the Servlet-based WebScope system (Mr. Le Ma's version) to improve its functionality. The major progress is that two new features have been added.

C.1 Metadata Table

A metadata table has been added to the User Information screen. This way, users will be able to get a list of metadata before connecting to the MDSPlus server. The screen shots below (Figure C.1 & Figure C.2) illustrate the change:



Figure C.1. Original User Information Screen



Figure C.2. Updated User Information Screen

C.2 Metadata Query Feature

A metadata query feature has been added on the Node Selection screen. Metadata can be queried by server, experiment, shot number, node path, user, metadata name, or metadata value. In addition, a link has been provided for every search result, which can lead users to the related graph data. The images below illustrate the changes:

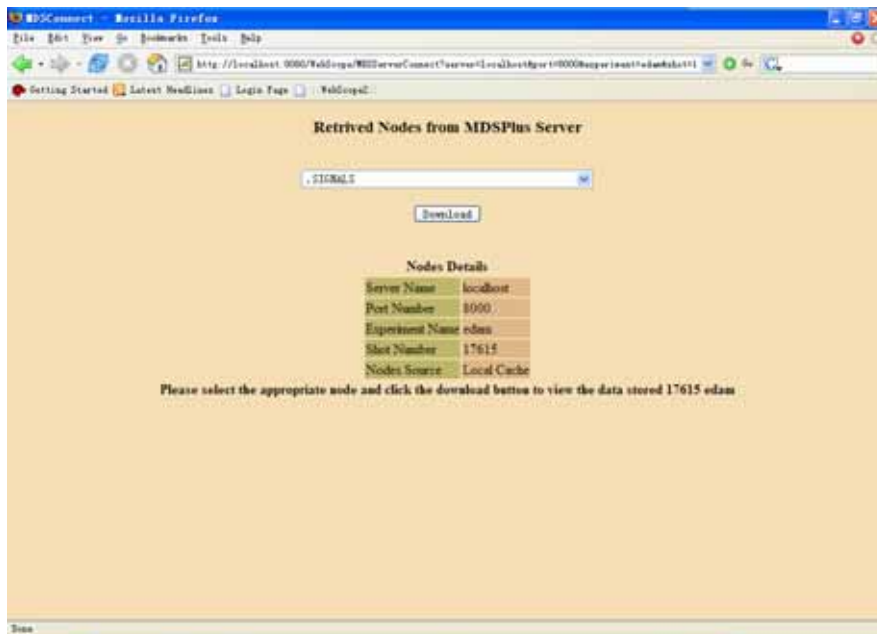


Figure C.3. Original Node Selection Screen



Figure C.4. Updated Node Selection Screen

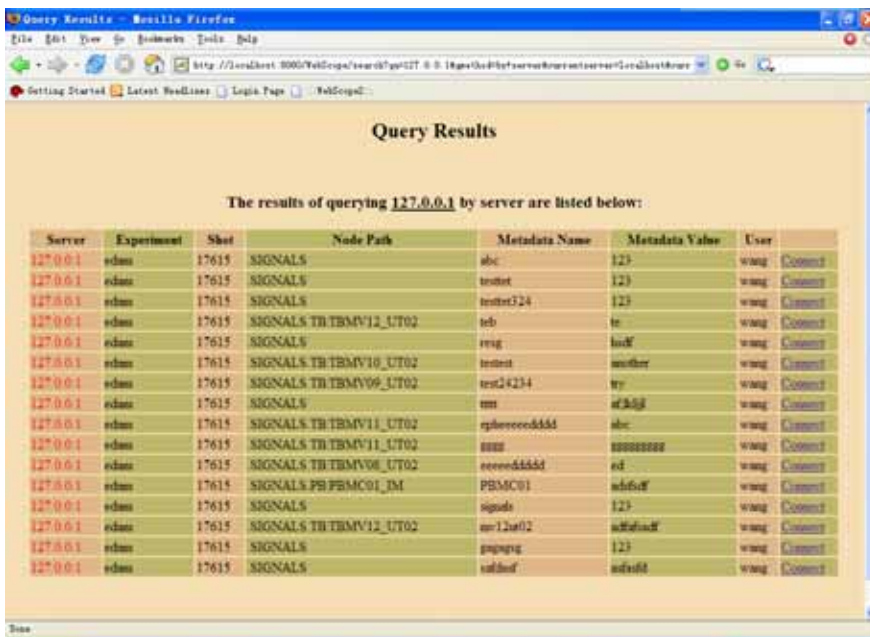


Figure C.5. Query Results Screen

Appendix D: Project Log

➤ 23rd Jul ~ 29th Jul

Understand the general idea of the WebScope project. Contact Dr. Henry Gardner to get some clarification.

➤ 30th Jul ~ 5th Aug

Study Ant, NetBeans, Eclipse, Apache Tomcat, Java Servlets, Echo2, Cooee and Hibernate. Prepare for the actual project. Install required programs on my laptop to establish a developing Environment for the WebScope3 project.

➤ 6th Aug ~ 12th Aug

Get the Le Ma's copy of code. Compile and run it on my laptop. Read the codes and get familiar with the coding style.

Problem found:

Le Ma's code cannot retrieve graph information from the local MDSPlus Server. However, it works with the ephebe server.

➤ 13th Aug ~ 19th Aug

Modify Le Ma's copy of code according to Henry's requirements. The following features have been added:

1. A metadata table is displayed on the page just after a user logs on his account.
2. The metadata query feature has been realized. Metadata can be queried by server, experiment, shot, node path, metadata name, metadata value and user. There is a link beside every query result which can lead the user to the actual graph.

➤ 20th Aug ~ 26th Aug

Get Zhongshan Tan's copy of code. Compile and run it on my laptop. Compare it with Le Ma's. The following feature has been added to Zhongshan's version:

1. Migrate Zhongshan's version from Echo2 to Karora-Cooee.
2. Contribute metadata to the database and display the metadata table on the user information screen and the data plotting screen.
3. Record access log of users and display a user log table on the data plotting screen.
4. Metadata querying feature has also been realized. Metadata can be queried by server, experiment, shot, node path, metadata name, metadata value and user. There is a link beside every query result which can lead the user to the actual graph.

Note:

Zhongshan's copy can work fine with the local MDSPlus Server

➤ **27th Aug ~ 2nd Sep**

Add the dynamic metadata feature to the new Karora-Cooee version of the WebScope system. This includes:

1. Dynamically create .java files.
2. Dynamically compile the .java files and create .class files.
3. Dynamically add table information to the Hibernate mapping file Dynamic.hbm.xml
4. Contribute dynamic table to the HSQL database.

Problem found:

Although Le Ma's version does create .java files, compile them and create .class files, and add the table information to the file Dynamic.hbm.xml, NO dynamic table is contributed to the database because:

1. The column names in the HTML output do not match.
2. Le Ma tried to save a Class object using the "save" method of the "session" class, which is not right. A new instance should be created for the Class object and we should save the new instance instead.
3. Although table information is added to the Hibernate mapping file Dynamic.hbm.xml correctly, the table is not loaded as expected during runtime. To resolve this issue, I use the "addResource" method of the "Configuration" class to re-add the mapping file.

➤ **3th Sep ~ 9nd Sep**

Modify the source code so that canonical file path is used instead of absolute path.

Study the new scripting technology OpenJFX and try to find out a way to load OpenJFX scripts in the Cooee framework. Write the report draft.

Problem found:

OpenJFX scripts cannot be loaded directly in the Cooee framework. However, I can insert a HTML panel using the DirectHTML class of Cooee, load a Java Applet using HTML language, and then load the OpenJFX script in the applet. Maybe I can find a more direct way to achieve this.

➤ 10th Sep ~ 16nd Sep

Continue studying the OpenJFX technology and try to make it work with the Cooee framework. Continue writing the report. As I still have many questions on OpenJFX, I am currently trying to plot the graphs with a Java Applet instead of a OpenJFX script.

Problem found:

1. It seems that we cannot bypass Java Applet if we would like to load a OpenJFX script in the Cooee framework. The problem is that if Java Applet is involved, we will need the full version of JRE on the client side to load the WebScope3 program, which does not meet the requirement of Dr. Henry. One of the main reasons that we try adopting OpenJFX is that only a minimum version of JRE is required on the client side to run OpenJFX script.
2. OpenJFX script cannot be loaded in Cooee framework. I have performed some thorough research on the OpenJFX and Karora website and found that there are two possible ways to load a OpenJFX script in a web page:
 - Via Java Applet
 - Via Java Web Start

For the Java Applet method, two examples have been found.

- Load the OpenJFX script with the FXMain class
- Load the OpenJFX script with the ScriptEngine class.

However, I have tried all the methods above without any success. I have posted some questions in the forums of the two websites mentioned above and somebody suggested that we use JFreeChart(<http://www.jfree.org/jfreechart/>) instead. Here is the demo: <http://demo.nextapp.com/Demo/app>

3. I cannot find a way to make a OpenJFX script communicate with Cooee. The communication is required because we have to change the graph every time a new

tree node is clicked.

➤ **17th Sep ~ 23rd Sep**

After discussing with Dr. Henry Gardner, the OpenJFX technique has been abandoned as it cannot meet our needs. Instead, a new requirement has been issued that we should provide users with two methods of graph-displaying: the Image Map method and the Applet method. A radio button has been added on the Graph pane in the main screen of the system and the graph-displaying method can be switched from one to another by a single click on the radio button. I'm also writing up the report draft during this week.

Problem found:

1. I'm currently using the URL class to create a URLConnection connection and retrieve a OutputStream from this connection. However, it seems that I cannot get the nodedata object in the Applet which is written using the writeObject method of the ObjectOutputStream class in the main program. An "Invalid stream header" error is always received. I have not been able to resolve this issue.
2. Although only one graph-displaying method is working at any time, I still need to inform both methods of the node-clicking event and update the graph information of them whenever a tree node is clicked on. Otherwise, I will not be able to get the proper graph when switching between the graph-displaying methods.

➤ **24th Sep ~ 30th Sep**

A cognitive walkthrough has been performed during this week. The report draft is being written and the communication problem between the WebScope3 system and the Java Applet has been resolved now. Some application bugs have been resolved, too.

Problem found:

I was convinced that the method I was testing during the last week was not correct. Therefore, I searched on the web and the Cooee website trying to find another way to get an HTTP connection.

After reading the Cooee javadoc thoroughly, I found a class called WebRenderServlet which could be used to simulate a servlet in the Cooee framework. I tried calling the getActiveConnection() method of this class to get an HTTP connection and then retrieving an outputstream from this connection. The process went smoothly until I

tried writing something into this stream. The whole application froze and no error messages were thrown out in the GUI or in the TomCat terminal.

I thought that this might be because I was not writing the right type of data into the stream. I tried writing a string, a char, an integer into the stream to no avail. I also tried using the `DataOutputStream`, `ByteArrayOutputStream`, `BufferedOutputStream` and all the other available sub-classes of the `OutputStream` class. The same issue persisted no matter what I did. No one knew what was happening even on the Cooe website.

After testing and struggling for more than 40 hours, I gave up on the `WebRenderServlet` class and picked up the `Socket` method again. To prevent the socket server from freezing the whole application when waiting for the client to connect, I learned to create a thread for each socket connection. The applet works fine after all.

➤ **1st Oct ~ 7th Oct**

A Heuristic Evaluation has been performed and the results have been written up. Some changes have been made to the layout of the `WebScope3` system according to the results. The design pattern analysis has been partially finished. The report has also been modified according to Dr. Henry's advice/feedback.

Some of the problems found in the Heuristic Evaluation cannot be resolved for the time being due to the limitation of time. I have included these problems in the Future Work section of the report.

➤ **8th Oct ~ 14th Oct**

The design pattern analysis has been finished and written up in the report. I have been writing up the report and now a good draft can be handed in. I have also performed a thorough test on the `WebScope3` system and written up the results in the report. Modification has been made to the codes according to the results. Some bugs in the code and the report have been fixed.

➤ **15th Oct ~ 21st Oct**

Prepare for the talk on the `COMP6444` lecture. Modify the report as per Dr. Henry's suggestions.

➤ **22nd Oct ~ 28th Oct**

Prepare for the final presentation. Modify the report as per Dr. Henry's suggestions. Test the WebScope3 system again in both Windows and Linux environments. Some bugs have been fixed.

Appendix E: Installation and Compilation Guide

E.1 Prerequisites:

The WebScope3 system requires JRE on the clients and Apache Tomcat running on the server. The paragraphs below describe the steps to install JRE on the clients and Tomcat on the server:

E.1.1 Install JRE 5.0 or later versions on the clients

- On the clients, download Java 2 Standard Edition Runtime Environment (JRE), release version 5.0 or later, from <http://java.sun.com/j2se> .
- Install JRE according to the instructions shipped with the installation package.
- Set an environment variable named JAVA_HOME to the path where JRE is installed. e.g. c:\Program Files\j2sdk5.0 (Windows) or /usr/local/java/j2sdk5.0 (Unix/Linux).

E.1.2 Install Apache Tomcat5.0 or Later

- Download Apache Tomcat version 5.0 or later from the link below:
<http://tomcat.apache.org/>
- Unzip the downloaded installation package to required location. The symbolic name "\$CATALINA_HOME" will be used to refer to the full path of the installation directory for convenience purpose.

To run Tomcat:

- Tomcat can be started by executing the following commands:

```
$CATALINA_HOME\bin\startup.bat (Windows)  
$CATALINA_HOME/bin/startup.sh (Unix/Linux)
```

- Afterwards, the default web application shipped with Tomcat will be available at the web address below:

<http://localhost:8080/>

To shut down Tomcat:

- Tomcat can be shut down by executing the following command:

```
$CATALINA_HOME\bin\shutdown (Windows)
```

`$CATALINA_HOME/bin/shutdown.sh` (Unix)

E.2 Installation of WebScope3:

To install the WebScope3 system on a server equipped with Apache Tomcat, please use the following steps:

1. Obtain the file `WebScope3_all_in_one.zip`.
2. Unzip this file and you will get the file `WebScope3.war`.
3. Copy the file `WebScope3.war` to the folder below:
`$CATALINA_HOME\webapps` (Windows)
`$CATALINA_HOME/webapps` (Unix/Linux)
4. Run Tomcat.
5. Run the HSQLDB database with the commands below:

(Windows)

```
java -Xmx256m -cp $CATALINA_HOME\webapps\WebScope2\WEB-INF\lib  
hsqldb.jar org.hsqldb.Server -database.0 hdb -dbname.0 hdb
```

(Unix/Linux)

```
java -Xmx256m -cp $CATALINA_HOME/webapps/WebScope2/WEB-INF/lib  
hsqldb.jar org.hsqldb.Server -database.0 hdb -dbname.0 hdb
```

6. WebScope3 is installed and can be accessed from the clients via the link below:

http://SERVER_HOST_NAME:8080/WebScope3/app

NOTE: Please replace the `SERVER_HOST_NAME` part with the proper IP address or host name of the server.

E.3 Compilation of the WebScope3 source code:

To compile the WebScope3 source code, please use the following steps:

1. Obtain the file `WebScope3_all_in_one.zip`.
2. Unzip this file and you will get the file `apache-ant-1.7.0-bin.zip`.
3. Install Apache Ant according to the user manual from the link below:
<http://ant.apache.org/manual/>
4. Set Environment variables using the commands below:

(Windows)


```
set ANT_HOME=(Ant installation path)
set JAVA_HOME=(Java path)
set PATH=%PATH%; ANT_HOME \bin
```

For example:

```
set ANT_HOME=c:\ant
set JAVA_HOME=c:\java\jdk1.5.0_01
set PATH=%PATH%;c:\ant\bin
```

(Unix/Linux)

```
export ANT_HOME=( Ant installation path)
export JAVA_HOME=(Java path)
export PATH=${PATH}:{ANT_HOME}/bin
```

For example:

```
export ANT_HOME=/usr/local/ant
export JAVA_HOME=/usr/local/java/jdk1.5.0_01
export PATH=${PATH}:{ANT_HOME}/bin
```

5. Unzipping the file `WebScope3_all_in_one.zip` also gives you the file `WebScope3_src.rar`. Unzip this file to a desired folder/directory. You should see the folder `WebScope3_src`.
6. Open a command terminal and change the current directory to the folder `WebScope3_src`.
7. Run the command “ant” (without the quotation marks).

NOTE: This command will create the file `WebScope3.war` and copy it to the `webapps` directory in the default installation path of Apache Tomcat. However, it may not work on your side as the installation path of Apache Tomcat may not match the default directory. If this happens, please manually copy the file `WebScope3.war` to the proper directory as instructed in D.2.