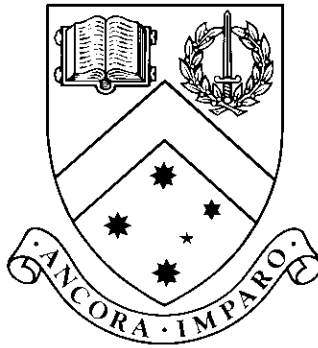


Interfaces Supporting Knowledge Discovery in Data

by

Mark Hollands



Thesis

Submitted by Mark Hollands

in partial fulfillment of the Requirements for the Degree of

Bachelor of Software Engineering with Honours (2770)

in the School of Computer Science and Software Engineering at

Monash University

Monash University

November, 2003

© Copyright

by

Mark Hollands

2003

Contents

List of Figures	v
Abstract	vi
Acknowledgments	viii
1 Introduction	1
2 Literature Survey	3
2.1 KDD Process	3
2.2 User-centered Design	4
2.3 Human Computer Interaction	4
2.4 Interface Techniques	5
2.5 Applying Interface Techniques to KDD	6
3 Methods	8
3.1 Snob	8
3.1.1 Snob-vanilla	8
3.2 Requirements of the system	9
3.2.1 Web Application	9
3.2.2 Interactivity	9
3.3 Interface Layering	10
4 Implementation	12
4.1 System Specifications	12
4.1.1 Consistent requirements	12
4.1.2 Client-side Structure	14
4.1.3 Volatile requirements	14
4.2 Module Implementation	18

4.3	Interpretation	18
4.3.1	Interpretation Functionality	18
4.3.2	Interpretation Implementation	19
4.4	Visualisation	19
4.5	Testing Procedures	21
4.5.1	Usability Testing	21
4.5.2	Stage #1	23
4.5.3	Stage #2	23
4.5.4	Stage #3	23
5	Results and Discussion	24
5.1	Stage 1 Results	24
5.1.1	Introductory Questions	24
5.1.2	Interface Questions	25
5.1.3	Header Bar Questions	26
5.2	Stage 2 and 3 Results	26
5.2.1	Interpretation Discussion	26
5.2.2	Visualisation Discussion	27
6	Conclusion	28
6.1	Recommendations for Future Work	28
	References	30
	Cgi Script Module	32
	Snob Server Module	90

List of Figures

3.1	Flowchart of the system interface architecture	11
4.1	Sitemap for Snob-Online	15
4.2	Header bar in project state	16
4.3	Header bar in data state	16
4.4	Header bar in session state	16
4.5	Command input interface	17
4.6	Interpretation class heirarchy	20
4.7	Color Shading Visualisation	22

Interfaces Supporting Knowledge Discovery in Data

Mark Hollands, BSE(Hons)
Monash University, 2003

Supervisor: Associate Professor Trevor Dix

Abstract

This thesis examines the amalgamation of technologies used to interface the data mining system, Snob, with the internet. The system Snob-Online was developed, providing an interface focused upon the Knowledge Discovery in Databases (KDD) process. Usability testing has been carried out to determine the effectiveness of the web interface, for enhancing the usability and knowledge discovery of the system.

The KDD process is examined and an explanation of how user-centered design techniques can assist in the development of data mining systems is given. Discussion of the system architecture highlights the design decisions that justify the selection of technologies which were used to create Snob-Online. The implementation of this system is described by examining the control flow of Snob-Online and how it is mapped to the KDD process. Based upon the output from Snob, result interpretation and visualisation functionality was implemented in Snob-Online by utilizing an XML interface.

Interfaces Supporting Knowledge Discovery in Data

Declaration

I declare that this thesis is my own work and has not been submitted in any form for another degree or diploma at any university or other institute of tertiary education. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Mark Hollands
November 10, 2003

Acknowledgments

I would like to thank everyone who supported me, particularly my supervisor Associate Professor Trevor Dix, whose guidance throughout this year helped steer me on the right path.

Mark Hollands

Monash University
November 2003

Chapter 1

Introduction

Data mining is defined as "the exploration and analysis by automatic or semi-automatic means of large quantities of data in order to discover meaningful patterns or rules" (Berry and LinHoff, 1997). It first received recognition as a significant field of research during the early 1990s. Considerable commercial interest in extracting patterns from large real world data sets has seen it become one of the most active research fields in modern computing. Data mining has combined, and built upon, research borrowed from fields such as pattern recognition and machine learning, towards its aim of knowledge discovery.

The term *knowledge discovery* implies more than just mathematical techniques applied to real world data. Without successfully imparting this discovered knowledge to the user, data mining is a pointless exercise. Towards this end, the Knowledge Discovery in Databases (KDD) process has endeavored to build a multidisciplinary framework that aims to support the user's interactions with the underlying mathematical data mining system.

"KDD aims to provide tools to automate (to the degree possible) the entire process of data analysis and the statistician's *art* of hypothesis selection" (Fayyad, Piatetsky-Shapiro and Smyth, 1996). Some of these tools solve specific problems such as: how the data is initially accessed from the source; how algorithms can be scaled to massive data sets and still run efficiently; how results can be interpreted and visualized and a number of other aspects for generally assisting man-machine interactions to make the data mining environment conducive to user learning.

Historically data mining systems have been primarily focused upon the mathematical algorithms that are used to examine the data. In certain cases this has been at the demise of the system as a whole, causing the system to be difficult to use or ineffectually presenting the discovered knowledge to the user. Data mining systems should have a primary focus upon the KDD process throughout the design and development phases of their development.

Snob is a data mining system which carries out data classification using Minimum Message Length (MML) criteria for optimization. It is a highly efficient program which can classify thousands of records, each with tens of attributes. Irrespective of its algorithmic prowess, Snob is by no means easy to use, with some experienced users expressing that they still needed to keep the user manual close at hand to interpret the results.

The primary goal of this research is to present the data mining system, Snob, to a broader audience by making Snob accessible on the internet and improving its user interface. This project will examine the most effective amalgamation of web technologies to fit with the inherent system constraints that Snob presents. A User-centered design methodology will be applied throughout the system development lifecycle. Usability testing will be carried out to establish the effect upon system usability and potential knowledge discovery.

The system extension, Snob-Online, will be designed with a KDD focus which will govern the structure of the functionality in the system. The functionality envisaged in this system will include session and data handling to manage user interactions, and result interpretation and visualisation achieved through the use of an XML interface.

Chapter 2 first details the individual stages of the KDD process, and how data mining fits within the KDD process. This is followed by an exploration of the purposes of the KDD process. Then the concepts involved in User-centered design and human-computer interfacing, are examined and shown to be well suited to practical implementation of the various stages of the KDD process. Chapter 3 defines the chosen architecture for Snob-Online and justifies the design decisions in relation to the requirements of the system. The implementation of Snob-Online, as a KDD focused, data mining environment is discussed in Chapter 4. The testing methodology and results are discussed in Chapter 5. The limitations and potential further work are explored in Chapter 6, before the conclusion in Chapter 7.

Chapter 2

Literature Survey

2.1 KDD Process

The process is iterative in nature; the system begins with low-level real world data sets, combined with user interaction; this is then refined to become high-level knowledge and hopefully understanding on the part of the user. The KDD process can be generally described as (Robinson and Shapcott, 2002):

1. Determining what problem is to be solved, i.e. what domains are to be explored. These constitute hypotheses that the data mining process is being conducted to test.
2. Creation of the relevant data set for mining, from the source database.
3. Pre-processing of the data set, dealing with missing values and/or invalid data.
4. Performing data reduction upon the data set, to simplify or change the domain of the model to be generated. This step is particularly useful after the first iteration in refining the data set to produce a more accurate model.
5. Data mining, using the system to build the required model from the data.
6. Analysis of the model with potential use of visualisation against the hypotheses. This may be coupled with returning to any of the previous steps for further iteration. Refinement of the domain of the model being processed based upon the information provided by the DM is part of assisting the user to gain a better understanding from a more comprehensive model.
7. Acting upon the discovered knowledge, in using the knowledge directly, in conjunction with another system, or merely documentation and reporting the knowledge.

Different researchers have classified the KDD process in various ways, combining or elaborating upon certain steps, yet the underlying process remains essentially the same. For example, Fayyad et al. (1996) gives a process where the data mining (Step 5) is expanded out to a series of sub-steps covering matching the goals of data mining (Step 1) to a particular data mining method and selecting specific algorithms to use in the data mining step.

2.2 User-centered Design

The goal of the KDD process is to assist the user's learning towards knowledge discovery. To achieve this goal it is essential to understand the ways that users interact and perceive data mining systems. These principles can be practically applied under the model of software design known as User-Centered Design (UCD). Historically, computer aided learning systems have used certain aspects of UCD to assist the usability of their systems (Lewis, Brand, Cherry and Rader, 1998).

UCD is a software development methodology comprising of three main principles (Gould and Lewis, 1985). The first of which is the way that UCD is primarily concerned with the user's perspective of the system. It involves controlled collection of data from the users regarding their goals and subsequent actions within the system. This involvement should begin early in the software development lifecycle. The second principal is gathering empirical measurement of usage of the system. This can be carried out in the form of usability testing which, combined with feedback from the users, forms an important source of information for further direction of the software. Iterative design is the third principal of UCD software development. The usability goals gathered from interactions in the previous two points are repeatedly assessed through iterations in the development process of design, implementation and testing (Alexander, 2003).

The principal of iterative development can be seen to be very similar to the rapid prototyping model of Software Engineering. Gulliksen, Lantz and Boivie (1999) have stressed while this can provide invaluable insight in the UCD process it can also lead to *functionality creep* if it is not successfully managed.

2.3 Human Computer Interaction

The process of human-centered system design subdivides a task between a machine and a human (de Figueiredo and Lai, 2000) :

- The machine acts on computationally intensive sub-tasks.
- The human acts on sub-tasks that require complex perception.
- The human-machine interface reconciles the state of the machine with the state of the human.

This division of tasks relates very well to the interactions between a user and a data mining system. Implementation of a data mining system modeled upon the KDD process seems to lend itself naturally to the principals of UCD. The need for enhanced user interactions in an essentially task-(or job-) orientated system matches well with the structured usability testing focused upon users and their tasks. To successfully *reconcile* the gap between the state of the machine and the state of the human, the language in which the two communicate must be considered.

Basden (1996) has examined data mining systems from the point of view of knowledge base creation. Previously two forms of knowledge have been predominantly used in knowledge base construction. In the first, knowledge is drawn from a domain expert, conceptualized and then assembled into a knowledge base through the use of knowledge representation language and software. The second technique, commonly found in neural networks and case based reasoning, elicits knowledge through rule based induction. He explores that the type of creative thinking used in the KDD process is instantiating a third type of knowledge base creation. It differentiates itself from the two main techniques because the source of much of the content is created by the design process itself. He hypothesizes from this idea that the existing user interface conventions, commonly found in API libraries present in "Windows" computing, are stifling the creative design process by limiting the extent to which a user can be immersed in their task. The interface applications of this hypothesis are discussed further in section 4.

This fits with the thoughts of interface expert, Norman (1988), who speculates that "The best computer programs are the ones in which the computer itself *disappears*, in which you work directly on the problem without having to be aware of the computer."

This concept of making the computer *disappear* is attempting to bridge the gap between the user and computer, or the human - machine interface. When a user is immersed within the environment that the computer provides, the practical limitations of the computer are minimised.

It is important that the environment that the computer provides, maps well to the way that the user visualises the problem. If the environment and user expectations are well mapped then there is a better potential for the user being immersed in the environment.

2.4 Interface Techniques

The interface modelling technique of mapping an environment to a user's expectations is known as a conceptual model. A conceptual model assists a user in understanding the way a system works by associating functionality with an existing metaphor they would be familiar with. This familiarity if used well can provide the effect of immersing the user in their task. Due to the iterative and somewhat complex nature of the KDD process, finding a suitable conceptual model is by no means a simple task.

Chattratchat, Guo and Syed (1999) have applied an iconic visual programming technique with graphical visualization techniques to create an internet-based framework for convenient and effective data mining. They model the concept of work flow using the established convention of arranging and linking graphical icons. Together with the use of threading and multi-tasking, they have explored ways to maintain a level of interactivity with the user despite the constraints placed upon the system by deploying over the internet.

Robinson and Shapcott (2002) have expanded upon the concept of work flow, experimenting with a highly graphical representation of data flow attempting to move data visualisation past the classical *charts and graphs*. Using 3d graphics techniques common in computer gaming, they model the concept of water flowing through pipes to

describe data *flowing* through the data mining system. Their model's inability to scale to datasets with large numbers of attributes is one of the limitations of an otherwise innovative approach.

Using the creative knowledge discovery principal, a conceptual model focused upon a continual user process was created (Basden, 1996). The user is engaged in a consistent stream of thought that must not be interrupted. Through the use of interface concepts, the system allows the user to explore; using tentative action, to try things; discerning the change in system state on the fly. This is used to stimulate a deeper understanding for the model.

2.5 Applying Interface Techniques to KDD

Applying the UCD iterative design processes this project will develop a user interface for the existing data mining system, Snob. Usability testing will measure the effectiveness of system development towards the goal of developing a successful conceptual model based upon the KDD process within the constraints of an internet-based architecture.

Specifically, it will attempt to address the following aspects across the KDD process:

1. Determining the goals and hypotheses to be tested by the KDD process. Potentially outside the scope of a data mining system since it is essentially a purely human pursuit.
2. Creating the relevant data set for mining from the source database. Provide means for the user to easily enter the dataset and label the meaningful attributes.
3. The pre-processing phase found in KDD has given rise to the commercial trend of data warehousing, which has become a field in its own right (Fayyad et al., 1996). This refers to collecting and cleaning data for ease of use in potential further projects. It has become increasingly popular from a business perspective to have data collected in an easily accessible format, to enable exportation to a range of potential data mining practical uses. By storing data in an easily accessible format, such as XML, this does not place undue constraints upon how the data can be used subsequently.
4. Limiting the domain of the system is most often carried out during the subsequent iterations of the KDD process. It is essential that after completing one iteration of the system, users do not become lost when they are forced to return to an earlier point and modify previous choices. Maintaining a consistent interface throughout the KDD environment can minimize the chances of the user becoming lost.
5. How efficiently data mining algorithms can be scaled, when used upon massive real world datasets, can assist the KDD process by cutting down execution time and giving accurate estimates as to how long the system will take to accomplish a task.
6. Visualisation and interpretation of results play a large role in assisting the user in assessment of the active state of the system. Based on this information they will be making decisions regarding further iterations through the KDD process.

Hence it is essential that they are presented with an informative and up to date picture of the model state.

7. Providing facilities to allow the output of knowledge to other applications or documentation facilities to support later data mining exploration. This can support the user in acting upon the knowledge gained from the KDD process.

Chapter 3

Methods

This chapter will detail the design decisions that went into the development of the architecture that forms Snob-Online. First the constraints that exist due to the underlying system, snob, will be examined. These constraints will be used to formulate the goals of the system extension. These goals, first discussed, will be used to justify the specific selection of web technologies that form a layering of interfaces which coordinate to implement this system.

3.1 Snob

The practical aspect of this project was to create a internet-based, KDD-focused, graphical environment to extend the command line driven, data mining system, Snob. Snob was first developed in 1968, and has since enjoyed a long history within the Computer Science department, and subsequently school of Computer Science and Software Engineering (CSSE). Originally coded in the Algol, Snob has been ported to a host of different language including the Fortran and C programming languages.

Previously, the use of Snob was limited to members of the CSSE Department who had been granted access. One of the major purposes of this extension is to allow a broader audience to access Snob via the internet. For this project, snob-vanilla was selected as it provides the highest levels of stability and functionality.

3.1.1 Snob-vanilla

Attempting to extend any system, first requires a thorough understanding of the underlying program's structure. Practically snob-vanilla is a data mining application, written in the C programming language, which is intended for use upon Unix systems. Snob-vanilla takes data inputs from two plain text files, which together comprise all the necessary information regarding a dataset, that is required for processing. The vset, or variable set, input file defines the number of attributes that each data item contains and the type of each attribute, as either a continuous variable or a member of a discrete set. The samp, or sample, input file contains all of the records which makes up the dataset.

The system primarily takes user inputs through a command line interface, but it also has limited functionality to accept input directly from the file system. The option of accepting user inputs from a file was initially investigated during the process of assessing the various system interfaces, from which the extension architecture could be based upon. This method was later abandoned due to the unrobust nature with which the system functioned while operating in this mode and the inflexible nature of the interface itself.

When taking input from the command line, the system operates in an interactive fashion; first prompting for input, processing the received command and outputting the results directly to the screen. Some inconsistency exists in this interactive processing model, in that commands which detail the class membership of individual records, are capable of having an output many pages in length depending on the size of the dataset. These commands output their results to an output file and only send acknowledgement of the command to the visible output.

3.2 Requirements of the system

User-centered Design principals are based upon involvement from the users throughout the development lifecycle. Consultation with end users provided invaluable insight into the ways in which snob-vanilla is used and what functionality is desired in the system extension. Initial interviews with previous users of snob-vanilla showed that despite their experience with the system, few users were capable of using the system without having the user manual on hand at all times. The primary desire of end users was seen to be able to provide easy visualization based upon the results of the system.

3.2.1 Web Application

As one of the primary goals of this extension was to present Snob to a wider, more general audience, the internet is the ideal medium to interface with. The internet provides a universal medium and users have a high level of familiarity with html interfaces. The aims in this regard were that the resultant system provide a high level of browser cross-compatibility, with a minimum of additional software required for use of the core of the system functionality.

3.2.2 Interactivity

Due to the iterative and interactive nature of the KDD process, it was seen as essential that the underlying interactivity present in snob-vanilla was maintained in the extension, Snob-Online. To try and distort the underlying system may result in the creation of an incorrect conceptual model, which would result in further usability problems. Yet from discussions with users the current method of interaction was seen to be too inflexible, with users being required to maintain a constant connection to the snob process to maintain the system use. The conception of the notion of a Snob session was to allow the user flexibility in the manner in which they interacted with the system. By delegating the handling of sessions to the system, users would be able to issue commands to the system, log out from the system and then return to resume the session from when Snob

had finished processing their commands. This expands upon the notion of interactivity, while still allowing users the same level of control that is available in the underlying system.

3.3 Interface Layering

As a web application, it will be necessary that Snob-Online be capable of interacting with multiple users simultaneously. Therefore it was necessary that the system is capable of handling multiple sessions, each connected to a separate process of Snob-vanilla. Because of the desired flexibility needed to achieve the attribute of interactivity for the client-side of the system a decision was made to separate the architecture primarily based upon user interface and Snob session handling.

To address this issue of portability in the client-side of the system, several technical solutions were appraised to find the most apt method of providing the desired functionality. The options of using Java in either a dedicated client or applet form, were discussed but were seen to raise more technical issues with language consistency and requiring of more software downloads than was seen to be necessary. An interface mainly comprised of html forms, driven by a server side scripting language was seen as an effective way to maintain the system's flexibility and portability, while not requiring any further downloads on the part of the user. While the use of html forms limited the extent to which the user interface could successfully deliver complex visualisation, it was seen that visualisation is an end point for the KDD process. By providing an portable interface for the results of Snob, a range of visualisation packages could be used by coercing the Snob results to fit the inputs of a 3rd party system.

The chosen architecture for Snob-Online consists of four separate modules, which coordinate their actions to provide a high level of flexibility in both the system design and functionality that is allowed:

1. The interface between the client and server side of any internet-based application forms an important distinction for how the control code of the system executes. The user interface is the main aspect of the system that is present upon the client side of the system. There is some minor JavaScript present on the client side which interacts with server-side to present a more dynamic interface.
2. The majority of the system control code and complexity is contained within the cgi scripts that form the next level of interfacing. This code is responsible for managing the user and data aspects of the system. Using socket connections, it sends commands to the snob server to be executed. Finally it reads the output of snob from the file system and interprets the output.
3. A daemon process running on the webserver manages socket connections from the cgi scripts. Based upon these inputs it creates and subsequently manages the snob instances and sends input into these sessions.
4. A process of snob-vanilla executing upon the webserver, with standard input coming from the snob server and output redirected to the file system.

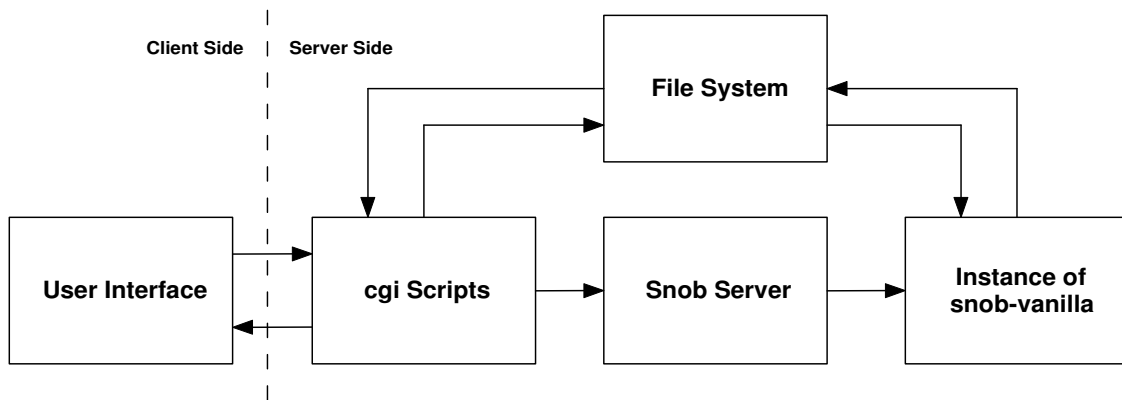


Figure 3.1: Flowchart of the system interface architecture

Chapter 4

Implementation

This chapter begins with a detailed description of the system specifications and how these were used to structure the design of the system functionality. This is followed by the technical implementation of the architecture discussed in the previous section; first detailing the languages selected and then describing the techniques used to create interpretation and visualisation functionality based upon the output from Snob. This chapter is concluded with a description of the testing procedures that were used to gather the results that measure the effectiveness of this system.

4.1 System Specifications

As the underlying system, Snob, is ever undergoing functionality updates it is essential that Snob-Online is designed to ensure that future updates do not cause the system to cease functioning. Therefore the design should try to be as general as possible by minimizing the aspects in which it relies upon the underlying system. System analysis using the principals of UCD, can achieve the desirable attribute of flexibility in system design. Yet it is can be difficult to have a design which is flexible in every aspect. The cost of maintaining a system is considerably reduced if functionality updates do not need to change the system framework(Lassing, Rijsenbrij and van Vliet, 2002).

Therefore it is important to predict which aspects of the system are unlikely to experience change, and formulate the framework of the system based upon these stable characteristics. The uncertain aspects of the system can be encapsulated, so that they may be readily changed or extended as requirements are eventually altered as circumstances dictate. These predictions can be used to order the system design based upon which functionality has been predicted as unlikely to change(Witt, Baker and Merritt, 1994). This ordering is helpful in developing a system framework which is comprised consistent functionality.

4.1.1 Consistent requirements

The following consistent requirements upon which the system is structured are now discussed.

Users

The top level division is based upon the need to provide user authentication for security of data they are providing, which may be of a sensitive nature. This authentication is provided by a username and password combination which must be entered to progress into the site.

Projects

Each user is able to maintain multiple projects within the system. This is used to separate information lower down in this heirarchy and provide a clear distinction about what is currently being worked upon. Once a user has logged into the system, they are presented with a choice of the current projects which they have created. If the user has no projects currently open, they are automatically redirected to the new project page.

Data

Each project can maintain multiple datasets, as a user modifies the domain of their project, potentially adding or removing attributes from the dataset. When a user selects which project they wish to work upon, they progress onto the data stage of the system. This section functions in a similar manner to the project selection functionality, as they can select from a list of current datasets or choose to add a new one to the list.

Sessions

The final section of system structure is where actual interactions with the underlying system are carried out. Each project can have one active process at any point in time. Once a session is opened, the user selects Snob commands to process the active dataset. After each command is selected, the user moves to the session results interface. This page displays the results for all the commands which have been processed during this session. When a user has finished with a particular session, they can select to end the session. When a session is closed it is archived so a user can return and examine previous results, but no further commands can be added to a session once it has been closed.

Persistence

It is possible that a user can set the system to process a command and logout from the system while the session is still active. When the user returns to the system, this session will still remain active and the user can continue as normal. It is possible to command snob-vanilla to create a checkpoint file. That can be reloaded at a later stage in that session or in a different session to return the system state to when the checkpoint file was created.

4.1.2 Client-side Structure

The four consistent requirements discussed are directly reflected in the system sitemap shown in figure 4.1. All direct user actions are reflected by the solid black arrows which connect the various screens within the system. The dotted arrow up the right hand side of the diagram represents the iterative nature of the KDD process. This iterative aspect of the KDD process has been practically implemented through the use of a header bar, which appears on every page in the system. The main purpose of this header bar is to allow the user to easily assess the current state of the system and enable them to return to an earlier point in the process and modify the domain of their data mining model. It also helps to maintain a consistent look and feel throughout the user interface of the system, which in turn is important to lessen the chance that a user could become lost within the iterative nature of the KDD process. Examples of the header bar as a user progresses through the system are given in figures 4.2 through 4.4.

4.1.3 Volatile requirements

The following requirements are aspects of the system which cannot be relied upon to remain consistent. The implementation of the system aims to present this functionality in a flexible manner so that future updates to the underlying system do not render the extension inoperable. The encapsulation of these requirements also means that when updates to Snob do occur they can be easily incorporated into Snob-Online.

Commands

Snob is an interactive command-line driven program. The user inputs commands which are automatically processed by the system and results are written to standard output. The set of commands that are supported by Snob varies between versions. The command input interface for Snob-Online is divided into several sections. Screenshots of this section of the Snob-Online interface can be viewed in the figure 4.5. The two top sections of this interface provide a convenient interface for input of select range of commonly used processing and reporting commands. The bottom section of this interface is for general command inputs. This section allows users to execute the more obscure Snob commands by typing them into the text field. No restrictions are placed upon what commands a user can enter into the text field hence this section is without form validation.

Output

The results of each command is also another volatile requirement of the system as the format or content of this output could change in subsequent releases of the system. The generalised structure for the output of snob-vanilla is characterised below:

The first line of every command output text is preceded by a `#` character. The end of a command output is followed by 3 lines, each of which begin with the `>>` characters. Some commands may also have specific header and footer output. The most important information contained within the snob output can be generally classified as a series of

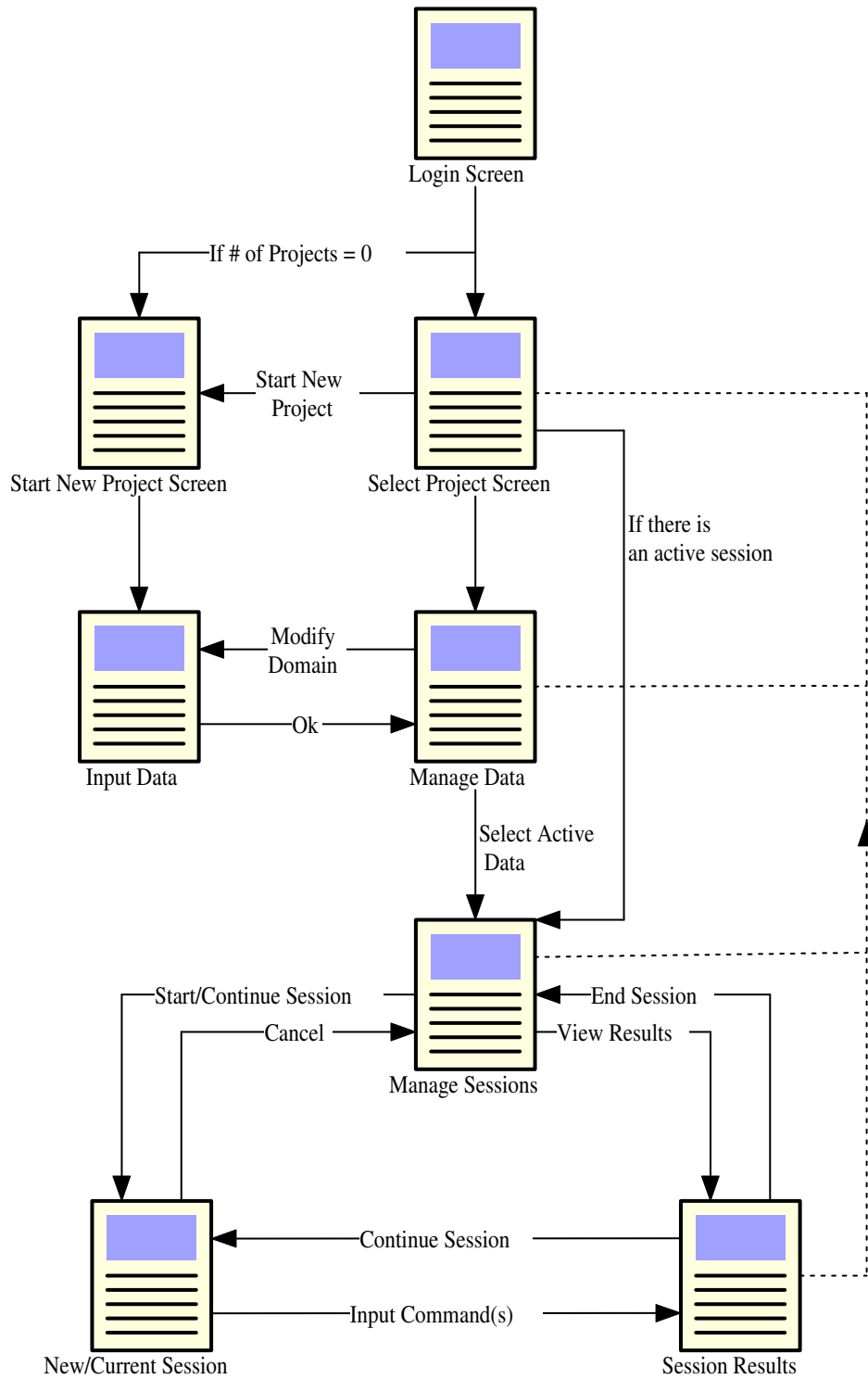


Figure 4.1: Sitemap for Snob-Online

Welcome to SNOB Online Mark Hollands
Manage Projects | Manage Data | Manage Sessions | Logout |

Figure 4.2: Header bar in project state

Welcome to SNOB Online Mark Hollands
Manage Projects | Manage Data | Manage Sessions | Logout |

Figure 4.3: Header bar in data state

Welcome to SNOB Online Mark Hollands
Manage Projects | Manage Data | **Manage Sessions** | Logout |

Figure 4.4: Header bar in session state

data records, which relate to the command that was issued. The structure of these data records varies widely between the commands they relate to. These data records are output in 1 or 2 lines each depending on the command. Each record generally contains several data items, each of which is usually space delimited and sometimes prefixed with a text identifier.

Results

Given that Snob commands and their subsequent output form the most important information that leads to knowledge discovery in the system, access to this information must still be provided regardless of its potential volatility.

Processing Commands

Process data for cycles

Reporting Commands

Class Overview (sum)

Class Summary (prclass -1 1)

Class Heirarchy (tree)

Class Detail (trep)

Item Detail (mrep)

General Command Interface

Command

Figure 4.5: Command input interface

4.2 Module Implementation

The four modules that make up the Snob-Online architecture discussed at the end of the previous chapter, are primarily divided based upon their relation to the client and server side design. The cgi scripts module is implemented using the PHP scripting language, as it has excellent facilities for session handling. To communicate with the next level of the interface heirarchy, the cgi script module makes a socket connection to the daemon process, resident upon the web server. The daemon process, labelled the snob server, is written using the Perl programming language. This socket interface between the two modules is used to pass data for tasks such as creating a new session, or command processing. The snob server is responsible for creating processes of snob-vanilla to which it sends commands. Each individual process of snob-vanilla created by the snob server, has its output re-routed to a file system location which is based upon the current user and project. Apart from commands which are passed through the interface heirarchy, Snob recieves its data from the file system. This data is placed on the file system by the cgi scripts that are responsible for data handling.

4.3 Interpretation

Based upon the volatile requirements of the system, an extensible subsystem for output interpretation was devised. The interpretation subsystem takes the output from Snob and parses this plain text, based upon the command for which it relates to. Once parsed, the system then outputs the required information from the results in a portable, consistent, XML format.

Defining a consistent interface for storage of the interpreted results of Snob, minimizes the amount of code that is required to be updated when the underlying functionality of Snob is changed. When new functionality is added to Snob, Snob-Online will still continue functioning without any update. To include interpretation of this new functionality, a new module can be added into the extensible interpretation subsystem.

4.3.1 Interpretation Functionality

Each module in this subsystem implements the following functionality:

XML input

Read in an XML results file and create a data structure to represent the results, so that it can be modified.

XML output

Writing the data structure of results out to an XML file.

Input from Snob

Takes the output from Snob as input and parses it to create the results data structure.

Interpreted output

Reading the results data structure and outputting the information stored within together with english text to create readable text detailing each record.

4.3.2 Interpretation Implementation

The implementation of this interpretation subsystem is created using the object orientated techniques available in the PHP scripting language. Each of the interpretation modules extend the Results class, as described in figure 4.6. Every class that extends the Results class forms an interface through which its own unique data types are accessed by generic method calls. The functionality that was described in the above section is implemented in the following function calls in each module.

1. parseFile()
2. outputXml()
3. readOutput(id, text)
4. writeOutput()

4.4 Visualisation

By combining the interpretation results regarding class allocation with raw snob input files, the system can output a data file in XML format so that it is compatible with 3rd party visualisation systems. The visualisation system that was chosen for testing of this functionality is entitled Ggobi(Swayne, Cook, Buja and Lang, 2002).

GGobi

GGobi is an open source, data visualisation system for viewing high-dimensional data. Extending it's predecessor Xgobi(Swayne, Cook and Buja, 1992), it provides a portable visualisation solution for both Windows and Linux systems. Ggobi supports a range of input formats, from Microsoft Excel output to plain text. Its recently added support for XML inputs was an important factor in its use with this project.

Colour Shading

The class distinctions that Snob outputs are based upon a partial mapping system, where it is possible for a data record to have several partial class allocations. Colour shading is used to display this partial class allocation. Each Snob class, has its items

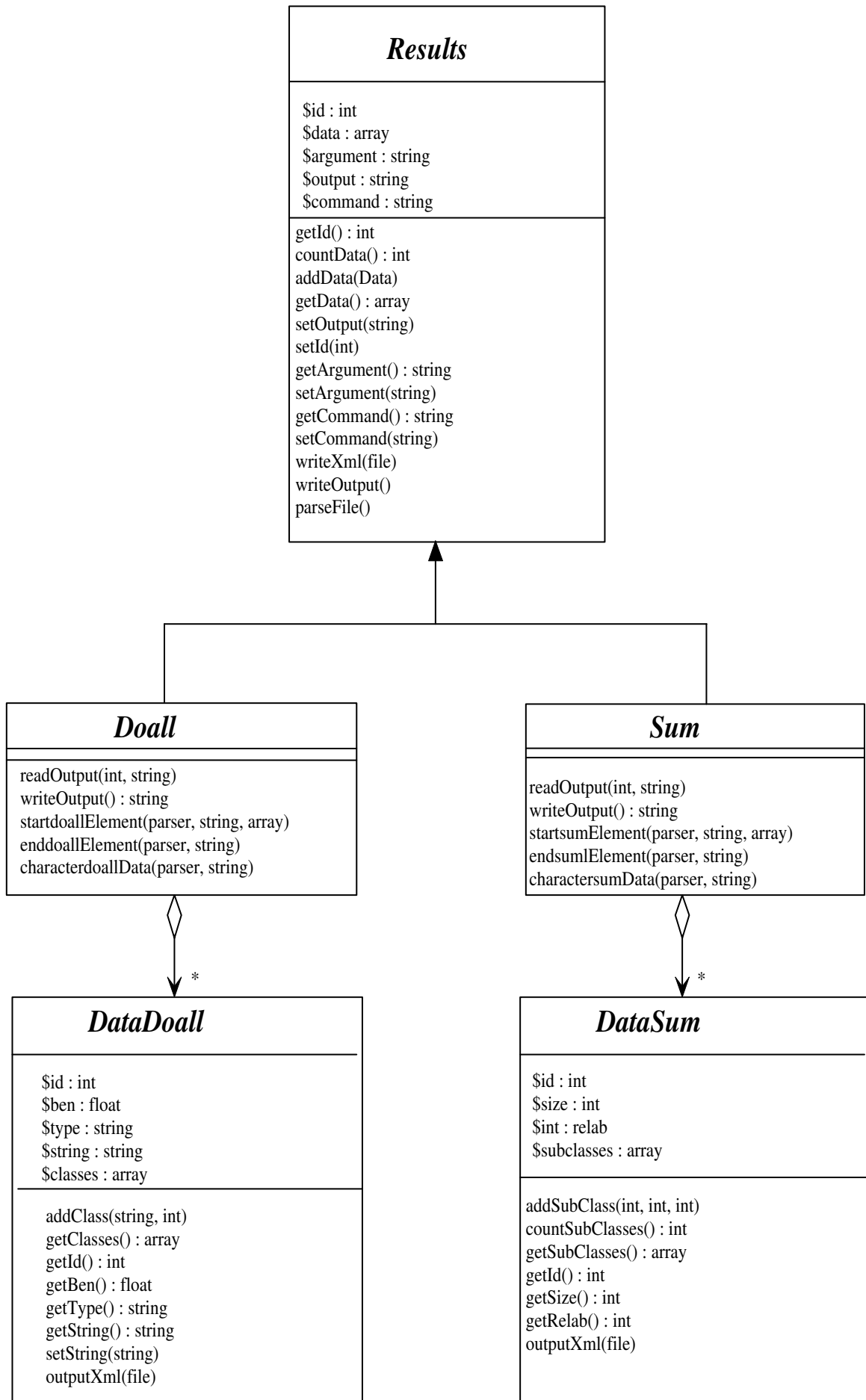


Figure 4.6: Interpretation class heirarchy

partitioned based upon their likelihood allocation percentage, with a 10% granularity. In the Ggobi colour scheme, each item maintains a label attribute which is mapped to a colour. Each Snob class is initially assigned a base colour. Each partition of the Snob class, then is mapped to a shade of that base colour depending on the allocation percentage range it encompasses, with decreasing percentage mapping to a lower colour intensity. This creates a display with areas of intense colour, where the allocation to a particular Snob class is the strongest. The subclasses form areas of diminishing colour intensity around the middle of the class. The formula for establishing ggobi class allocation is displayed below:

$$\text{ggobi label} = \text{class allocation} * 10 - \text{integer}[\max(\text{percent allocation})/10]$$

For example if an item has greater than 90% allocation to Snob class #5 then it is mapped to Ggobi label #50. If a record had between 70% and 80% allocation to class #5 then it would be mapped to label #48. Label #50 might be represented in the visualisation by a pure red colour. Label #48 would then be represented in the visualisation by a red colour with only 80% intensity. An example of this colour shading technique is evident in figure 4.7.

4.5 Testing Procedures

The design of this system has endeavoured to apply the techniques of user-centered design throughout this project's development.

Initial interviews with users of the previous system were carried out to:

- Determine how they used the system.
- Which functionality of the system they commonly used.
- What aspects of the system they thought could be improved.

This information formed much of the motivation for functionality that was developed in Snob-Online. The goal of this project was to improve the usability and knowledge discovery of Snob by implementating a web interface. The effectiveness of the implemented functionality to achieve this goal has been tested through the use of usability testing.

4.5.1 Usability Testing

Usability testing is a process which can be utilized to gain an understanding of how a computer system is used. It involves end-users, typically one or two working together and interacting with the system, while one or more people watch, listen, and take notes. The primary focus of the usability testing carried out in this project, was to compare snob-vanilla and Snob-Online and analyse the changes in usability and knowledge discovery that the interface changes effected. The test subjects were selected to allow for as wide a range of past experience with Snob as possible, encompassing both experts and novices. Each test subject was initially presented with the two programs and two

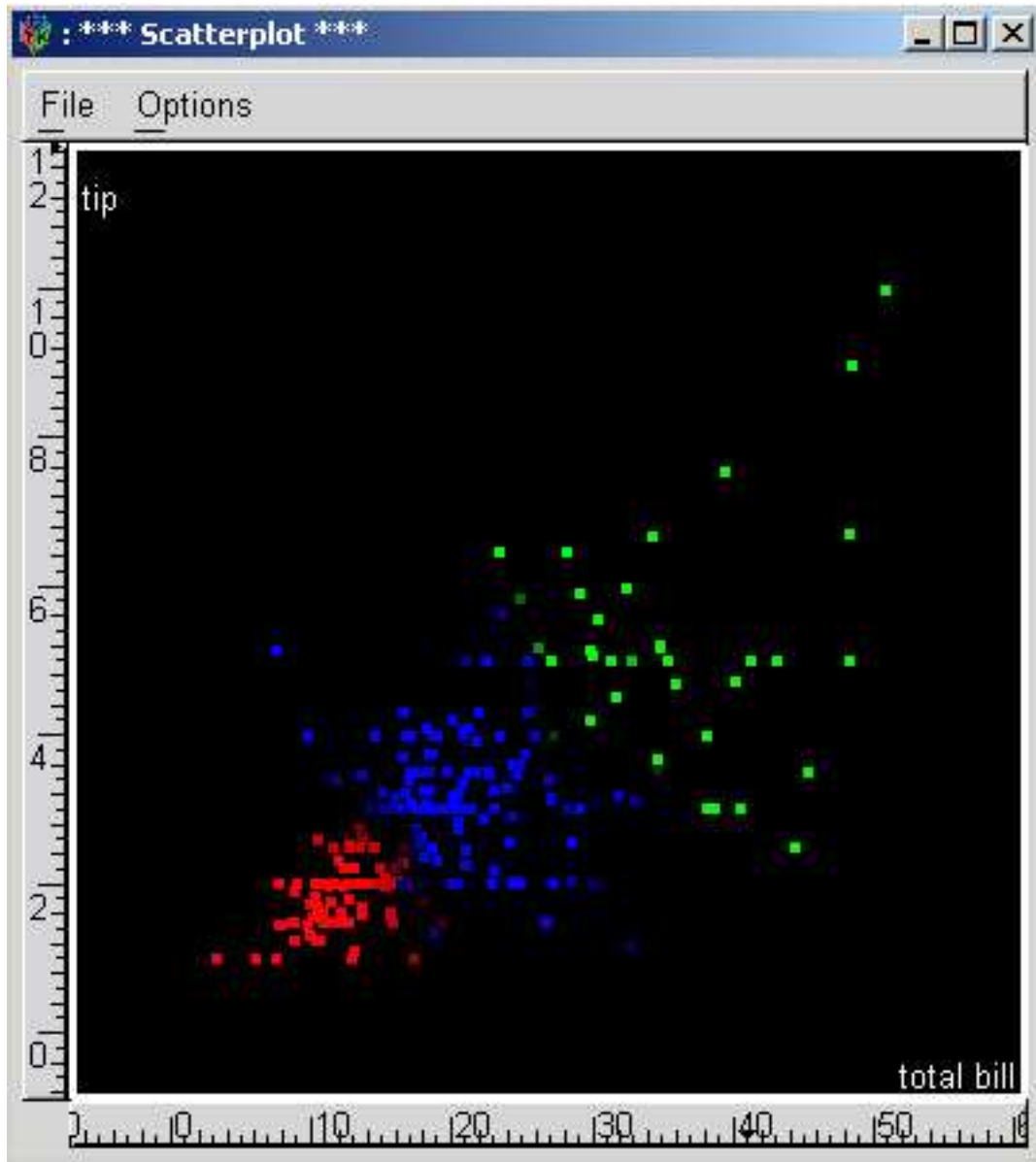


Figure 4.7: Color Shading Visualisation

datasets, of simple and intermediate complexity respectively. The testing procedure was divided into three separate stages, each attempting to compare different aspects of the two systems.

4.5.2 Stage #1

The first stage was focused upon comparing the perceived usability gains in the change of interface from command-line to form based web-interface. No interpretation or visualisation was carried out in Snob-Online during this stage and the text results output by both systems was identical. Each user was initially given either Snob-vanilla or Snob-Online, randomly selected, and the simple dataset, and asked to use the system to try and gain understanding about the dataset. Once finished, they were given the other program and the intermediate dataset, and tested this system. After they had completed the use of both systems, each user filled out a survey, detailing their experiences of both systems.

4.5.3 Stage #2

Stage two of the usability testing added interpretation functionality to the Snob-Online system. The purpose of this stage of the testing was to ascertain the change in knowledge discovery that was caused by presentation of the output from Snob in a more readable fashion when compared to their experience of stage #1.

4.5.4 Stage #3

Stage three of the usability testing added visualisation functionality to the Snob-Online system. The purpose of this stage of the testing was to ascertain the change in knowledge discovery that was caused by allowing the user to view the class distinctions using a graphical interface. After completing stage #2 and #3 of the testing, each user filled out another survey, comparing which aspects of the system were helpful in their understanding of the datasets and underlying patterns inherent within.

Chapter 5

Results and Discussion

5.1 Stage 1 Results

The questions posed in the feedback form for this section of the usability testing were chosen to emphasize comparison between the two different forms of user interface from a usability perspective. The two datasets that were used for testing in this stage were ordered based upon their complexity. The first, simple dataset details a population of Murray Cod fish and their length and age. Based upon the results gained from Snob users can make inferences about the fish population, such as the population of fish after a certain age significantly drops off once they reach the size for legal fishing. The second dataset describes information gathered from a restaurant regarding the size of tips that are received. This dataset includes seven attributes, detailing total bill, tip size, day of the week, lunch or dinner, smoking or non-smoking section, size of the group, male or female predominance. Based upon this information users can ascertain which groups of people give the most lucrative tips. Based upon this dataset, males seem to give bigger tips than females and meals eaten at dinner are usually accompanied by bigger tips. The results sets were presented to the user in order of increasing complexity. Users were randomly allocated which system to test first with the Murray Cod dataset. Once completed they tested the other system using the tips dataset.

5.1.1 Introductory Questions

The introductory questions posed at the start of this form were to provide background understanding about the users performing the testing. Using a multiple choice system, each user assessed their own experience or knowledge in the below sections:

1. Computer Literacy
2. Experience with Data Classification
3. Past Experience with Snob

The selection of users that participated in the usability testing were volunteers which who had a range of experience across the three variables listed above. These variables

are used to refer to specific subsets of users, throughout the following discussion of the results. Generally users of all experience levels were capable of making some inferences about the datasets during the testing. Some users with a low computer literacy, struggled with the concept of data mining and didn't provide detailed results for the questionnaire.

5.1.2 Interface Questions

The main section of the first feedback form aims to assess the differences in usability between two different interfaces effectively running the same program. Users were first asked in general which interface they preferred. The emphasis here was placed upon the interface, in order to get the users to think about the interfaces of both programs and the effectiveness with which both presented identical data.

Using the html form interface, Snob-Online endeavours to display additional state information about the session and commands currently being processed. The effectiveness of these techniques was qualitatively assessed in the next series of questions, which rated both interfaces in terms of usability. Through feedback during the initial testing, it was shown that some users were not clear about what usability was, and a better definition was developed for the instructions used in later testing.

The final set of questions in this section looked at which aspects of both interfaces users found to be a help or hinderance in accomplishing their task. The design of the user interface of Snob-Online was intended to provide as much control as possible, while still making common tasks easier to accomplish. This balance between control and usability, was to ensure that experienced users of Snob would not find Snob-Online to be a hinderance to their interactions with the system. Generally, experienced users were quick to understand the flow of the system and appreciated the flexible nature of persistent sessions. Some experienced users voiced concerns regarding session checkpoint files being difficult to use due to not being able to see the filesystem. This concern has been addressed by displaying the status of the last checkpoint file in the interface. The majority of novice users understood the structure of Snob-Online more quickly than Snob-vanilla. Helpful interface feedback from this section of the questionnaire highlighted aspects such as, better control for data inputs and defaulting to the last command processed for display in the results page.

Qualitative Results

1. Preferential comparison between interfaces
 - Snob-Vanilla 16%
 - Snob-Online 84%
2. Usability Ratings (Average rating between 1 - 5)
 - Snob-Vanilla 2.4
 - Snob-Online 4.1

5.1.3 Header Bar Questions

The section of the questionnaire relating to the header bar, aims to measure the effectiveness of this interface tool. The purpose of the header bar is to display the current state of the system and promote the iterative nature of the KDD process, by easily allowing users to navigate through the consistent interface to modify their previous choices. Users are asked to describe how effective they thought the header bar was at relating the state of the system. They were then asked to compare this with their testing of Snob-vanilla and operating without this state information directly visible.

Experienced users were quick to understand the structure of Snob-Online and were equally capable at using either system, with or without the header bar. Novice users sometimes struggled to recognize the iterative nature of the system, but generally found the display of state information to be helpful.

5.2 Stage 2 and 3 Results

Stages 2 and 3 of the usability testing examined the analysis techniques or result interpretation and visualisation. This section discusses the knowledge discovery potential using both techniques. Users were first asked to describe their understanding of the dataset and then assess which technique they attributed the knowledge gain to. In these sections, users were processing the restaurant tip dataset again, in order to gain a better understanding than was possible in stage #1 without the benefit of analysis tools.

Qualitative Results

Analysis technique effectiveness for improving knowledge discovery (Average rating between 1 - 5)

- Interpretation 2.1
- Visualisation 4.6

The effectiveness rankings show that users found visualisation considerably more effective at enhancing their understanding of the dataset, than interpretation. Because these two techniques show a vastly different scale, there is still potential for interpretation in data mining, but it should be used together with visualisation when users need to examine the specific details of the results.

5.2.1 Interpretation Discussion

Interpretation of Snob results showed some signs of providing enhanced knowledge discovery for novice users. Novice users commonly commented about the confusing nomenclature present in the output of Snob results. At a very minimum, the interpretation interface provided novice users with a better understanding of the commands they were executing. Novice users sometimes understood the underlying concepts of data mining better once the confusing aspects of the output had been removed. Many users

expressed desire to have a wider selection of commands supported within the interpretation framework.

5.2.2 Visualisation Discussion

Users of all experience levels reacted very well to visualisation analysis. The ability to view the entire dataset and class distinctions made by Snob in a graphical manner had a significant effect upon the knowledge discovery of the system. Almost all users were capable of making some inferences about the dataset once they had used the visualisation functionality. Feedback showed that clarity of attribute labelling within Ggobi is one aspect which could be improved upon. Every user in this usability test attributed a larger portion of their understanding to the visualisation analysis than the interpretation.

Chapter 6

Conclusion

The primary goal of the project, broadening the active user base of Snob by interfacing it with the internet, has been fulfilled in the creation of Snob-Online. The amalgamation of web technologies used to create Snob-Online, has formed a flexible and extensible system, capable of maturing with the future development of Snob.

Based upon the results discussed in the previous chapter, the secondary goal of improving the user interface of the system has also been accomplished, providing significant gains in the usability and knowledge discovery of the system. Experience here suggests that future data mining development should shift its focus towards ensuring that knowledge discovered by the mathematical processes is imparted to the user. Utilizing user-centered design techniques, combined with a focus upon the KDD process, can be used to model the user interactions that take place within a data mining system. Designing the user interface based upon user interactions leads to an extensible data mining environment and also has positive effects upon the usability and knowledge discovery of the system.

Visualisation is a highly effective manner for data mining systems to impart knowledge to their users. The integration of data mining to a wide range of visualisation applications can be achieved through the use of an XML interface. The portable nature of this data format proved highly effective in interfacing Snob-Online with the visualisation package Ggobi.

6.1 Recommendations for Future Work

The maintenance of Snob-Online will have to take into account the future development of Snob. Implementation of a wider set of commands into the interpretation framework is one immediate task for this maintenance. Creating interfaces with more visualisation systems is another task, which if well integrated into the web interface could have significant usability gains for the system.

The analysis stage of the KDD process recieved the most attention in the implementation of Snob-Online, as it has the potential to provide enhanced knowledge discovery. Significant work can still be undertaken in extending this system to enhance the usability of the system in handling varied data inputs. Storing Snob input data in XML

format, allows for ease of data modification and custom Snob input files can be generated on the fly. Beyond the scope of this system, research towards a standard data mining XML schema would significantly improve interoperability between data mining applications. Systems generating data to be used for data mining applications could directly output within this schema and data mining systems could interpret the schema as input, removing the need to time consuming data conversion between systems.

While the design of Snob-Online has inherently been structured around the constraints of a data mining system, there is desire to be able to interact with generic text-based Unix applications via the internet. Snob-Online could be used as a basis for further work in this area. Interface techniques such as the result interface could be extended to operate with any interactive Unix application. The amalgamation of technologies used to create Snob-Online would also serve as an excellent case study towards what can be accomplished with a purely html based user interface.

References

- Alexander, D. (2003). Redesign of the monash university web site: A case study in user-centred design methods, *AusWeb'03*. (last accessed July 28th, 2003).
URL: <http://ausweb.scu.edu.au/aw03/papers/alexander/paper.html>
- Basden, A. (1996). User interface for knowledge discovery, *Digest of Colloquium of Knowledge Discovery and Data Mining*.
- Berry, M. and LinHoff, G. (1997). *Data Mining Techniques for Marketing, Sales and Customer Support*, Wiley Computer Publishing, New York.
- Chattratchat, J., Guo, Y. and Syed, J. (1999). A visual language for internet-based data mining and data visualization, *IEEE Symposium on Visual Languages*, IEEE, Tokyo, Japan, pp. 64–71.
- de Figueiredo, R. J. P. and Lai, G. C. (2000). A perspective of human-centered systems, *IEEE - Circuits and Systems* **11**(2): 3.
- Fayyad, U., Piatetsky-Shapiro, G. and Smyth, P. (1996). From data mining to knowledge discovery: an overview, *Advances in knowledge discovery in databases* **17**: 37–54.
- Gould, J. and Lewis, C. (1985). Designing for usability: Key principals and what designers think, **28**(3): 300–311.
- Gulliksen, J., Lantz, A. and Boivie, I. (1999). User centered design in practice - problems and possibilities, *User Centered Design in Practice - Problems and Possibilities*, Centre for User Orientated IT Design, pp. 7–27.
- Lassing, N., Rijsenbrij, D. and van Vliet, H. (2002). How well can we predict changes. (last accessed November 1st, 2003).
URL: <http://citeseer.nj.nec.com/493104.html>
- Lewis, C., Brand, C., Cherry, G. and Rader, C. (1998). Adapting user interface design methods to the design of educational activities, *CHI'98 Proceedings*, pp. 619–626.
- Norman, D. A. (1988). *Design of Everyday Things*, Bantam Doubleday Dell Publishing Group, New York.
- Robinson, N. and Shapcott, M. (2002). Data mining information visualisation - beyond charts and graphs, *Proceedings of the Sixth International Conference on Information Visualisation (IV'02)*, IEEE, pp. 577–.

- Swayne, D. F., Cook, D. and Buja, A. (1992). *User's Manual for XGobi: A Dynamic Graphics Program for Data Analysis*, Bellcore, Morristown, N.J.
- Swayne, D. F., Cook, D., Buja, A. and Lang, D. T. (2002). Ggobi manual. (last accessed November 6th, 2003).
URL: <http://www.ggobi.org/manual.pdf>
- Witt, B., Baker, T. and Merritt, E. (1994). *Software Architecture and Design: Principles, Models, Methods*, Van Nostrand Reinhold, New York, NY.

Cgi Script Module

```

#!/usr/local/bin/php
<?

/**
 * Form for selecting the active dataset for a project
 * @package snob
 * @category data
 */

require_once('header.cgi');

if (isset($_POST['project']))
{ // if post data exists
$project = $_POST['project'];
session_register('project');
$_SESSION['project'] = $project;

// ensure that the user hasn't gone back through the process invalidating any of their pre
$_SESSION['data'] = FALSE;
$_SESSION['session'] = FALSE;
}
else if (session_is_registered('project'))
{ // if no post data exists but session data exists
$project = $_SESSION['project'];
}
else
{ // if no project data exists
redirect('project-manage.cgi');
}

$username = $_SESSION['username'];
// Form output
if (chdir('users/' . $username . '/' . $project . '/data/'))
{
print "<table>
<tr><td><h2>Project: $project</h2></td></tr>
<tr><td><form method='POST' action='session-manage.cgi'></td></tr>
<tr><td><select name='data' multiple size='5'>";

$dir = opendir('.');
// Read each file in the data directory
while ($file = readdir($dir))
{
if (is_file($file))
{
// Display only the ones with .vset suffix
if (substr_count($file, '.vset') > 0)
{

```

```

$file = str_replace('.vset', '', $file);
print "<option value='$file'>$file</option>\n";
}
}
}
print "</select></td></tr>";
print "<tr><td><input type='submit' value='Submit'></form></td></tr>";
closedir($dir);

print "<tr><td><a href='data-new.cgi'>New Data</a></td></tr></table>";
}

#!/usr/local/bin/php
<?

/**
 * Form for uploading a new dataset to a project
 * @package snob
 * @category data
 */

require_once('header.cgi');

$submit = $_POST['submit'];

$username = $_SESSION['username'];
$project = $_SESSION['project'];

// Form processing
if ($submit)
{
// If the uploaded vset file doesn't exist
if (!is_uploaded_file($_HTTP_POST_FILES['vset']['tmp_name']))
{
$error = "You didn't select a vset file to upload";
unlink($_HTTP_POST_FILES['vset']['tmp_name']);
}
else
{
// If the uploaded vset file is too large
$maxfilesize=1024000;
if ($_HTTP_POST_FILES['vset']['size'] > $maxfilesize)
{
$error = "The selected vset file is too large, it must be less than $maxfilesize bytes.";
unlink($_HTTP_POST_FILES['vset']['tmp_name']);
}
else
{

```

```

$filepath = 'users/' . $username . '/' . $project . '/data/' . $HTTP_POST_FILES['vset']['name'];
$vsetpath = ereg_replace(".txt$", ".vset", $filepath);

// Copy the vset file to the webserver
if (copy($HTTP_POST_FILES['vset']['tmp_name'], $vsetpath))
{
print "file copied";
}
else
{
$error = "The vset file copy failed";
}

}

}

// If the samp file is a valid file
if (!is_uploaded_file($HTTP_POST_FILES['samp']['tmp_name']))
{
$error = "You didn't select a sample file to upload";
unlink($HTTP_POST_FILES['samp']['tmp_name']);
}
else
{
$maxfilesize=1024000;
// If the samp file is too large
if ($HTTP_POST_FILES['samp']['size'] > $maxfilesize)
{
$error = "The selected samp file is too large, it must be less than $maxfilesize bytes.";
unlink($HTTP_POST_FILES['samp']['tmp_name']);
}
else
{
$filepath = 'users/' . $username . '/' . $project . '/data/' . $HTTP_POST_FILES['samp']['name'];
$samppath = ereg_replace(".txt$", ".samp", $filepath);

// Copy the file to the webserver
if (copy($HTTP_POST_FILES['samp']['tmp_name'], $samppath))
{
print "samp copied";
}
else
{
$error = "The sample file copy failed";
}
}
}

```

```

}
}

// If there were no errors during uploading
if (!isset($error))
{

    $vset = $HTTP_POST_FILES['vset']['name'];
    $vseta = split(".", $vset);

    $datastr = substr($vset, 0, strpos($vset, ".txt"));

    $_SESSION['data'] = $datastr;
    require_once('dataParser.cgi');
    // Parse the data file and create and XML representation
    $dataparse = new DataParser();
    $dataparse->parseDataFiles();

    redirect('session-manage.cgi');

}
else
{
    print "$error Please try again";
}

}
// Form output
$self = $_SERVER['SELF'];
print "<table><form name='submitform' action='$self' method='post' enctype='multipart/form
<tr><td>Vset File Location:</td>";

print "<td><input type='file' name='vset'></td></tr>";
?>
<tr>
<td>Sample file Location:</td>
<td><input type='file' name='samp'></td>
</tr>

<tr>
<td>Description: </td>
<td><textarea name='desc'></textarea></td>
</tr>

<tr>
<td><input type='submit' name='submit' value='Submit'> </td>
<td></td>
</tr>

```

```

</table>

</form>

<?

/**
 * XML handling for Snob input files
 * @package snob
 * @category data
 *
 */

/**
 * Data Record object
 */
class Record
{
var $id;
var $data;
var $class;
var $percent;

/**
 * Record constructor
 * @param int $id Record id
 * @param string $data String of all the data objects space delimited
 */
function Record($id, $data)
{
$this->id = $id;
$this->data = $data;
}

/**
 * Return the id of this Record
 * @return int Record id
 */
function getId()
{
return $this->id;
}

/**
 * Return the data string of this Record
 * @return string Record data
 */

```

```

function getData()
{
return $this->data;
}

/**
 * Return the class of this Record
 * @return int Record class value
 */
function getClass()
{
if ($this->class == 5)
{
$class = 4 - (10 - ceil($this->percent/10));
}
else if ($this->class == 6)
{
$class = 9 - (10 - ceil($this->percent/10));
}
else if ($this->class == 9)
{
$class = 14 - (10 - ceil($this->percent/10));
}
print "cl : $this->class pc : $this->percent co : $class<br>";
return $class;
}

/**
 * Set the class value of this record
 * @param int $class Updated class value
 */
function setClass($class)
{
$this->class = $class;
}

/**
 * Set the percentage allocation value of this record
 * @param int $percent Updated percent value
 */
function setPercent($percent)
{
$this->percent = $percent;
}

/**
 * Return a boolean whether this record has a class value set
 * @return boolean Class is set

```

```

    */
function hasClass()
{
if (isset($this->class))
{
return true;
}
return false;
}

/**
 * Output the XML version of this record to file handle
 * @param file Target file for output
 */
function outputXml($file)
{
if ($this->hasClass())
{
fputs($file, "<record label='" . $this->getId() . "' color='" . $this->getClass() . "'>\n"
}
else
{
fputs($file, "<record label='" . $this->getId() . "'>\n");
}

fputs($file, trim($this->getData()) . "\n</record>\n");
}
}

/**
 * Object for storing information classifying each variable in a dataset
 *
 */
class Variable
{
var $name;
var $type;
var $levels;

/**
 * Constructor for the Variable dataset
 * @param string $name Name of the variable
 * @param int $type 1 = Real variable, 2 = Categorical Variable
 */
function Variable($name, $type)
{
$this->name = $name;
$this->type = $type;
}
}

```

```

}

/**
 * Set the number of categories for a categorical variable
 * @param int $level Category number
 */
function setLevels($level)
{
$this->levels = $level;
}

/**
 * Return the name of this Variable
 * @return string variable name
 */
function getName()
{
return $this->name;
}

/**
 * Return the Variable type
 * @return int variable type
 */
function getType()
{
return $this->type;
}

/**
 * Return the number of levels in this categorical variable
 * @return int number of variables
 */
function getLevels()
{
return $this->levels;
}

/**
 * Output the Variable in XML format to file
 * @param file $file Target file
 */
function outputXml($file)
{
if ($this->type == "1")
{
fputs($file, "<realvariable name='" . $this->name . "' />\n");
}
}

```

```

else if ($this->type == "2")
{
fputs($file, "<categoricalvariable name='" . $this->name . "'>\n");
fputs($file, "<levels count='" . $this->levels . "'>\n");

// put handling in for variable names

fputs($file, "</levels>\n");
fputs($file, "</categoricalvariable>\n");
}
}

}

/**
 * Entry point for the data XML handler
 *
 */

class DataParser
{

var $file;
var $vset;
var $samp;
var $trep;

var $filename;

var $records = array();
var $variables = array();

var $name;

/**
 * Constructor for the DataParser object
 *
 */
function DataParser()
{
$username = $_SESSION['username'];
$project = $_SESSION['project'];
$dataset = $_SESSION['data'];

$path = "users/$username/$project/data/";
$this->filename = $path . $dataset . ".xml";
$vsetfile = $dataset . ".vset";
$sampfile = $dataset . ".samp";

```

```

$this->vset = fopen($path . $vsetfile, "r");
$this->samp = fopen($path . $sampfile, "r");
}

/**
 * Set the name of this dataset
 * @param string dataset name
 */
function setName($name)
{
$this->name = $name;
}

/**
 * Output the dataset XML header
 */
function printHeader()
{
$str = "<?xml version='1.0'?>\n<!DOCTYPE ggobidata SYSTEM 'ggobi.dtd'>\n<ggobidata>\n";
fwrite($this->file, $str);
}

/**
 * Output the data header in XML
 * @param string $name dataset title
 */
function printDataHeader($name)
{
$str = "<data name='$name'>\n<description>\n</description>\n<activeColorScheme file='color

fputs($this->file,$str);
}

/**
 * Output the variable stored in this dataset
 */
function printVariables()
{
fputs($this->file,"<variables count='" . sizeof($this->variables) . "'>\n");
foreach ($this->variables as $variable)
{
$variable->outputXml($this->file);
}
fputs($this->file, "</variables>\n");
}

```

```

/**
 * Parse Variables from array
 * @param array $lines Array of strings defining the variables taken from vset file
 */
function parseVariables($lines)
{
    foreach ($lines as $line)
    {
        $items = preg_split("/[\s]+/", trim($line));
        $name = $items[0];
        $type = $items[1];
        $variable = new Variable($name, $type);

        if ($type == "2")
        {
            $levels = $items[2];
            $variable->setLevels($levels);
        }
        array_push($this->variables, $variable);
    }
}

/**
 * Add a variable object to the array
 * @param Variable $variable New Variable object
 */
function addVariable($variable)
{
    array_push($this->variables, $variable);
}

/**
 * Add a Record object to the array
 * @param Record $record New Record object
 */
function addRecord($record)
{
    $id = $record->getId();
    $this->records["$id"] = $record;
}

/**
 * Get a Record object at index
 * @param int $id Target index
 */
function getRecord($id)
{
    $rec = $this->records["$id"];
}

```

```

return $rec;
}

/**
 * Print the header for the records section of the XML
 * @param int $records Number of records
 */
function printRecordHeader($records)
{
fputs($this->file, "<records count='$records'>\n");
}

/**
 * Print the footer for the records section of the XML
 */
function printRecordFooter()
{
fputs($this->file, "</records>");
}

/**
 * Print the footer for the XML file
 */
function printFooter()
{
fputs($this->file, "</data>\n</ggobidata>\n");
}

/**
 * Parse the trep file to gain class allocation details
 */
function parseTrepFile()
{
{
$username = $_SESSION['username'];
$project = $_SESSION['project'];
$session = $_SESSION['session'];
$data = $_SESSION['data'];
$fileSrc = "users/$username/$project/data/$data.trep";

$this->trep = fopen($fileSrc, "r");
$line = fgets($this->trep, 4096);
// Discard lines until the line starting with + is reached
while (!ereg("\+", $line))
{
$line = fgets($this->trep, 4096);
}

// Parse each class allocation ordered by Record id

```

```

while (!feof($this->trep))
{
$line = fgets($this->trep, 4096);
$item = preg_split("/[\s]+/", trim($line));
$id = $item[0];
$class = $item[1];
$percent = $item[2];
$percent = ereg_replace("\(", "", $percent);
$percent = ereg_replace("\)", "", $percent);

$record = $GLOBALS['dataparser']->getRecord($id);
// Add class data to a record
if (is_object($record))
{
$record->setClass($class);
$record->setPercent($percent);
$GLOBALS['dataparser']->addRecord($record);
}
else
{
}
}
}

/**
 * Parse the vset and samp files to create the data structure
 */
function parseDataFiles()
{

$name = trim(fgets($this->vset, 4096));
$vars = trim(fgets($this->vset, 4096));
$lines = array();

// Push lines into array
while (!feof($this->vset))
{
$var = trim(fgets($this->vset, 4096));
if (strlen($var) > 0)
{
array_push($lines, $var);
}
}

$this->name = $name;
$this->parseVariables($lines);

// Discard the first 3 lines of the .samp file

```

```

fgets($this->samp, 4096);fgets($this->samp, 4096);fgets($this->samp, 4096);

$records = trim(fgets($this->samp, 4096));
while (!feof($this->samp))
{
$line = fgets($this->samp, 4096);

$data = preg_split("[ ]+", trim($line));
$id = $data[0];
$str = "";
$i = 1;
while ($i < sizeof($data))
{
$item = $data[$i];
if ($i != 1)
$str .= " ";
$str .= trim($item);
$i++;
}

if ($id > 0)
{
$record = new Record($id, $str);

$this->addRecord($record);
}
}
$this->writeXml();
}

/**
 * Output the data structure to XML format
 */
function writeXml()
{
$this->file = fopen($this->filename, "w");
print "Writing to " . $this->filename . "<br>";
$this->printHeader();
$this->printDataHeader($name);
$this->printVariables();
$this->printRecordHeader(sizeof($this->records));

foreach ($this->records as $record)
{
$record->outputXml($this->file);
}

$this->printRecordFooter();

```

```

$this->printFooter();
}

}

/**
 * XML input handler entry point
 */
function parseDataFile()
{
// Get session data
$username = $_SESSION['username'];
$project = $_SESSION['project'];
$session = $_SESSION['session'];
$data = $_SESSION['data'];
$fileSrc = "users/$username/$project/data/$data.xml";

// Setup the XML parser
$xml_parser = xml_parser_create();
xml_set_element_handler($xml_parser, "startDataElement", "endDataElement");
xml_set_character_data_handler($xml_parser, "characterDataData");

xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, false);
if (file_exists($fileSrc))
{
if (!($fp=fopen($fileSrc, "r")))
{
die("Cannot open $fileSrc");
}
}
else
{
print "\nCannot find " . $fileSrc . "\n";
}

while (($data=fread($fp, 4096))
{
if (!xml_parse($xml_parser, $data, feof($fp)))
{
die(sprintf("Xml error at line %d column %d ",
xml_get_current_line_number($xml_parser),
xml_get_current_column_number($xml_parser)));
}
}
xml_parser_free($xml_parser);
fclose($fp);

```

```

}

/**
 * Handler for the start of an XML tag and attribute values stored within
 * @param parser $parser XML parser
 * @param string $name Name of tag
 * @param array $attr Attributes for this tag
 */
function startDataElement($parser, $name, $attr)
{
$GLOBALS['currentElement'] = $name;

if ($GLOBALS['currentElement'] == "data")
{
$GLOBALS['dataparser']->setName($attr["name"]);
}
else if ($GLOBALS['currentElement'] == "realvariable")
{
$GLOBALS['variablename'] = $attr["name"];
// name
}
else if ($GLOBALS['currentElement'] == "categoricalvariable")
{
$GLOBALS['variablename'] = $attr["name"];
// name
}
else if ($GLOBALS['currentElement'] == "levels")
{
$GLOBALS['templevels'] = $attr["count"];
// count
}
else if ($GLOBALS['currentElement'] == "record")
{
$GLOBALS['templabel'] = $attr["label"];
$GLOBALS['tempcolor'] = $attr["color"];
// process count and class
}
}

/**
 * Handler for end an XML tag, retrieve data from middle of tag assigned to $GLOBALS['tagname']
 * @param parser $parser XML parser
 * @param string $name XML tag name
 */
function endDataElement($parser, $name)
{
if (strcmp($name, "categoricalvariable") == 0)

```

```

{
$var = new Variable($GLOBALS['name'], "2");
$var->setLevels($GLOBALS['templevels']);
$GLOBALS['templevels'] = "";
$GLOBALS['dataparser']->addVariable($var);

}
else if (strcmp($name, "realvariable") == 0)
{
$var = new Variable($GLOBALS['name'], "1");
$GLOBALS['dataparser']->addVariable($var);
}
else if (strcmp($name, "record") == 0)
{
$id = $GLOBALS['templabel'];
$data = $GLOBALS['record'];
$GLOBALS['record'] = "";
$GLOBALS['templabel'] = "";

$rec = new Record($id, $data);
if (isset($GLOBALS['tempcolor']))
{
$rec->setClass($GLOBALS['tempcolor']);
}
$GLOBALS['dataparser']->addRecord($rec);
}
}

/**
 * Pushes xml tag data into $GLOBALS array.
 * Any tags that have data stored inside <tag>data</tag> need to be in the $elements array
 *
 */
function characterDataData($parser, $data)
{
$elements = array("record");

foreach ($elements as $element)
{
if ($GLOBALS['currentElement'] == $element)
{
$GLOBALS[$element] .= $data;
}
}
}

?>

```

```

<?
/**
 * Handler factory creates and returns a hanlder object derived from the Results class
 * @package snob
 * @category interp
 */
class handlerFactory
{
/**
 * Get a handler class
 * @param string $command Command being interpreted
 * @param int $id Command id
 * @param string $arg arguments of the command
 */
function & getHandler($command, $id, $arg)
{
$href = trim($command) . ".cgi";
$dir = ".";

$test2 = "handler/$href";

if (file_exists($test2))
{
require_once($test2);
$trimcom = trim($command);
if (strcmp($trimcom,"doall")==0)
{
return new DoAll($arg, $id);
}
else if (strcmp($trimcom, "sum")==0)
{
return new Sum($id);
}
}
else
{
require_once('results.cgi');
return new Results($id);
}

}
}

?>

<?

/**

```

```

* Print out the headers for the Snob-Online system
* @package snob
* @category header
*/

session_start();
require_once('miscfunc.cgi');
if (!session_is_registered('username'))
{
if ($_SERVER['PHP_SELF'] != '/ml_2003_g01/hons/login.cgi')

{
redirect('login.cgi');
}

}

print "<html><head></head><body>";
$fullnam = $_SESSION['fullname'];
print "Welcome to SNOB Online " . $fullnam . "<br>";
buttonOption('Manage Projects', 'project-manage.cgi');
buttonOptionCheck('Manage Data', 'data-manage.cgi', 'project');
buttonOptionCheck('Manage Sessions', 'session-manage.cgi', 'data');
buttonOptionCheck('Session Interp', 'session-interp.cgi', 'data');
buttonOption('Logout', 'logout.cgi');

?>

<?

/**
* Output the javascript used in the interpretation screen of the system
* @package snob
* @category jscrip
*/

?>
<SCRIPT LANGUAGE="JavaScript">

var available=new Array();
var results=new Array();

<?

$username = $_SESSION['username'];

```

```

$project = $_SESSION['project'];
$session = $_SESSION['session'];

require_once('sessionxml.cgi');

$i = 0;
$sessions = readXml();
$sessiona = $sessions[0];
$commands = $sessiona->getCommands();

foreach ($commands as $command)
{
    $timest = $command->getTimestamp();
    $comname = $command->getCommand();
    $comfile = "users/$username/$project/sessions/$session/" . $timest . ".xml";
    if (file_exists($comfile))
    {
        print "available[$i] = true;\n";
        require_once('handlerFactory.cgi');
        $hf = new handlerFactory();
        $items = split(" ", $comname);
        $handler = $hf->getHandler($items[0], $timest, $items[1]);
        if (is_object($handler))
        {
            $GLOBALS['parser'] = $handler;
            $GLOBALS['parser']->parseFile();
            $str = $GLOBALS['parser']->writeOutput();
            print "results[$i] = '$str';\n";
        }
    }
    else
    {
        print "available[$i] = false;\n";
        print "results[$i] = 'The Intepretation Data is not currently available for this command.'\n";
    }
    $i++;
}

?>

function interpUpdate()
{
    var index = document.results.comlist.selectedIndex;
    if (document.results.displaytype[0].checked)
    {
        document.results.result.value = array[index];
    }
}

```

```

}
else
{
document.results.result.value = results[index];
}

}

</SCRIPT>

<?

/**
 * Output the javascript used in normal results screen of the system
 * @package snob
 * @category jscript
 */

?>

<SCRIPT LANGUAGE="JavaScript">

var array=new Array();
<?php

$username = $_SESSION['username'];
$project = $_SESSION['project'];
$session = $_SESSION['session'];

$filename = "users/$username/$project/sessions/$session/output";
//if (file_exists($file))
//{
$file = fopen($filename, "r");
$row = 0;
$text = "";
while (!feof($file))
{
$line = fgets($file, 1000000);
$linetrim = trim($line);
if (ereg("^>> Adjust PT Assign P",$linetrim))
{

$text = $text . $linetrim . '\n';
?>
array[<? print $row; ?>] = "<? print $text; ?>";
<?
$text = "";

```

```

$row++;
}
else
{
$text = $text . $linetrim . '\n';
}
}
?>
array[<? print $row; ?>] = "<? print $text; ?>";
<?
//}
//else
//{{
//}
?>

function update()
{

var index = document.results.comlist.selectedIndex;
document.results.result.value = array[index];
}

</SCRIPT>

#!/usr/local/bin/php
<?

/**
 * Login screen for Snob-Online
 * @package snob
 * @category login
 */

session_start();
require_once('loginxml.cgi');
print "<html><head></head><body>";
$use = $_POST['username'];
$pas = $_POST['password'];
if (isset($username) && checkPassword($use, $pas))
{
echo "<script language='JavaScript'>
window.location = 'project-manage.cgi'</script>";
}
else
{

```

```

// output login form
?>

</td></tr></table>
<FORM ACTION="<?php echo($_SERVER[$_SERVER['PHP_SELF']]); ?>" METHOD=POST>
<table align="center">
<?php if (isset($user)) {
echo "<p align='center'>Invalid username password combination. Please try again</p>";} ?>

<tr>
<td>Username: </td>
<td><input type="text" name='username' size="15"></td>

</tr>
<tr>
<td>Password: </td>
<td><input type="password" name='password' size="15"></td>
</tr>

<tr>
<td><input type="submit" name='submit' value='Login'></td><td></td>
</tr>
</form>

</table>

<?php
}
print "</body></html>";

?>

<?

/**
 * Login handler for the authentication system
 * @package snob
 * @category login
 */

// the xml file we want to parse
$xmlSource="accdat.xml";

//First we define a number of variables to store the data from each element

$username="";

```

```

$password="";
$fullname="";
$email="";

$currentElement=""; //holds the name of the current element

$accounts=array(); //array to hold all the account data

/* The start Element Handler
* This is where we store the element name, currently being parsed, in $currentElement.
* the character data handler uses this to identify the element.
* This is also where we get the attribute, if any.
*/
function startElement($parser,$name,$attr){

$GLOBALS['currentElement']=$name;

} //end startElement()

/*
* The end Element Handler
*/

function endElement($parser,$name){
$elements=array('username','password','fullname','email');

if(strcmp($name,"user")==0)
{
foreach($elements as $element)
{
$temp[$element]=$GLOBALS[$element];
}
$GLOBALS['accounts'][]=$temp;
}

if(strcmp($name,"user")==0){

    $GLOBALS['username']="";

    $GLOBALS['password']="";

    $GLOBALS['fullname']="";

    $GLOBALS['email']="";

}

```

```

} //end endElement()

/* The character data Handler
* Depending on what the currentElement is,
* the handler assigns the value to the appropriate variable
*/

function characterData($parser, $data) {
    $elements = array (
        'username', 'password', 'fullname', 'email'
    );

    foreach ($elements as $element) {
        if ($GLOBALS["currentElement"] == $element) {
            $GLOBALS[$element] .= $data;
        }
    }
}

/* This is where the actual parsing is going on.
* parseFile() parses the xml document and return an array
* with the data we asked for.
*/

function parseFile(){
    global $xmlSource, $accounts;

    /*Creating the xml parser*/
    $xml_parser = xml_parser_create();

    /*Register the handlers*/
    xml_set_element_handler($xml_parser, "startElement", "endElement");
    xml_set_character_data_handler($xml_parser, "characterData");

    /*Disables case-folding. Needed for this example*/
    xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, false);

    /*Open the xml file and feed it to the parser in 4k blocks*/
    if (!$fp = fopen($xmlSource, "r")) {
        die("Cannot open $xmlSource ");
    }
    while (($data = fread($fp, 4096))){

        if (!xml_parse($xml_parser, $data, feof($fp))) {
            die(sprintf("XML error at line %d column %d ",
                xml_get_current_line_number($xml_parser),

```

```

        xml_get_current_column_number($xml_parser));
    }
}

/*Finish ! we free the parser and returns the array*/
xml_parser_free($xml_parser);

return $accounts;

} //end parseFile()

/**
 * Authenticate a username and password combination and set the session variable if success
 * @return boolean Authentication pass
 *
 */
function checkPassword($usern, $passw)
{
$result=parseFile();

foreach($result as $arr)
{
$username = $arr["username"];
$username = rtrim($username);
$password = $arr["password"];
$password = rtrim($password);

$u = strcmp($username,$usern);
$p = strcmp($password,$passw);

if (strcmp($username,$usern) == 0)
{
if (strcmp($password,$passw) == 0)
{
session_register('username');
$_SESSION['username'] = $usern;
session_register('fullname');
//print "gf: " . $arr["fullname"] . "<br>";
$_SESSION['fullname'] = trim($arr["fullname"]);
session_register('email');
$_SESSION['email'] = trim($arr['email']);
return true;
}
}
}

}
return false;

```

```

}

function writeXml($accounts)
{
global $xmlSource;
$file = fopen($xmlSource, "w");
if (!file)
{
echo "Failed to open file<br>";
}
else
{
$header = '<?xml version="1.0" encoding="ISO-8859-1"?>' . "\n";
fputs($file, $header);
fputs($file, "<users>\n");
foreach ($accounts as $account)
{
fputs($file, "<user>\n");

$user = trim($account['username']);
$pass = trim($account['password']);
$full = trim($account['fullname']);
$emai = trim($account['email']);

$usertext = "<username>" . $user . "</username>\n";
$passtext = "<password>" . $pass . "</password>\n";
$fulltext = "<fullname>" . $full . "</fullname>\n";
$emailtext = "<email>" . $emai . "</email>\n";

fputs($file, $usertext);
fputs($file, $passtext);
fputs($file, $fulltext);
fputs($file, $emailtext);

fputs($file, "</user>\n");
}
fputs($file, "</users>\n");
fclose($file);
}
}

function addAccount($u, $p, $f, $e)
{
$newacc = array("username" => $u, "password" => $p, "fullname" => $f, "email" => $e);
$accountsall = parseFile();
array_push($accountsall, $newacc);
writeXml($accountsall);
}

```

```

}

?>

#!/usr/local/bin/php
<?

/**
 * Logout from Snob-Online
 * @package snob
 * @category login
 */

session_start();

// Destroy session variable and redirect to login page
session_destroy();
echo "<script language='JavaScript'>
window.location = 'login.cgi'</script>";

?>

<?php

/**
 * A collection of miscellaneous functions used in Snob-Online
 * @package snob
 * @category func
 */

/**
 * Return the current date in yearmonthday format
 * @return int Today's date from the server
 */
function getTodaysDate()
{
$today = getdate();
$mon = $today['mon'];
$mday = $today['mday'];
$year = $today['year'];
if ($mon < 10)
$mon = "0$mon";
if ($mday < 10)
$mday = "0$mday";
$returnval = "$year$mon$mday";
return $returnval;
}

```

```

/**
 * Return the current time from the server
 * @return int Current time in hoursminssecs format
 */
function getCurrentTime()
{
$today = getdate();
$hours = $today['hours'];
$mins = $today['minutes'];
$secs = $today['seconds'];
if ($hours < 10)
$hours = "0$hours";
if ($mins < 10)
$mins = "0$mins";
if (secs < 10)
$secs = "0$secs";

$returnval = "$hours$mins$secs";
return $returnval;
}

/**
 * Display the javascript code to redirect to another page
 * @param string $txt Target page
 */
function redirect($txt)
{
echo "<script language='JavaScript'>
window.location = '$txt';
</script>";
}

/**
 * Display a header in bold
 * @param string $txt Bold text
 */
function active($txt)
{
//echo "<td><b>$txt</b> | </td>";
echo "<b>$txt</b> | ";
}

/**
 * Display a header link
 * @param string $txt Link text
 */
function notActive($txt, $url)

```

```

{
//echo "<td><a href='$url'>$txt</a> | </td>";
echo "<a href='$url'>$txt</a> | ";
}

/**
 * Display header text
 * @param string $txt Text
 */
function disabled($txt)
{
echo "$txt | ";
}

/**
 * Format header links
 * @param string $txt Text
 * @param string $url Desired url
 */
function buttonOption($txt, $url)
{
$path = "/cgi-bin/cgiwrap/mlhol1/";
$self = $_SERVER['SCRIPT_NAME'];
// If currently displaying active path
if ($self == $path . $url)
active($txt);
else
notActive($txt, $url);
}

/**
 * Format header links based upon module location
 * @param string $txt Header text
 * @param string $url Url
 * @param string $check Check for existence of Session data in this module
 */
function buttonOptionCheck($txt, $url, $check)
{
if (session_is_registered($check))
{
$path = "/cgi-bin/cgiwrap/mlhol1/";
$self = $_SERVER['SCRIPT_NAME'];
if ($self == $path . $url)
active($txt);
else
notActive($txt, $url);
}
else

```

```

{
disabled($txt);
}

}

?>

#!/usr/local/bin/php
<?

/**
 * Process a command by sending the data via socket connection to the snob_server
 * @package snob
 * @category process
 */

require_once('header.cgi');
require_once('sessionxml.cgi');

$port = 6124;
$username = $_SESSION['username'];
$comm = $_POST['command'];

// Get command from input interface
if ($_POST['process'])
{
$comm = "doall " . $_POST['cycles'];
}
else if ($_POST['reportsubmit'])
{
$comm = $_POST['report'];
}
else if ($_POST['general'])
{
$comm = $_POST['command'];
}

$project = $_SESSION['project'];
$session = $_SESSION['session'];

require_once('header.cgi');
// Create socket connection
$socket = fsockopen("localhost", $port,$errno, $errstr,30);

//$socket = socket_create(AF_INET,SOCK_STREAM, SOL_TCP);
//$connection = socket_connect($socket, 'localhost', $port);

```

```

if (!$socket)
{

}
else
{
// Send socket
fputs( $socket, "$username:$comm");
$_SESSION['active'] = TRUE;

// Add command to the session xml
addCommand($comm, getTodaysDate() . getCurrentTime());
$filename = "users/$username/$project/sessions/$session/comlist";
if (is_writeable($filename))
{
$file = fopen($filename, "a");
fputs($file, "$comm\n");
fclose ($socket);

require_once('miscfunc.cgi');
// Return to session results
redirect('session-results.cgi');
}
else
{
print "File not writeable";
}
}
?>

#!/usr/local/bin/php
<?
/**
 * Display the active projects available within a user account
 * @package snob
 * @category project
 */

require_once('header.cgi');
// If account exists
if (chdir('users/' . $_SESSION['username']))
{
// Display form
print "<table>
<tr><td><h2>Projects</h2></td></tr>
<tr><td><form method='POST' action='data-manage.cgi'></td></tr>
<tr><td><select name='project' multiple size='5'>";

```

```

$dir = opendir('.');
while($file = readdir($dir))
{
// Read in each directory
if (is_dir($file))
{
// Ignore . and .. dirs
if (strcmp($file, ".") != 0 && strcmp($file, "..") != 0)
{
echo "<option value='$file'$>$file</option>\n";
}
}
}
closedir($dir);
print "<br></select></td></tr>
<tr><td><input type='submit' value='Submit'></form></td></tr>";
print "<tr><td><a href='project-new.cgi'>New Project</a></td></tr></table>";
}
else
{
print "This account is yet to be activated please contact the administrator<br>\n";
}

?>

#!/usr/local/bin/php
<?

/**
 * Form for creation of a new project
 * @package snob
 * @category project
 */

require_once('header.cgi');

$title = $_POST['title'];
$description = $_POST['desc'];
// Form processign
if (isset($title))
{
$user = $_SESSION['username'];
if (chdir('users/' . $user))
{
// create directory
mkdir($title);
$_SESSION['project'] = $title;

```

```

chdir($title);
// create project subdirectories
mkdir('sessions');
mkdir('data');

redirect('data-new.cgi');
}
else
{
print "User not properly initialized";
}

}
else
{
// Form output
$self = $_SERVER['PHP_SELF'];
print "<table>
<tr><td><h2>New Project</h2></td></tr>
<tr><td><form method='POST' action='$self'></td></tr>";
print "<tr><td>Project Title</td></tr><tr><td><input type='text' name='title' size='20'></td></tr><tr><td>Project Description</td></tr><tr><td><textarea rows='10' cols='20'></td></tr><tr><td><input type='submit' value='Submit'></td></tr>";
print "<tr><td><a href='project-new.cgi'>New Project</a></td></tr></table>";
}
?>

<?

/**
 * Base class for the interpretation heirarchy
 * @package snob
 * @category interp
 */
class Results
{
var $id;
var $data = array();
var $argument;
var $output;
var $command;

/**
 * Empty constructor
 */
function Results()
{

```

```
}

/**
 * Return the id of this command processor
 * @return int Command id
 */
function getId()
{
return $this->id;
}

/**
 * Return a count of the number of data items in this processor
 * @return int data count
 */
function countData()
{
return sizeof($this->data);
}

/**
 * Add a data item to the array
 * @param Data $dataobject data item
 */
function addData($dataobject)
{
array_push($this->data, $dataobject);
}

/**
 * Get the array of data items
 * @return array Array of data items
 */
function getData()
{
return $this->data;
}

/**
 * Set the output text for this processor
 * @param string $out output text
 */
function setOutput($out)
{
$this->output = $out;
}

/**
```

```
* Set the id for the processor
* @param int $id id
*/
function setId($id)
{
$this->id = $id;
}

/**
 * Get the argument string for this processor
 * @return string Argument string
 */
function getArgument()
{
return $this->argument;
}

/**
 * Set the argument string for this processor
 * @param string $arg argument string
 */
function setArgument($arg)
{
$this->argument = $arg;
}

/**
 * Get the output text for this processor
 * @return string output text
 */
function getOutput()
{
return $this->output;
}

/**
 * Get the command name for this processor
 * @return string command name
 */
function getCommand()
{
return $this->command;
}

/**
 * Set the command name for this processor
 * @param string $com command name
 */
```

```

function setCommand($com)
{
$this->command = $com;
}

/**
 * Write out the XML for this command
 * @param filehandle $file target file
 */
function writeXml($file)
{
fputs($file, "<result>\n");
fputs($file, "<command>" . $this->getCommand() . "</command>\n");
fputs($file, "<argument>" . $this->getArgument() . "</argument>\n");
fputs($file, "<output>" . $this->getOutput() . "</output>\n");
fputs($file, "<interpretation>\n");
foreach ($this->data as $datobj)
{
$datobj->outputXml($file);
}
fputs($file, "</interpretation>\n");
fputs($file, "</result>\n");
}

/**
 * Entry point for XML reading
 */
function readXml()
{
$blah = parseFile();
}

/**
 * Write interpretation output for this processor
 * @return string output string
 */
function writeOutput()
{
$str = "There is currently no interpretation code available for this function. This may b
return $str
}

/**
 * Parse an XML file to build the data structure for this processor
 */
function parseFile()
{
$username = $_SESSION['username'];

```

```

$project = $_SESSION['project'];
$session = $_SESSION['session'];
$id = $this->getId();
$fileSrc = "users/$username/$project/sessions/$session/" . $id . ".xml";

$GLOBALS['data'] = array();

$xml_parser = xml_parser_create();
xml_set_element_handler($xml_parser,"start" . $this->getCommand() . "Element","end" . $this->getCommand() . "Element");
xml_set_character_data_handler($xml_parser,"character" . $this->getCommand() . "Data");

xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, false);

if (file_exists($fileSrc))
{
if (!($fp=fopen($fileSrc,"r")))
{
die("Cannot open $fileSrc");
}
}
else
{
print "\nCannot find " . $fileSrc . "\n";
return false;
}

while (($data=fread($fp,4096))
{
if(!xml_parse($xml_parser,$data,feof($fp)))
{
die(sprintf("Xml error at line %d column %d ",
xml_get_current_line_number($xml_parser),
xml_get_current_column_number($xml_parser)));
}
}
xml_parser_free($xml_parser);
fclose($fp);
return $GLOBALS['data'];

}
}
?>

#!/usr/local/bin/php
<?

/**
 * First command input screen when a session is started

```

```

* @package snob
* @category session
*/

require_once('header.cgi');
$self = $_SERVER['PHP_SELF'];
$project = $_SESSION['project'];
$data = $_SESSION['data'];
$session = "";
$_SESSION['session'] = $session;
// create directory for new session

// form output
print "<table>
<tr><td><h2>Project: $project</h2></td></tr>";
print "<tr><td><h2>Data : $data</h2></td></tr>";
print "<tr><td><h2>Session : $session</h2></td></tr>";

print "<tr><td><form method='POST' action='process-command.cgi'></td></tr>";

print "<tr><td>Command</td></tr><tr><td><input type='text' name='command' size='20'></td><
print "<tr><td><input type='submit' value='Submit' name='submit'></td></tr>";
print "</form></table>";

?>
#!/usr/local/bin/php
<?

/**
 * Form to end an active session
 * @package snob
 * @category session
 */

require_once('header.cgi');
require_once('sessionxml.cgi');
$port = 6124;
$username = $_SESSION['username'];
$project = $_SESSION['project'];
$session = $_SESSION['session'];

$sessfile = "users/$username/$project/sessions/$session/sessdat.xml";
// Form processing
if ($_POST['submit'])
{
$socket = fsockopen ("localhost", $port, $errno, $errstr, 30);
if (!$socket)

```

```

{
print "No Socket $errstr ($errno)";
}
else
{
fputs($socket, "$username:endsess");
if (file_exists($sessfile))
{
closeSession();
addDescription($_POST['description']);
redirect('session-manage.cgi');
}
else
{
print "Session file doesn't exist<br>";
}
}
}
else if ($_POST['cancel'])
{
redirect('session-results.cgi');
}
// Form output
?>

<form action='<? print $_SERVER['PHP_SELF']; ?>' method='POST'>
Are you sure you wish to end this session?
<table>
<tr><td>Session Description: </td><td><input type='text' name='description'></td></tr>
<tr><td><input type='submit' name='submit' value='Submit'></td><td><input type='submit' na
</td></tr>
</table>
</form>

<?

/**
 * Command input interface
 * @package snob
 * @category session
 */

$data = $_SESSION['data'];
// Form output
?>

<hr width='400' align='left'><table>

<tr><td><form method='POST' action='process-command.cgi'></td></tr>

```

```

<table>
<tr><td><h2>Processing Commands</h2></td></tr>
<tr><td><table>
<tr><td>Process data for <input type='text' name='cycles' size='3'> cycles</td></tr>
</table></td>

<td><input type='submit' name='process' value='Process'></td>
</tr>

<tr><td><h2>Reporting Commands</h2></td></tr>
<tr><td><table>
<tr><td>Class Overview (sum)</td><td><input type='radio' name='report' value='sum'></td></tr>
<tr><td>Class Summary (prclass -1 1)</td><td><input type='radio' name='report' value='prcl<
<tr><td>Class Heirarchy (tree)</td><td><input type='radio' name='report' value='tree'></td>
<tr><td>Class Detail (trep)</td><td><input type='radio' name='report' value='trep <? print
<tr><td>Item Detail (mrep)</td><td><input type='radio' name='report' value='mrep <? print

</table></td>

<td valign='bottom'><input type='submit' name='reportsubmit' value='Submit'></td>

</tr>

<tr><td><h2>General Command Interface</h2></td></tr>

<tr><td><table>
<tr>
<td>Command</td><td><input type='text' name='command'></tr>
</tr>
</table></td>
<td><input type='submit' name='general' value='Submit'></td></tr>

</form></table>

<?
?>

#!/usr/local/bin/php
<?

/**
 * Results screen for sessions with interpretation enabled
 * @package snob
 * @category session
 */

```

```

require_once('header.cgi');
require_once('jscript-interp.cgi');

$project = $_SESSION['project'];
$username = $_SESSION['username'];
$data = $_SESSION['data'];
$ses = $_POST['session'];

if (isset($ses))
{
$_SESSION['session'] = $ses;
}
$session = $_SESSION['session'];

require_once('jscripts.cgi');
require_once('sessionxml.cgi');
// Interpretation handling
if ($_POST['interp'])
{
// Split the arguments from the comlist up
$c1 = $_POST['comlist'];
$clar = split(":", $c1);
$time = $clar[0];
$com = $clar[1];
$arg = split(" ", $com);
// Get new handler from factory
require_once('handlerFactory.cgi');
$hf = new handlerFactory();
$handler = $hf->getHandler($arg[0], $time, $arg[1]);
if (is_object($handler))
{
// Parse output
$filename = "users/$username/$project/sessions/$session/$time.xml";
$file = fopen($filename, "w");
$GLOBALS['parser'] = $handler;
$GLOBALS['parser']->readOutput($time, $_POST['result']);

// Write out new XML
$GLOBALS['parser']->writeXml($file);
$GLOBALS['parser']->writeOutput();
$GLOBALS['parser'] = "";
}
else
{
print "Handler failure<br>";
}
}

```

```

}
// Visualisation handling
else if ($_POST['visualise'])
{
require_once('dataParser.cgi');
$GLOBALS['dataparser'] = new DataParser();
parseDataFile();

$username = $_SESSION['username'];
$project = $_SESSION['project'];
$data = $_SESSION['data'];
// Requires that the trep file exists
$file = "users/$username/$project/data/$data.trep";
if (file_exists($file))
{
// Parse trep file
$GLOBALS['dataparser']->parseTrepFile();
$GLOBALS['dataparser']->writeXml();
// Provide link to xml data file for ggobi
print "<br>XML Visualisation <a href='http://www.datamining.monash.edu.au/cgi-bin/cgiwrap/"
}
else
{
// Instructions to output trep file
print "Please run the Class Detail reporting command before running visualisation";
}
}

// Form output
$file = "users/$username/$project/sessions/$session/output";

if (file_exists($file))
{
print "<table><form name='results' method='POST' action='\" . $_SERVER['PHP_SELF'] . "\">
<tr><td><h2>Session Output</h2></td></tr>
<tr><td><table><tr><td>Raw Snob Output</td><td><input type='radio' name='displaytype' value="
// insert radio button enabled handling here

print "checked onClick='interpUpdate();'></td><td>Interpreted Output</td><td><input type='
<tr><td><textarea name='result' rows='20' cols='80'></textarea></td>
<td>";
?>
<select multiple name='comlist' size='10' onChange="interpUpdate();">
<?

$sessions = readXml();
$session = $sessions[0];

```

```

$commands = $session->getCommands();

foreach ($commands as $command)
{
$time = $command->getTimestamp();
$commanddata = $command->getCommand();

print "<option value='$time:$commanddata'>$commanddata</option>";
}

print "</select></tr><tr><td><input type='submit' name='interp' value='Interpret'><input t

require_once('session-interface.cgi');

print "<tr><td><a href='session-end.cgi'>End Session</a></td></tr></table>";
print "<script language='JavaScript'>document.results.comlist.selectedIndex = 0;interpUpda
}
else
{
print "file not found";
}
?>

#!/usr/local/bin/php
<?php

/**
 * Output form for session selection
 * @package snob
 * @category session
 */

require_once('header.cgi');
require_once('sessionxml.cgi');
if (isset($_POST['data']))
{ // if post data exists
$data = $_POST['data'];
session_register('data');
$_SESSION['data'] = $data;

$_SESSION['session'] = FALSE;
}
else if (session_is_registered('data'))
{ // if no post data exists but session data exists
$data = $_SESSION['data'];

```



```

}
else
{
print "<option value='$file'>$desc : $id</option>\n";
}
}
}
}
}
}
print "</select></td></tr>";
print "<tr><td><input type='submit' value='Submit'></form></td></tr>";
closedir($dirs);
print "<tr><td><a href='session-new.cgi'>Start New Session</a></td></tr>";
}

?>

#!/usr/local/bin/php
<?

/**
 * Create a new session by passing information to the snob server using socket connection
 * @package snob
 * @category session
 */

require_once('header.cgi');
require_once('sessionxml.cgi');

$port = 6124;
$username = $_SESSION['username'];
$project = $_SESSION['project'];
$data = $_SESSION['data'];
session_register('session');
$_SESSION['session'] = getTodaysDate() . getCurrentTime();

// Socket connection
$socket = fsockopen ("localhost", $port, $errno, $errstr, 30);
if (!$socket)
{
print "<br><br>The Snob-Online system is currently offline<br>";
print "No Socket $errstr ($errno)";
}
else
{
$username = $_SESSION['username'];
mkdir("users/$username/$project/sessions/$session");
//$session = $_SESSION['session'];

```

```

fputs($socket, "$username:$project,$data,$session");
$_SESSION['active'] = TRUE;
$filename = "users/$username/$project/sessions/$session/comlist";
$file = fopen($filename, "w");
$str = "Session setup\n";
fputs($file, $str);
fclose($file);

// Pass session setup data to session xml
$sessfile = "users/$username/$project/sessions/$session/sessdat.xml";
$sessionobj = new Session($session, $username, $project, $data, "open");
$sessionobj->addCommand(new Command("Session Setup", getTodaysDate() . getcurrentTime()));
writeXml($sessionobj, $sessfile);

// Session command processing interface
$self = $_SERVER['PHP_SELF'];
$project = $_SESSION['project'];
$data = $_SESSION['data'];
$session = $_SESSION['session'];
$_SESSION['session'] = $session;
// create directory for new session
print "<table>
<tr><td><h2>Project: $project</h2></td></tr>";
print "<tr><td><h2>Data : $data</h2></td></tr>";
print "<tr><td><h2>Session : $session</h2></td></tr></table>";

require_once('session-interface.cgi');

}
?>

#!/usr/local/bin/php
<?

/**
 * Results interface for the session module
 * @package snob
 * @category session
 */

require_once('header.cgi');

// Get session data
$project = $_SESSION['project'];
$username = $_SESSION['username'];
$data = $_SESSION['data'];
$ses = $_POST['session'];

```

```

if (isset($ses))
{
$_SESSION['session'] = $ses;
}
$session = $_SESSION['session'];

require_once('jscripts.cgi');

$file = "users/$username/$project/sessions/$session/output";
// Form output
if (file_exists($file))
{
print "<table><form name='results'>
<tr><td><h2>Session Output</h2></td></tr>
<tr><td><textarea name='result' rows='20' cols='80'></textarea></td>
<td>";
?>
<select multiple name='comlist' size='10' onChange="update();">
<? $comlistfile = "users/$username/$project/sessions/$session/comlist";
$comlist = fopen($comlistfile, "r");
while (!feof($comlist))
{
$line = fgets($comlist,100000);
print "<option value='$line'>$line</option>";
}

print "</select></tr></table></form>";

require_once('session-interface.cgi');

print "<tr><td><a href='session-end.cgi'>End Session</a></td></tr></table>";
}
else
{
print "file not found";
}
?>

<?

class Command
{
var $id;
var $command;
var $timestamp;
var $snobresult;

function Command($command, $timestamp)

```

```
{
$this->command = $command;
$this->timestamp = $timestamp;
}

function getCommand()
{
return $this->command;
}

function getTimestamp()
{
return $this->timestamp;
}

function setSnobResult($result)
{
$this->snobresult = $result;
}

function getSnobResult()
{
return $this->snobresult;
}

function getId()
{
return $this->id;
}

function setId($id)
{
$this->id = $id;
}

}

class Session
{
var $id;
var $username;
var $project;
var $dataset;
var $status;
var $description = FALSE;
var $commands = array();
var $lastid;
```

```
function Session($sid, $susername, $sproject, $sdataset, $sstatus)
{
$this->id = $sid;
$this->username = $susername;
$this->project = $sproject;
$this->dataset = $sdataset;
$this->status = $sstatus;
$this->lastid = 0;
}

function setId($sid)
{
$this->id = $sid;
}

function getId()
{
return $this->id;
}

function getUsername()
{
return $this->username;
}

function setUsername($susername)
{
$this->username = $susername;
}

function getProject()
{
return $this->project;
}

function setProject($sproject)
{
$this->project = $sproject;
}

function getDataset()
{
return $this->dataset;
}

function setDataset($sdataset)
{
```

```
$this->dataset = $sdataset;
}

function setStatus($ssta)
{
$this->status = $ssta;
}

function getStatus()
{
return $this->status;
}

function setDescription($desc)
{
$this->description = $desc;
}

function getDescription()
{
return $this->description;
}

function addCommand($command)
{
$this->setId($lastid);
$lastid++;
array_push($this->commands, $command);
}

function getCommands()
{
return $this->commands;
}

}

// the xml file we want to parse

$username = $_SESSION['username'];
$project = $_SESSION['project'];
$session = $_SESSION['session'];

$xmlSource="users/$username/$project/sessions/$session/sessdat.xml";

//$xmlSource = "sessdat.xml";
```

```

$currentElement=""; //holds the name of the current element

/* The start Element Handler
* This is where we store the element name, currently being parsed, in $currentElement.
* the character data handler uses this to identify the element.
* This is also where we get the attribute, if any.
*/
function startElement($parser,$name,$attr)
{

$GLOBALS['currentElement']=$name;

// Handle element attributes
if ($GLOBALS['currentElement'] == "session")
{
$tmpid = $attr["id"];
$tmpuser = $attr["username"];
$tmpproj = $attr["project"];
$tmpdata = $attr["dataset"];
$tmpstat = $attr["status"];

$GLOBALS['sess'] = new Session($tmpid, $tmpuser, $tmpproj, $tmpdata, $tmpstat);

}

if ($GLOBALS['currentElement'] == "command")
{
$GLOBALS['tmptime'] = $attr["issued"];
}

} //end startElement()

/*
* The end Element Handler
*/

function endElement($parser,$name)
{
$elements=array('username','password','fullname','email');

if (strcmp($name,"command")==0)
{
$GLOBALS['sess']->addCommand(new Command($GLOBALS['command'] , $GLOBALS['tmptime']));
$GLOBALS['tmptime'] = "";
$GLOBALS['command'] = "";
}
}

```

```

}

if (strcmp($name, "description")==0)
{
$GLOBALS['sess']->setDescription($GLOBALS['description']);
$GLOBALS['description'] = "";
}

    if(strcmp($name,"session")==0)
    {
array_push($GLOBALS['sessions'], $GLOBALS['sess']);
    }

if (strcmp($name,"results")==0)
{
// get the current Command value
// $factory = new handlerFactory();
// $handler = $factory->getHandler($command);
//
}

} //end endElement()

/* The character data Handler
* Depending on what the currentElement is,
* the handler assigns the value to the appropriate variable
*/

function characterData($parser, $data)
{
    $elements = array ('command','description');

    foreach ($elements as $element)
    {
        if ($GLOBALS["currentElement"] == $element)
        {
            $GLOBALS[$element] .= $data;
        }
    }
}

/* This is where the actual parsing is going on.
* parseFile() parses the xml document and return an array
* with the data we asked for.
*/

```

```

function parseFile()
{
//global $xmlSource;
$username = $_SESSION['username'];
$project = $_SESSION['project'];
$session = $_SESSION['session'];

$fileSrc = "users/$username/$project/sessions/$session/sessdat.xml";
// $fileSrc ="sessdat.xml";
$GLOBALS['sessions'] = array();

/*Creating the xml parser*/
$xml_parser=xml_parser_create();

/*Register the handlers*/
xml_set_element_handler($xml_parser,"startElement","endElement");
xml_set_character_data_handler($xml_parser,"characterData");

/*Disables case-folding. Needed for this example*/
xml_parser_set_option($xml_parser,XML_OPTION_CASE_FOLDING,false);

/*Open the xml file and feed it to the parser in 4k blocks*/
if (file_exists($fileSrc))
{
if(!($fp=fopen($fileSrc,"r")))
{
die("Cannot open $fileSrc");
}
}
else
{
print "\nCannot find " . $fileSrc . "\n";
return false;
}

while(($data=fread($fp,4096)))
{
if(!xml_parse($xml_parser,$data,feof($fp)))
{
die(sprintf("XML error at line %d column %d ",
xml_get_current_line_number($xml_parser),
xml_get_current_column_number($xml_parser)));
}
}

/*Finish ! we free the parser and returns the array*/
xml_parser_free($xml_parser);

```

```

fclose($fp);
return $GLOBALS['sessions'];

} //end parseFile()

function readXml()
{
$s = parseFile();
return $s;

}

function writeXml($session, $path)
{
$file = fopen($path, "w");
if (!$file)
{
echo "Failed to open file<br>";
}
else
{
$header = "<session id='" . $session->getId() . "' username='" . $session->getUsername() .
fputs($file, $header);
fputs($file, "<commands>\n");

$commands = $session->getCommands();
foreach ($commands as $command)
{
$commandstr = "<command issued='" . $command->getTimestamp() . "'>" . trim($command->getCo
fputs($file, $commandstr);
}
fputs($file, "</commands>\n");
$descstr = "<description>" . $session->getDescription() . "</description>\n";
fputs($file, $descstr);
fputs($file, "</session>\n");
}

}

function addCommand($com, $tim)
{
global $xmlSource;
$sess = readXml();
$sessi = $sess[0];
$sessi->addCommand(new Command($com, $tim));
writeXml($sessi,$xmlSource);
}

```

```
function addDescription($desc)
{
    global $xmlSource;
    $sessions = readXml();
    $session = $sessions[0];
    $session->setDescription($desc);
    writeXml($session,$xmlSource);
}

function closeSession()
{
    global $xmlSource;
    $sessions = readXml();
    $session = $sessions[0];
    if (is_object($session))
    {
        $session->setStatus("closed");
        writeXml($session,$xmlSource);
    }
    else
    {
        print "Something is broke";
    }
}

function isActive()
{
    $sessions = readXml();
    $session = $sessions[0];
    $status = $session->getStatus();
    if (strcmp($status, "open") == 0)
    {
        return true;
    }
    else
    {
        return false;
    }
}

function outputTest($s)
{
    foreach ($s as $session)
    {
        print "id : " . $session->getId() . "<br>";
        print "username : " . $session->getUsername() . "<br>";
        print "dataset : " . $session->getDataset() . "<br>";
    }
}
```

```
print "status : " . $session->getStatus() . "<br>";

print "commands :<br>=====<br>";

$commands = $session->getCommands();
foreach ($commands as $command)
{
print "command : " . $command->getCommand() . "<br>";
print "timestamp : " . $command->getTimestamp() . "<br>";
print "=====<br><br>";
}

print "desc : " . $session->getDescription() . "<br>";

}

}

?>
```

Snob Server Module

```

#!/usr/monash/bin/perl -w

use IO::Select;
use IO::Socket;
use Session;
use Symbol;
$port = $ARGV[0] || 6124;

#connection socket
$lsn = new IO::Socket::INET(Listen => 1, LocalPort => $port, Proto=>'tcp');
die "Unable to open socket" unless($lsn);
$sel = new IO::Select( $lsn );

%sessions = ();

# Setup exit routines
$SIG{QUIT} = \&done;
$SIG{INT} = \&done;

print "SNOB Online server running on port $port\n";

while(@ready = $sel->can_read)
{
    # Wait for an input on a socket.

    foreach $fh (@ready)
    {
        if($fh == $lsn)
        {
            # Got a new connection.
            print "New Connection\n";
            $new = $lsn->accept;          # Create a new socket
            $sel->add($new);
        }

        else
        {
            print "Data socket recieved\n";
            # Process socket
            $str = <$fh>;
            print "Full String: $str\n";
            if ($str)
            {

                $str =~ s/[\n\r]//g;          # Remove any \n or \r
                $str =~ tr/\0-\37\177-\377/./; # Remove control chars.
            }
        }
    }
}

```

```

#split the socket text up into account name and command
@test = split /:/, "$str";
$user = $test[0];
$string = $test[1];

print "Account: $user\n";
print "End String: $string\n";
# No session currently exists for this username
if (!exists($sessions{$user}))
{
if ($string eq "endsess")
{
print "End Session called when session no active session exists\n";
}
else
{
# Create session
print "Session doesn't exist under the username $user\n";
@spltstr = split /,/ , "$string";
$project = $spltstr[0];
$data = $spltstr[1];
$session = $spltstr[2];

$sessions{$user} = Session->new($project, $session, $data);
if (defined($sessions{$user}))
{
print "Session defined\n";
$val = $sessions{$user}->sessionReady();
print "Ready: $val\n";
if ($sessions{$user}->sessionReady())
{
print "Session ready\n";
$data = $sessions{$user}->getData();
$project = $sessions{$user}->getProject();
$session = $sessions{$user}->getSession();

# Work out data paths
$file = "../sessions/$session/output";
$datapath = "users/$user/$project/data/";
$snobpath = "../../../../../snob-vanilla/snob-vanilla";

chdir($datapath);
print("Data dir: $datapath\n");
system('pwd');
my $fh = gensym();
open ($fh, "| $snobpath > $file");
print "Snob session created for user: $user\n";
#start snob session

```

```

$fh->autoflush(1);
$sessions{$user}->setSnob($fh);

$datasub = $data . ".vset\n";
$sessions{$user}->printSnob($datasub);

$datasub = $data . ".samp\n";
$sessions{$user}->printSnob($datasub);
#print $fh $datasub;
chdir("../..../..");
}
else
{
print "Snob session not ready\n";
}
}
else
{
print "Session not defined";
}
}
}
else
{
# Continue processing for this session
print "Session found under the username $user\n";
if (defined($sessions{$user}))
{
if ($sessions{$user}->sessionActive())
{
if ($string eq "endsess")
{
print "Closing Session\n";
#send stop command to snob
$sessions{$user}->printSnob('stop');

# close file handle
close($sessions{$user}->getSnob());
# delete hash entry
delete $sessions{$user};
}
else
{
$snobsess = $sessions{$user}->getSnob();
print $snobsess "$string\n";
}
}
}
}
}

```

```

    }
    }
    else
    {
    print "Something is broke";
    }

    }

foreach $key (keys(%sessions))
{
$val = $sessions{$key}->getSnob();
print "$key : $val\n";

}

}
else
{
print "No String recieved\n";
}
$sel->remove($fh);
# close filehandle
close($fh);
}
    }
}

sub done {

#close all the stored filehandles
    $lsn->close;
    print "Shutting down\n";
    exit;
}

package Session;
use FileHandle;
sub new
{
my ($class) = shift;
my ($self) = {};

bless $self, $class;

$self->{'PROJECT'} = $_[0];
$testpro = $self->getProject();

```

```
print "Project : $testpro\n";

$self->{'SESSION'} = $_[1];
$testses = $self->getSession();
print "Session : $testses\n";

$self->{'DATA'} = $_[2];
$testdat = $self->getData();
print "Data : $testdat\n";

$fh = FileHandle->new();
$self->{'SNOB'} = $fh;
return $self;

}

sub setProject
{
my ($self) = shift;
$self->{'PROJECT'} = $_;
}

sub getProject
{
my ($self) = shift;
return $self->{'PROJECT'};
}

sub setSession
{
my ($self) = shift;
$self->{'SESSION'} = $_;
}

sub getSession
{
my ($self) = shift;
return $self->{'SESSION'};
}

sub setData
{
my ($self) = shift;
$self->{'DATA'} = $_;
}

sub getData
{
```

```

my ($self) = shift;
return $self->{'DATA'};
}

sub setSnob
{
my ($self) = shift;
my $fh = shift;

$self->{'SNOB'} = $fh;

}

sub getSnob
{
my ($self) = shift;
$fh = $self->{'SNOB'};
return $fh;
}

sub printSnob
{
my ($self) = shift;
my ($val) = shift;
my $fh = $self->{'SNOB'};
print $fh $val;
}

sub sessionReady
{
my ($self) = shift;
if (defined($self->getData()) and defined($self->getProject) and defined($self->getSession))
{
return "1";
}
return "0";
}

sub sessionActive
{
my ($self) = shift;
if (defined($self->{'SNOB'}))
{
return "1";
}
return "0";
}

```

1;