

DØ Calibration Database Browser

Henry Barnor
Oberlin College

<http://www.cs.oberlin.edu/~hbarnor>

Supervisor: Taka Yasuda

August 7, 2003

Abstract

The key to analyzing data from the DØ detector is the calibration database. This paper describes the design and implementation of a web-based browser for the DØ calibration database.

Contents

1	Introduction	4
2	Design Considerations	4
3	First Attempt - MISWEB	5
3.1	MISWEB Implementation	6
4	Problems with MISWEB	6
5	Second Attempt - Java Applet	6
5.1	Package calibDbView	8
5.1.1	Class diagram of calibDbView package	9
5.1.2	D0PE class	9
5.1.3	ID0View interface	9
5.1.4	D0PPanel class	9
5.1.5	SearchPanel class	10
5.1.6	TablePanel class	10
5.1.7	D0ChartPanel class	10
5.2	Package calibDbData	11
5.2.1	Class diagram of calibDbData package	11
5.2.2	OracleConnector class	11
5.2.3	DataManager class	11
5.2.4	D0TableModel class	12
5.3	Package calibDbData.states	13
5.3.1	Partial class diagram calibDbData.states package	13
5.3.2	AState class	14
5.3.3	ADrillState class	14
5.3.4	APlotState class	14
5.3.5	AChartState class	14
5.3.6	CustomChartState class	14
5.3.7	TopState class	14
5.3.8	NullState class	15
5.4	Package calibDbCommand	15
5.4.1	Class Diagram of calibDbCommand package	15
5.4.2	D0PACommand class	15
5.4.3	AFrameCommand class	16

5.4.4	SwitchComponent class	16
5.4.5	D0Query class	16
5.4.6	TopQuery class	16
5.4.7	ChartQuery class	16
5.5	Plotting Library - JFreeChart	16
6	Problems with Java Applet	16
7	Third Attempt - Java Server Pages	18
7.1	Package beans	18
7.1.1	DrillBean class	19
7.1.2	TableBean class	19
7.1.3	SearchBean class	19
7.1.4	ChartBean class	19
7.2	Package calibDbServlets	19
7.2.1	SearchServlet class	19
7.2.2	DrillServlet class	19
7.2.3	DrillServlet class	19
7.3	Package calibDbData.states	20
7.3.1	IServletState interface	20
7.4	Package calibDbData	20
7.4.1	CalibDatasetProducer class	20
7.5	Java Server Pages(JSP)	20
8	Conclusion	22
9	Acknowledgement	23
	References	24
	Appendix	25
A	Class Diagram of subclasses of ADrillState	25
B	Class Diagram of subclasses of APlotState	26
C	Model-View-Controller Design Pattern	27
D	The State Design Pattern	28

E	The Singleton Design Pattern	29
F	The Command Design Pattern	30
G	The Template Design Pattern	31
H	The Observer-Observable Design Pattern	32
I	The Null design Pattern	33

List of Figures

1	Comparison of data in a tabular form to data in a plot form	5
2	The MISWEB search page	7
3	A page returned by MISWEB	8
4	The MISWEB drill down scenario	8
5	The Class diagram of the calibDbView package	9
6	The search panel	10
7	A table panel	11
8	The class diagram of the calibDbData Package	12
9	The class diagram of the calibDbData.states package	13
10	The class diagram of the calibDbCommand package	15
11	The plot of data using JFreeChart	17
12	A page returned by the server-side implementation	21
13	Server-side with multiple drill down columns	21
14	Server-side plot of data	22
15	Class Diagram for subclasses of ADrillState	25
16	Class Diagram for subclasses of APlotState	26
17	The model-view-controller	27
18	The Class Diagram of the State Design Pattern	28
19	Class Diagram of the Singleton Design Pattern	29
20	Class Diagram of the Command Design Pattern	30
21	Class Diagram of the Template Design Pattern	31
22	Class Diagram of the Observer-Observable Design Pattern	32

1 Introduction

The Fermi National Accelerator Laboratory, Fermilab, advances the understanding of the fundamental nature of matter and energy by providing leadership and resources for qualified researchers to conduct research at the frontiers of high energy physics and related disciplines. One of its experiments is the DØ experiment. The research is focused on precise studies of interactions of protons and antiprotons at the highest available energies. The DØ detector is used to observe these interactions. The process of converting interactions to electric signals is described by the equation:

$$\epsilon = aV + b$$

Where a and b are constants.

a = multiplication factor(gain)

b = offset(pedestal value)

The process of obtaining these constants is called calibration. This is done by sending a known voltage through each channel¹. The constants for each channel are calculated and stored in a database. Researchers use computer programs that query the database for the values and combine it with data recorded during an actual proton/antiproton interaction to recreate what happened in the interaction. However, researchers need to look at the data themselves to ensure that constants in the database are reasonable. My project is to provide a graphical way for the researchers to review the data without having to learn complex database commands.

2 Design Considerations

The DØ experiment is an international collaboration which involves scientists in about 19 countries. They will need to access this data from wherever they are; thus a web based system is the best way to go. The users should be presented with a webpage that allows them to start from the top level of the database and explore the hierarchy of the database. Due to the estimated size of all the data in the database, the user should be given the option of searching for the calibration runs that interest him/her. Each calibration has a unique number associated with it called the calibration id. This is an obvious search parameter. The time of the run, the luminosity² and version³ can also be used as search parameters.

¹There are about 900,000 electric channels in the DØ detector.

²Luminosity -is a measure of the number of collisions between proton and anti-proton beams

³Version numbers are used when a calibration run has to be redone

Looking at a table of data can sometimes be daunting especially when one has to look at about a 100 or more rows of data. On the contrary, it is much easier to obtain pertinent information from a plot of the data. There are several advantages to using plots:

- It is easier to spot anomalies from a plot;
- Patterns can also be easily recognized;
- Comparisons of various calibration runs can be easily made.

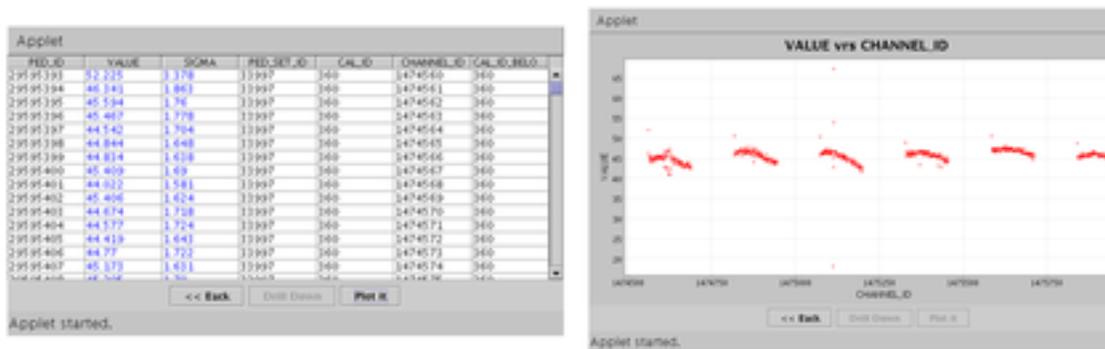


Figure 1: Comparison of data in a tabular form to data in a plot form

The user should therefore be given the option of plotting the constants against the various channel id's. See Figure 1.

There is a possibility that some users will be well versed in SQL⁴ syntax. These users should be provided with a field that allows them to type in an SQL query that will be executed and a plot of the results displayed.

Implementation

3 First Attempt - MISWEB

For my first attempt at implementing this system, my supervisor suggested MISWEB - a database tool developed by a contractor for Fermilab. It is implemented

⁴SQL is a programming language for databases[3]

in perl and allows a developer to query any Oracle® database running SQL*Net using basic html form elements.

3.1 MISWEB Implementation

To implement the system using MISWEB, I created a web-form with fields for the search parameters discussed above. Hidden in the html for the page is the name of the calibration database, the username and password for connecting to the database. The name of the table and the columns that MISWEB should return were also in hidden in the html. Variables in the html defined for MISWEB which of the columns it was returning could act as a drill down column. MISWEB also required that a link to another html page which defined all of the above options for the next level be included.

When these pages are loaded in a web browser, the browser displays the form and ignores the MISWEB specific options hidden in the form. See Figure 2. However, when the user submits the form, the hidden options together with the values the user entered in the fields are sent to MISWEB for processing. MISWEB then returns a page containing the results of the query in a tabular form to the user. See Figure 3.

4 Problems with MISWEB

MISWEB is a good tool for displaying data from a database in a webpage, but it is not the right tool for exploring a database. It lacks some very basic exploration functionality. MISWEB does not allow the programmer to specify more than one drill down direction. Thus we can only drill down in one direction. See Figure 4

Another problem with MISWEB is that, it cannot generate plots of the data. Therefore other alternatives were explored.

5 Second Attempt - Java Applet

In order to counter the problems associated with MISWEB and to satisfy all of the design considerations and goals of the project, the building of a web based program from scratch was necessary. One of the most important requirements is that

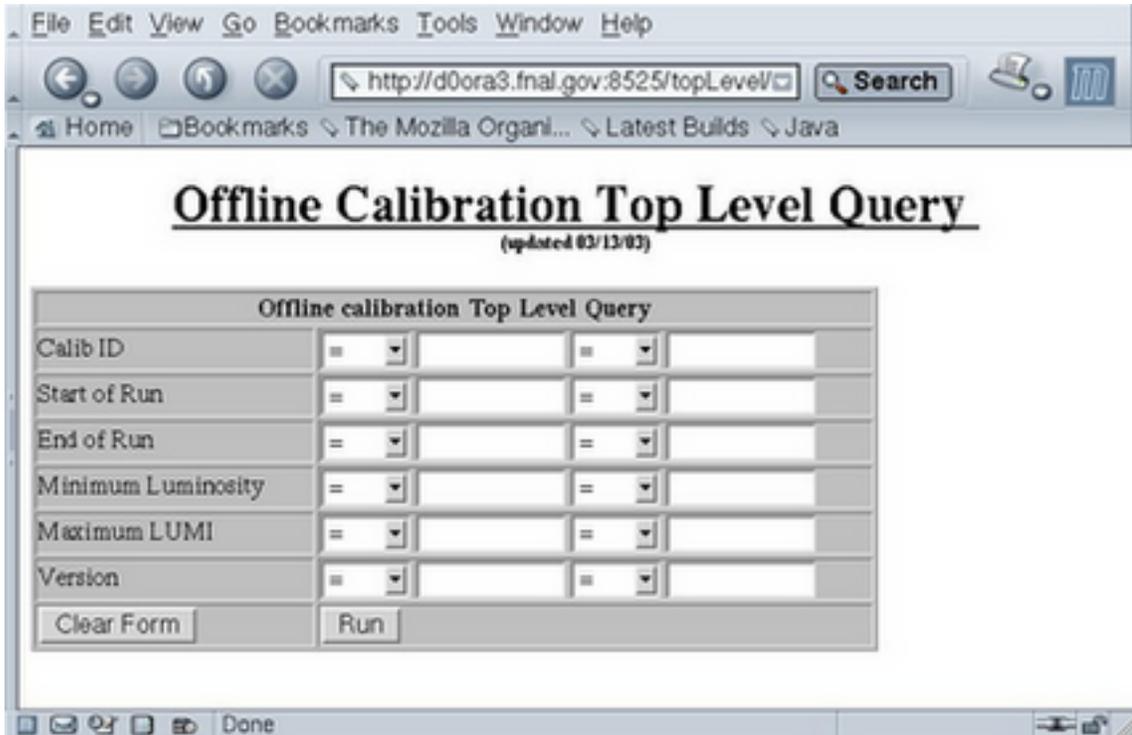


Figure 2: The MISWEB search page

the program must be able to plot the data. The availability of free Java™plotting libraries and the power of the Java applet technology made a strong case for implementing the second attempt using the Java™programming language.

Program Description

The program was written as a Java™applet and was designed according to the model-view-controller(MVC) design pattern (See Appendix C). This required separating the code into calibDbView, calibDbCommand, calibDbData and calibDbData.states packages. These packages are outlined below with descriptions and class diagrams ⁵.

⁵Class diagrams describe the static structure of a system[1]

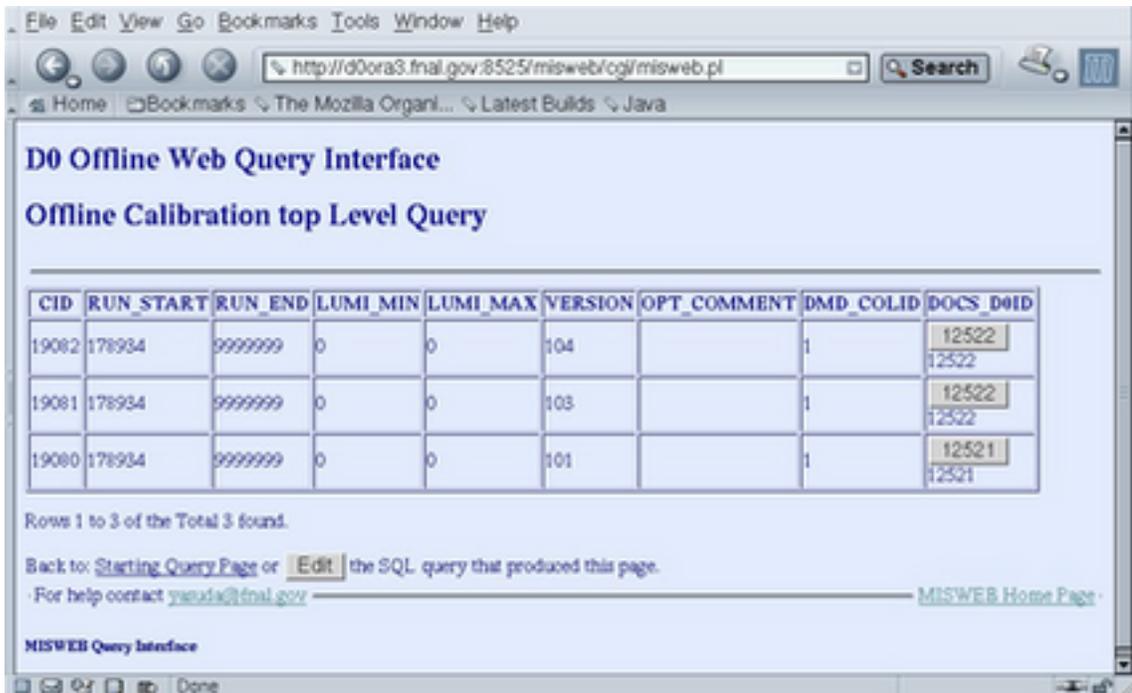


Figure 3: A page returned by MISWEB

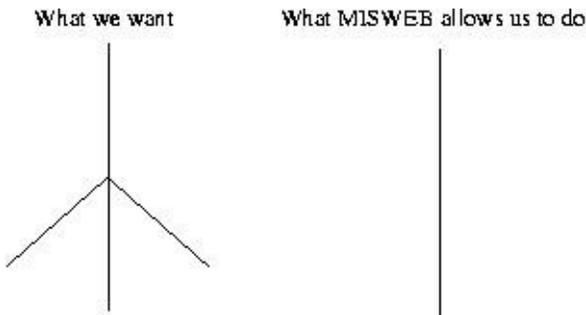


Figure 4: The MISWEB drill down scenario

5.1 Package calibDbView

The calibdbView package holds the view of the MVC design pattern used in the program. This package has 6 classes(D0ChartPanel, D0PE, D0PPanel, ID0View, SearchPanel and TablePanel).

5.1.1 Class diagram of calibDbView package

See Figure 5, page 9.

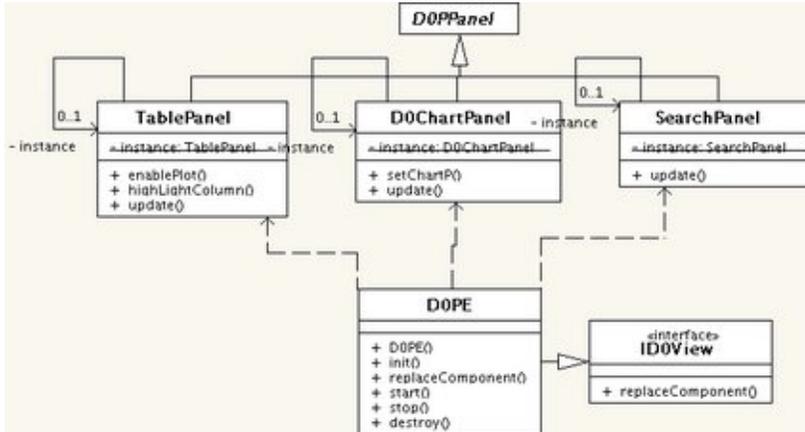


Figure 5: The Class diagram of the calibDbView package

5.1.2 DOPE class

This is a JApplet subclass and implements the ID0View interface. It is the applet instance that is displayed on the webpage and is responsible for creating the singletons⁶ SearchPanel, TablePanel and D0PChartPanel. The DOPE class can also read parameters from the html page that allows it to connect to another Oracle® database.

5.1.3 ID0View interface

This interface defines the method for adding and replacing panels in a windowing environment.

5.1.4 DOPPPanel class

DOPPPanel is an abstract class⁷ that allows the windowing environment to treat all the panels the same. It defines all the common methods and properties of panels

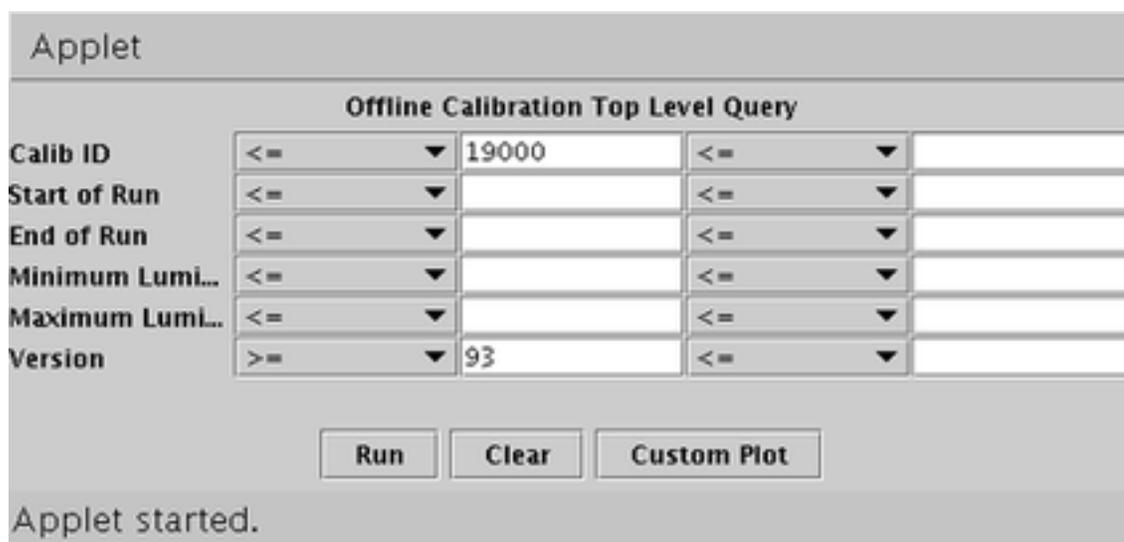
⁶Singleton - only one instance of a class in the whole program (See Appendix E)

⁷A class that does not provide an implementation for one of it's method is abstract

that will be added and displayed. SearchPanel, TablePanel and D0PChartPanel are all subclasses of D0PPanel.

5.1.5 SearchPanel class

This is the initial panel that is displayed. It has fields for the user to enter the initial search parameters. SearchPanel also has a “Custom plot” button that prompts the user for a query which is immediately plotted. See Figure 6.



Offline Calibration Top Level Query			
Calib ID	<=	19000	<=
Start of Run	<=		<=
End of Run	<=		<=
Minimum Lumi...	<=		<=
Maximum Lumi...	<=		<=
Version	>=	93	<=

Run Clear Custom Plot

Applet started.

Figure 6: The search panel

5.1.6 TablePanel class

This panel is responsible for displaying the values returned by the queries in a tabular format. It uses a JTable in a JScrollPane as a view and the D0TableModel as its model(data source). It has a “Drill Down” button for exploring the table and also a plot button. The plot button is initially disabled but gets enabled whenever a plot state is attained.

5.1.7 D0ChartPanel class

This panel basically holds and displays a plot of the data.

Applet								
CID	RUN_START	RUN_END	LUMI_MIN	LUMI_MAX	DMD_COLID	DOCS_D0ID	VERSION	OPT_COM...
18895	178652	9999999	0	0	1	12490	102	
18802	178634	178651	0	0	1	12400	101	
18528	178548	9999999	0	0	1	12213	100	
18437	178458	9999999	0	0	1	12198	99	
18347	178328	9999999	0	0	1	12183	98	
18257	178216	178327	0	0	1	12096	97	
18256	177917	178327	0	0	1	12095	96	
18168	177911	177916	0	0	1	12010	95	
18081	177665	177910	0	0	1	11926	95	
17994	177577	177664	0	0	1	11842	94	
17908	177351	177576	0	0	1	11759	94	
17734	177146	177169	0	0	1	11593	93	
14222	178216	9999999	0	0	1	10016	96	
14134	177917	9999999	0	0	1	10002	95	
14047	177665	9999999	0	0	1	9988	94	
14046	177100	9999999	0	0	1	9987	93	

<< Back Drill Down Plot it

Applet started.

Figure 7: A table panel

5.2 Package calibDbData

The calibDbData package contains both the controller and model of the MVC architecture. The classes in this package handle all the database related actions and other processing actions.

5.2.1 Class diagram of calibDbData package

See Figure 8, page 12.

5.2.2 OracleConnector class

This class is responsible for getting a connection to the oracle database. It has a static method that returns a connection to the specified database.

5.2.3 DataManager class

The DataManager class is the controller for the MVC architecture used in the applet. The model is a state design pattern (See Appendix D) with the DataManager

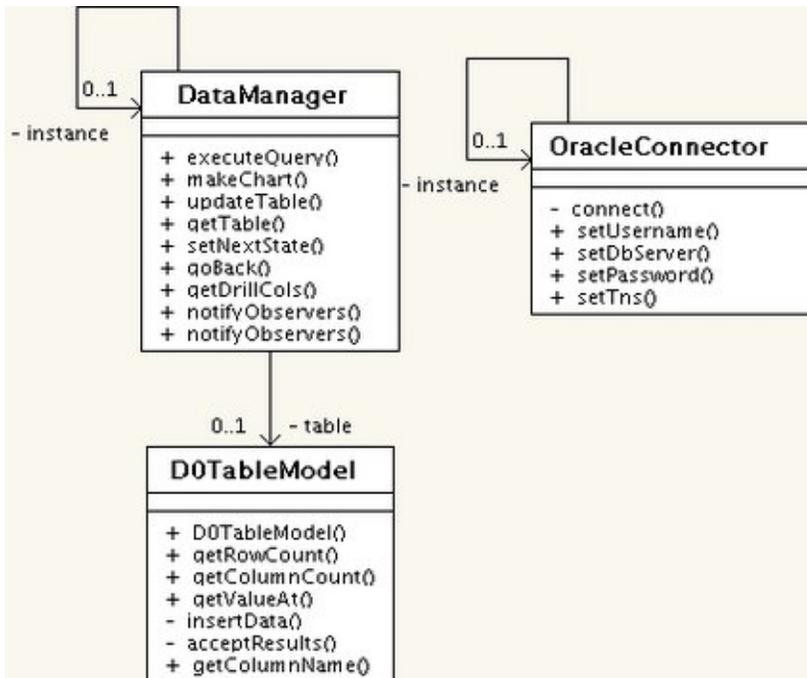


Figure 8: The class diagram of the calibDbData Package

acting as the context. All events from the view are sent to the DataManager, who in turn delegates it to its state for the actual processing. The state constructs the necessary command and sends it back to the DataManager for execution and notification of the view. The notification process is accomplished using an Observer-Observable design pattern (See Appendix H). The DataManager is also responsible for obtaining a connection to the database and executing all queries. It uses the OracleConnector to obtain the connection.

5.2.4 D0TableModel class

D0TableModel is an abstraction of the data being displayed. It is basically a model to the TablePanel class. It stores the results of all queries as a vector of vectors and notifies TablePanel whenever its data changes.

5.3 Package calibDbData.states

This package holds the various states used by the state design pattern of the Data-Manager class. The various states are classes implemented using the template design pattern (See Appendix G). Thus, the invariant behaviour is abstracted into 3 abstract classes and all other states subclass one of the 3 abstract classes.

5.3.1 Partial class diagram calibDbData.states package

See Figure 9, page 13.

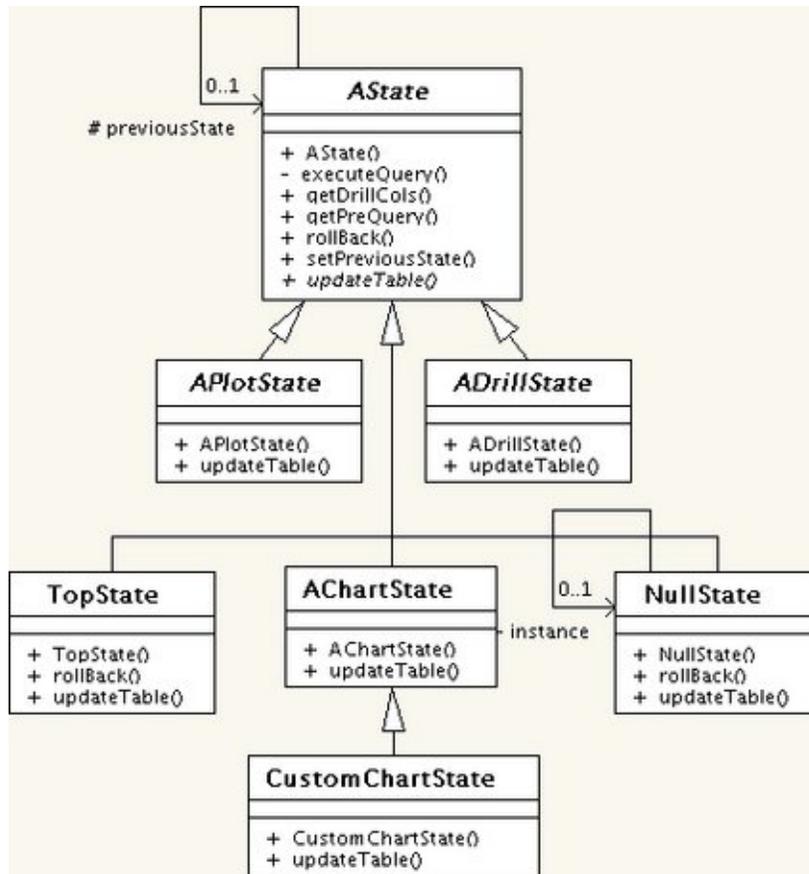


Figure 9: The class diagram of the `calibDbData.states` package

5.3.2 AState class

AState is the parent abstract class and defines methods and properties common to all states⁸ such as a `previousState` property. It has 2 abstract methods that all its subclasses must implement: the `undo()` method and the `updateTable(Object obj)` method.

5.3.3 ADrillState class

ADrillState is an abstract class that defines methods common to states that only drill down the database. It provides an implementation of the `undo()` method and the `updateTable(Object obj)` method for all its subclasses. ADrillState has a `doQuery(Point p)` method that is abstract. See Appendix A for a class diagram.

5.3.4 APlotState class

APlotState is an abstract class that defines methods common to states that are ready to plot. It provides an implementation of the `undo()` method and the `updateTable(Object obj)` method for all its subclasses. APlotState has a `doChart(int[] colIdx)` method that is abstract. See Appendix B for a class diagram.

5.3.5 AChartState class

This class is the state of the DataManager whenever there is a plot on the screen.

5.3.6 CustomChartState class

This state handles events whenever a custom plot is created.

5.3.7 TopState class

This class handles the events for the initial search query.

⁸All states are subclasses of AState

5.3.8 NullState class

This class is a null design pattern class and acts as a default state for the Data-Manager (See Appendix I).

5.4 Package calibDbCommand

The loose coupling of the MVC architecture requires an independent communication mechanism - this is achieved using the command design pattern (See Appendix F). This package contains the classes that make up the command objects.

5.4.1 Class Diagram of calibDbCommand package

See Figure 10, page 15.

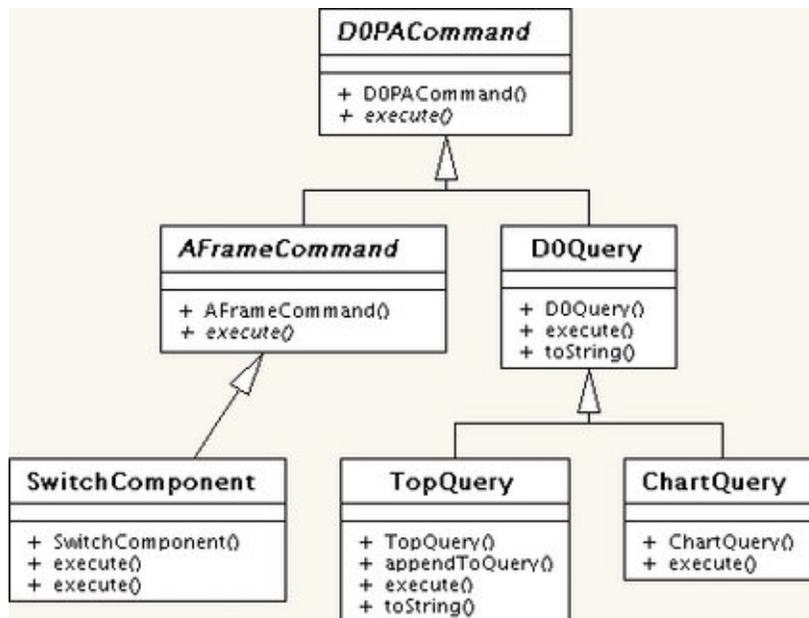


Figure 10: The class diagram of the calibDbCommand package

5.4.2 D0PACCommand class

This is the parent abstract command class. It defines the constructor and the abstract execute method that all subclass must implement.

5.4.3 AFrameCommand class

The program requires that we have a number of different views in one windowing environment. This class provides an abstraction for communication between the views (DOPanels) and the windowing environment.

5.4.4 SwitchComponent class

This is a subclass of AFrameCommand and is basically the command that tells the frame to switch views.

5.4.5 D0Query class

This is the encapsulation of a query. It is constructed by passing in the query as a string. The execution of the query is achieved by delegating it to the DataManager.

5.4.6 TopQuery class

This is a special case of the D0Query. It is the initial search query which needs to be constructed from multiple fields.

5.4.7 ChartQuery class

ChartQuery creates a chart from it's query. It is thus a special case D0Query.

5.5 Plotting Library - JFreeChart

The applet uses a free Java class library to generate the plots. There are quite a number of these libraries available on the world wide web. JFreeChart (<http://www.jfree.org/jfreechart/>) became the library of choice, because it made the generation of a chart from a database easy. JFreeChart also supported features like exporting charts into popular image formats. It provided support for other Java technologies and had good documentation. See Figure 11.

6 Problems with Java Applet

The Java applet satisfies all the design considerations and goals. However, the nature of the technology makes it undesirable in certain situations. Whenever the

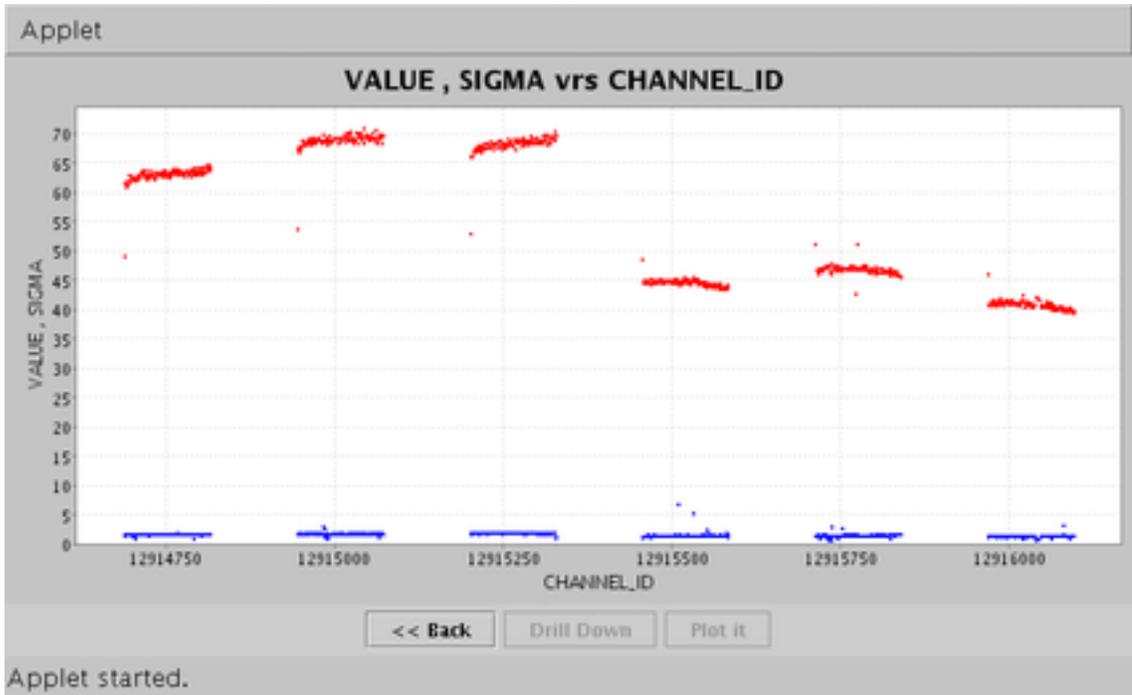


Figure 11: The plot of data using JFreeChart

applet page is loaded into a web browser, the web browser downloads the class files and runs the applet on the users computer. This introduces a number of problems.

- This applet uses two huge libraries which need to be downloaded together with the packages described above. On a slow network, this could potentially be a problem for the user. This problem is partially solved by compressing and storing all the necessary files into one big archive. The Java JAR⁹ archiving technology is used for this.
- In order for the applet to run on the user's computer, the user must install the Java Runtime Environment(JRE) version 1.4 and up. Most browsers come with their own JRE of a lower version and do not support the JAR archive format. This makes the applet unusable until the user downloads the Sun Microsystems' JRE.

⁹JAR archives are basically zip archives

- Applets run in a security sandbox that prevent them from doing harm to a user's computer or using a user's computer to do harm to another computer. This sandbox allows the applet to only connect to its web-server host. Thus the DOPE applet is prevented from connecting to the database server. This problem is solved by digitally signing the applet. At the time of writing, DØ did not seem to have a digital certificate. The applet at the moment is signed with my untrusted digital certificate.

7 Third Attempt - Java Server Pages

The DOPE applet is problematic for one very simple reason; all the processing is done client-side¹⁰. Therefore, moving to a server-side architecture will solve all the applet security problems.

In order not to reinvent the wheel, a java server-side technology was a reasonable choice for the third implementation.

Program Description

The server-side implementation was written using a combination JavaTMServlet[6], Javabeans¹¹ and JavaTMServer Pages technology[2]. JFreeChart has support for these technologies and can thus can still be used as a plotting library. The loose coupling of the MVC architecture allows for easy changing of the view to a web-based view. This was accomplished by writing a servlet class to interface with the DataManager and various java server pages to take user input and display the data. The packages used in the server-side implementation are described below.

7.1 Package beans

This package holds the javabeans used in the server-side implementation.

¹⁰Client-side means all the work is done by the user's computer as opposed to server-side where the server does all the work

¹¹Javabeans are regular Java classes designed according to a set of guidelines [2].

7.1.1 DrillBean class

This bean encapsulates the state information for the server-side.

7.1.2 TableBean class

This bean is an interface to the D0TableModel. It is basically an encapsulation of the data being displayed.

7.1.3 SearchBean class

This bean is responsible for capturing the initial search parameters and generating a TopQuery.

7.1.4 ChartBean class

The ChartBean builds the query to be plotted and extracts the labels of the axis and title from the query.

7.2 Package calibDbServlets

This package holds all the servlets that interface with the DataManager.

7.2.1 SearchServlet class

This servlet uses the SearchBean to update the D0TableModel with the results of the initial search query.

7.2.2 DrillServlet class

This servlet handles all drilling using the DrillBean.

7.2.3 DrillServlet class

This servlet uses the ChartBean to configure the properties of the chart. It then delegates the creation of the plot to the displayChart.jsp page.

7.3 Package calibDbData.states

This package was extended to get the servlets working with the state design pattern used in the DataManager class.

7.3.1 IServletState interface

This interface defines the methods used by the servlets to determine state information and queries. AState class now implements this interface and thus all states can be used by the servlets.

7.4 Package calibDbData

JFreeChart's jsp specification requires that the data to be plotted is created by a class that implements the de.laures.cewolf.DatasetProducer interface.

7.4.1 CalibDatasetProducer class

This class implements the DatasetProducer interface. It acts as the data source for displayChart.jsp.

7.5 Java Server Pages(JSP)

There are 4 jsp files three of which constitute the view. The view files are displayTable.jsp, displayPlotTable.jsp and displayChart.jsp. Their functions can be deduced from the file names.

The other jsp file, processForm.jsp, is responsible for passing the values of the search parameters to the SearchBean. The result of the server-side implementation can be seen in Figures 12, 13 and 14.

CID	RUN_START	RUN_END	LUMI_MIN	LUMI_MAX	DMD_COLID	DOCS_D0ID	VERSION	OPT_COMMENT
19555	179324	9999999	0	0	1	12748	108	
19554	179324	9999999	0	0	1	12748	107	
19553	179324	9999999	0	0	1	12748	106	
19552	179324	9999999	0	0	1	12748	105	
19551	179324	9999999	0	0	1	12747	103	
19550	179324	9999999	0	0	1	12746	101	
19549	179324	9999999	0	0	1	12746	100	
19548	179324	9999999	0	0	1	12746	99	
19547	179324	9999999	0	0	1	12745	97	
19546	179324	9999999	0	0	1	12745	96	
19545	179324	9999999	0	0	1	12745	95	
19458	179193	9999999	0	0	1	12731	107	
19457	179193	9999999	0	0	1	12731	106	

Figure 12: A page returned by the server-side implementation

D0ID	SMTID	CFTID	CPSID	FPSID	CALID	MDTID	MSCID	PDTID	OPT_COMMENT
12746	132	136	181	-1	-1	2	15	11	

Figure 13: Server-side with multiple drill down columns



Figure 14: Server-side plot of data

8 Conclusion

Three methods of browsing the $D\emptyset$ calibration database have been described in this paper. The MISWEB implementation was not very useful. The Java applet implementation is a great success and is available for users who have a high bandwidth and are willing to download the Sun Microsystem JRE. The server-side implementation was built on top of the applet implementation with a few modifications. This system works but due to time constraints, it has not been very well tested. It also has the disadvantage of not being optimized for server-side environment since it's based on client-side applet code. Optimizations that can be done include multi-threading, database connection pooling and caching.

The goals of the project have been achieved. There are now two web-based methods for users to browse the $D\emptyset$ calibration database.

9 Acknowledgement

I would like to thank my supervisor, Taka Yasuda, for his insights, assistance and support throughout this project. I would also like to thank Geoff Savage who helped me in the absence of my supervisor. Finally, I want thank Dianne Engram, Dr. McCroy, Dr. Davenport and all the members of the SIST committee for the opportunity to work at a great research institute - Fermilab.

References

- [1] Sinan Si Alhir. *UML in a Nutshell*. O'Reilly & Associates, Inc., Sebastopol, 1998.
- [2] Hans Bergsten. *JavaServer Pages*. O'Reilly & Associates, Inc., Sebastopol, second edition, 2002.
- [3] Chris Fehily. *SQL: Visual Quickstart Guide*. Peachpit, Berkeley, 2002.
- [4] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns Elements of Reusable Object Oriented Software*. Addison-Wesley, Massachusetts, 2000.
- [5] Mark Grand. *Patterns In Java: A Catalog of Reusable Design Patterns Illustrated With UML, Volume 1*, volume 1. Wiley, Indianapolis, 2002.
- [6] Jason Hunter and William Crawford. *Java Servlet Programming*. O'Reilly & Associates, Inc., Sebastopol, second edition, 2001.
- [7] Stephen Wong. Java resources: Design patterns. <http://exciton.cs.oberlin.edu/javaresources/>.

APPENDIX

A Class Diagram of subclasses of ADrillState

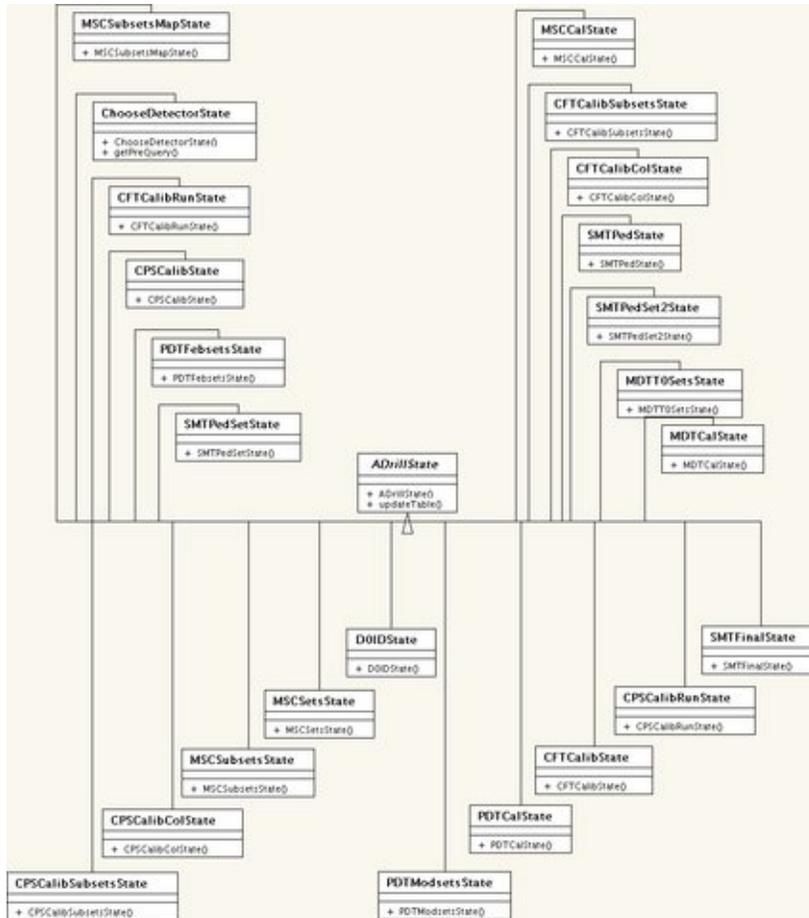


Figure 15: Class Diagram for subclasses of ADrillState

B Class Diagram of subclasses of APlotState

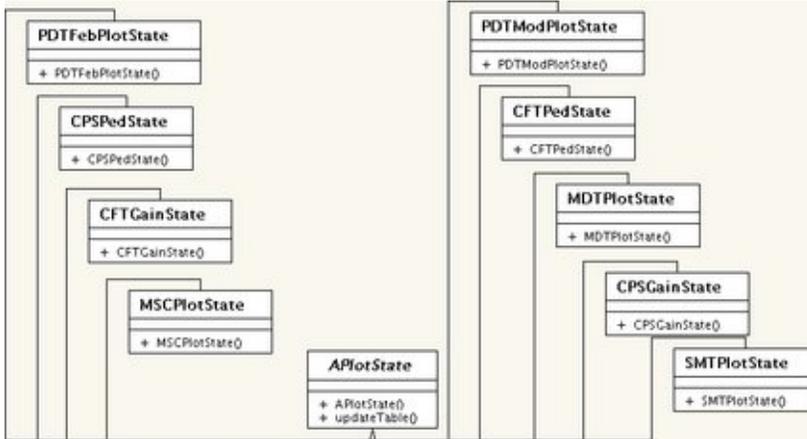


Figure 16: Class Diagram for subclasses of APlotState

C Model-View-Controller Design Pattern

A program can often be divided into two parts:

- The *Model*: the internals of the program where the actual processing takes place.
- The *View*: the inputs and output of the program which the user interacts with.

The model-view-controller design pattern decouples the model from the view enabling loose coupling and the ability to change one without affecting the other. Thus each part should be able to operate totally independent of the other. The controller is responsible for instantiating the two parts and the adaptor that connects them together[7]. See Figure 17.

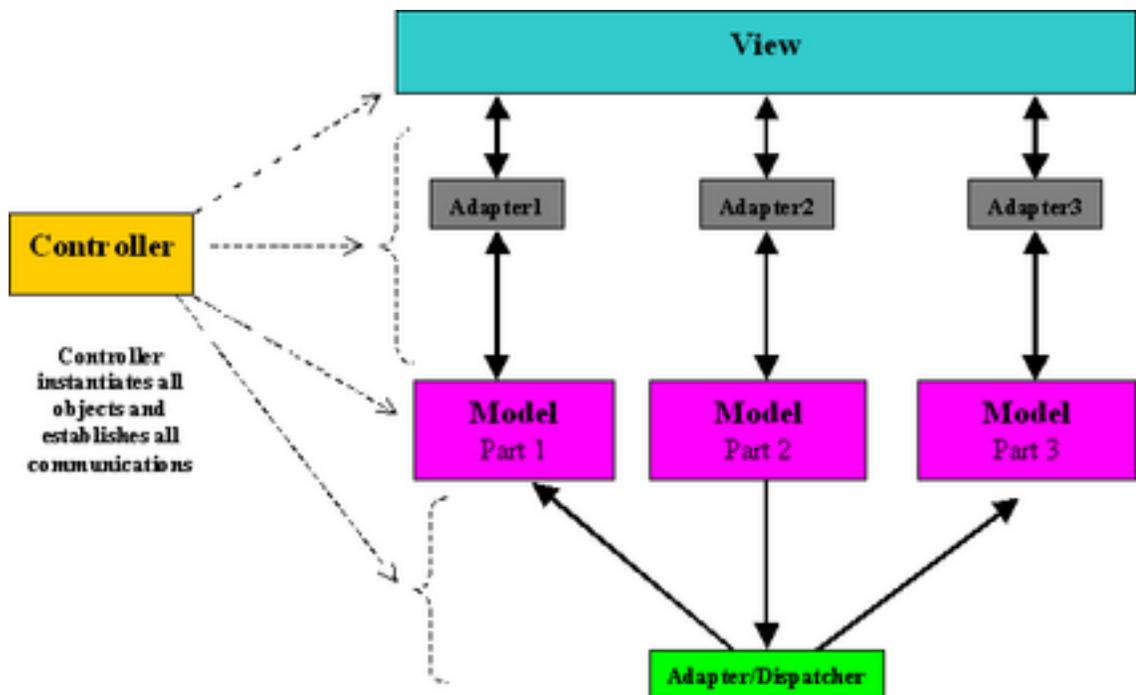


Figure 17: The model-view-controller

D The State Design Pattern

At any given time, an object can be described as being a state due to the value of its properties. The particular values of the properties affect the objects behaviour. By using the state design pattern, behaviours that depend on the state are simply delegated to the state. The state design pattern encapsulates the state of an object as discrete objects, each belonging to a separate subclass of an abstract state class.[5] See Figure 17.

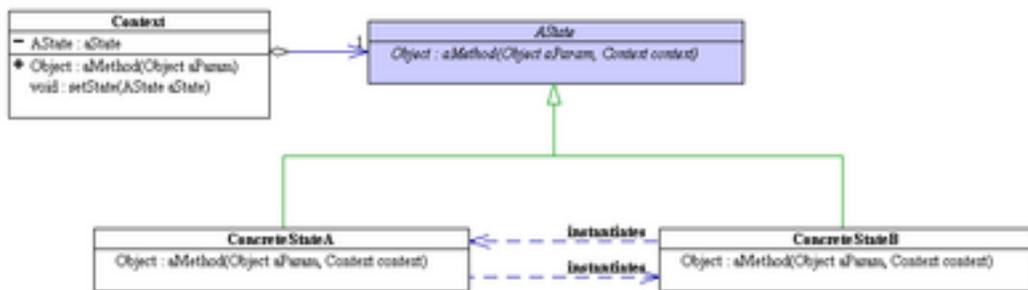


Figure 18: The Class Diagram of the State Design Pattern

E The Singleton Design Pattern

There are situations where only a single instance of an object is required. The singleton design pattern makes the object responsible for creating and keeping track of its single instance.[7] It provides a single global way for all clients to get access to this instance.

See Figure 19.

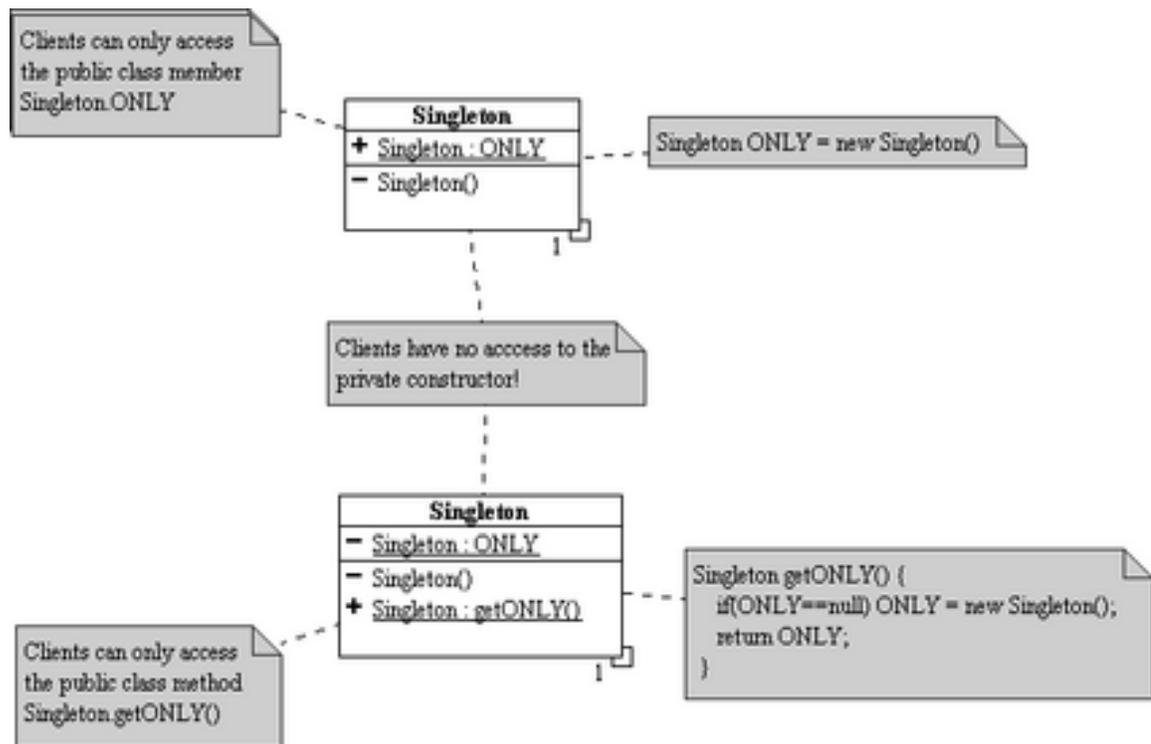


Figure 19: Class Diagram of the Singleton Design Pattern

F The Command Design Pattern

Communication between two objects is often one object telling the second object to perform a particular function. The command design pattern encapsulates the particular request into an object so that the programmer can control their selection, sequencing, queue them, undo them, and otherwise manipulate them.[5] See Figure 20.

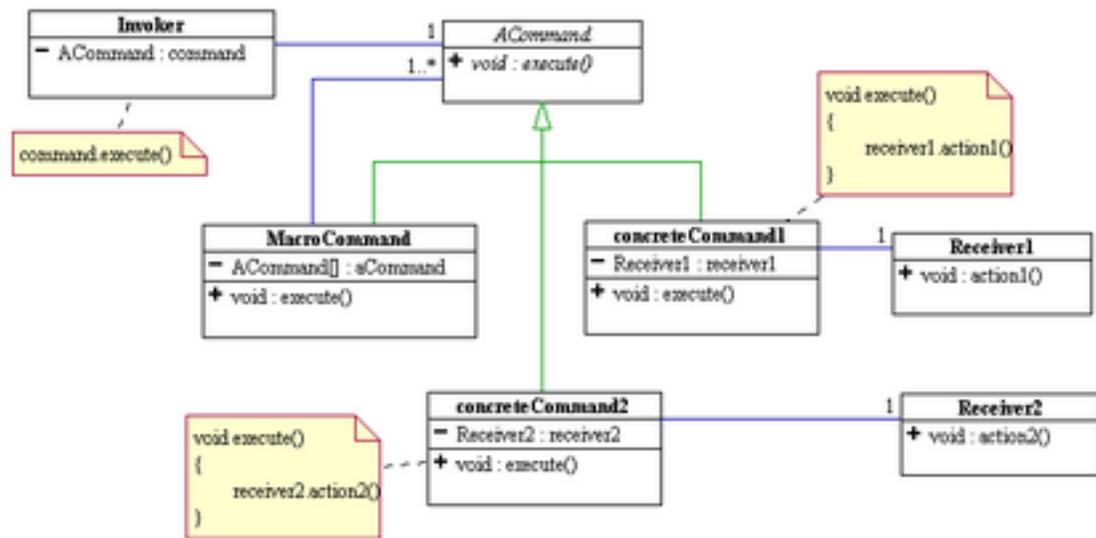


Figure 20: Class Diagram of the Command Design Pattern

G The Template Design Pattern

The template design pattern is used to setup the outline of an algorithm. It enables the programmer to separate variant behaviour from invariant behaviour.[4] See Figure 21.

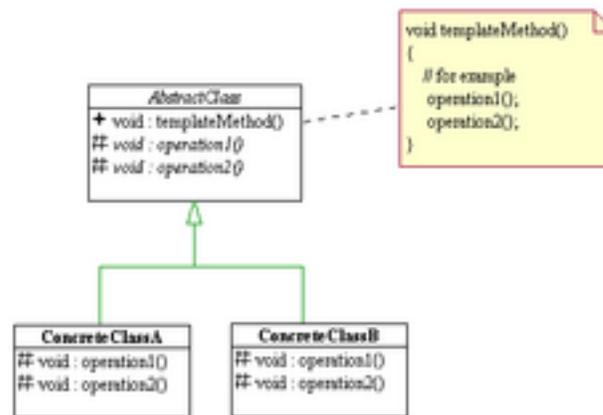


Figure 21: Class Diagram of the Template Design Pattern

H The Observer-Observable Design Pattern

This pattern defines a one-to-many relationship between objects so that when one object(the observable) changes its state, all its dependents(the observers) are notified and updated. The pattern allows the observers to dynamically register their dependencies.[5] See Figure 22.

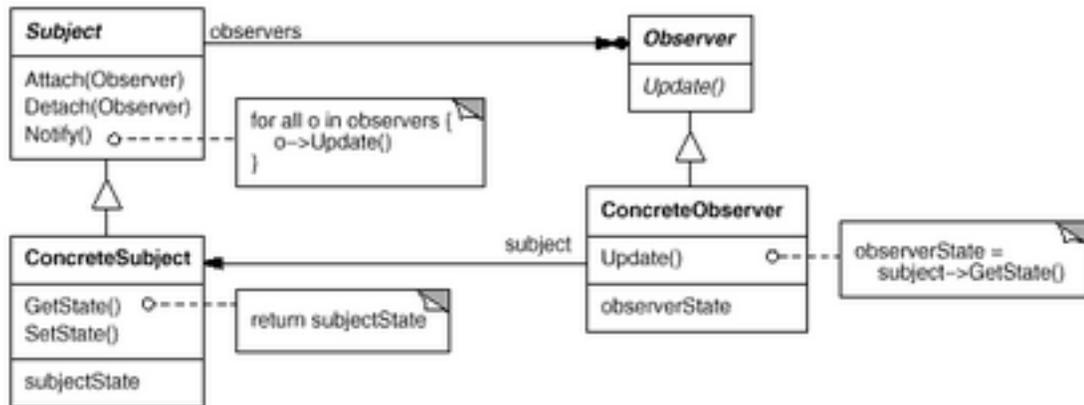


Figure 22: Class Diagram of the Observer-Observable Design Pattern

I The Null design Pattern

The null design pattern is used to indicate the absence of an object to delegate an operation to.[5] A null object knows what to do every time an operation is delegated to it: nothing.