Exploratory Information Visualization Editing

Author: Lukas Roberts

School of Computer Science, Bangor University, May 2012

Supervisor: Jonathan C Roberts

Acknowledgements:

Blank at the moment.

Statement of originality & release

Statement of Originality

The work presented in this dissertation is entirely from the studies of the individual student, except where otherwise stated. Where derivations are presented and the origin of the work is either wholly or in part from other sources, then full reference is given to the original author. This work has not been presented previously for any degree, nor is it at present under consideration by any other degree awarding body.

Student

……

Statement of Availability

I hereby acknowledge the availability of any part of this dissertation for viewing, photocopying or incorporation into future studies, providing that full reference is given to the origins of any information contained herein.

Student

……

Abstract

Current visualization tools offer a rich set of interaction techniques that focus on visualization, analysis, computation, and data acquisition, with less support for visual production and the presentation of information. A demand remains for information visualization tools that enable users to be creative and provide design support. This project focuses on the concept of visualization editing; whereby a visual design space enables the user to investigate alternative views of the data. The software developed within this project extends a previous prototype developed to support a visualization-editing model. The developed software implements new visualization structures, plot customizations, and exploratory visualization techniques to extend and evaluate the concept of visualization editing. The software is designed in relation to its extension of the previously defined visualization-editing model. Results from user participation define extended features to visualization editing as worthwhile, specifying the merits of features within the software. The interactive performance of the software is evaluated to address the concept for the use of visualization software in a public context.

Contents

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

2.1 Research Area

The domain of Information visualization involves generating visual structures based on data. Users change parameters to adapt the visualization to illustrate a hypothesis or derive insightful information that is relational to the data. The overall goal of this process is for the user to create a visualization that amplifies cognition [1]. The interactive process that defines the visual depiction is important as it provides more insight into the data.

Visual analytics involves gaining knowledge from data by searching for visual patterns through a process of interactive visual exploration. Techniques such as brushing ranges and creating subsets make these visual patterns instantly recognizable. Modifications to the plot structure can reveal different perspectives of the data that were previously unrecognizable. The overall goal of exploratory visual analytics is to comprehend knowledge from the data that can be visually represented in an appropriate way [2].

2.2 Research Problem and Motivation

Within common editing tools such as OmniGraffle, users can create visual depictions that exist in there imagination. This is achieved through the use of common editing operations such as drag, drop, copy, paste, as well as history lists to support backtracking to previous actions. These tools also focus on the form of the graphical creation, with specific functions such as snap grids and rulers to aid the user in the process. Information visualization software rarely utilizes common editing operations available in commercial tools, as the focus is on exploiting methods such as visualization, analysis, composition and data acquisition. More focus is required with methods such as production, presentation, and dissemination as these factors affect the output of the process that defines the visualization.

The process of defining knowledge from data is an exhaustive process that can combine mapping the raw data to any number of visual structures to best represent it. Often users may want to generate simple forms of structures based on queries of the data, this indeed still requires the user to program some form of mapping conversion for the data into a visual structure. Once the data is mapped it may not represent the data within the best form, thus the requirement for interaction to manipulate the visual structure to derive selective subsets of data is required. A process that aids the user through creating alternative visual depictions that could be presented and edited in a way that promotes visible thought would provide beneficial to the problem of data exploration. This process is defined as visualization editing.

2.3 Aims and Objectives

The major aim of the project is to extend the process of visualization editing; to achieve this aim the current prototype Edivis (Editor for Information Visualization) shall be extended with specific features. These specific features correspond to the three objectives of the project.

The first objective is to enhance the number of visualizations that exist within the prototype, currently only parallel coordinates and star plots exist. If more visualization types are implemented this will give users a broad range of selection as to which type will best represent data.

The second objective is to implement common editing features into the prototype. Currently the prototype only supports basic operations such as copy and cut. More advanced operations such as resizing, colour modification, and spatially defining the location of axes for visualizations are required to achieve correspondence to common editing features within other software packages.

Finally the third objective is to provide methods that enhance exploratory visualization. The process of exploratory visualization includes common actions such as drilling down into the data, and finding common values across subsets of data. The extended software should aid this process by providing helpful features that allow the user to easily visually explore meanings within data.

The process of visualization editing will need to be evaluated through a user study to analyze if it is beneficial within exploratory visualization.

2.4 Structure

The organization of this thesis is laid out accordingly, first the concepts and terms used through the project are explained within Chapter 2 (the background). This includes visualization types (Section 2.2), forms of interactive techniques (Section 2.3), and visual analytics (Section 2.4).

Alternative research shall be discussed in relation to visualization editing within Chapter three (Related Work). This includes areas such as common editing programs such as text and video editors (Section 3.2). High and low level programming languages (Section 3.3). Information visualization models that attempt to encapsulate visualization processes (Section 3.4). Programming toolkits that enable high level programming of common information visualization elements (Section 3.5). Finally visualization software that relates to features of information visualization editing (Section 3.6) shall be defined and discussed.

The current prototype built within Processing is explained, whilst the Put and Pickup model are defined within chapter four. The software is defined in terms of specification (Section 4.3) and architecture (Section 4.4).

Designs for the extension to the prototype are created and considered within chapter five (Design). This includes the creation of three designs (Sections 5.3, 5.4, and 5.5), before a final design was created (Section 5.6).

The development of the extension is documented through chapter six (Implementation). This includes the discussion and rationale behind using particular technologies (Section 6.2). Creating a new menu system that better corresponds to actions within visualization editing (Section 6.3). New visualization structures are implemented to the software with a refinement in the architecture of the prototype (Section 6.4). Features that correspond to common editing operations are provided and implemented (Section 6.5). Exploratory visualization is handled with several features that aid the user with the process of visualization editing (Section 6.6). Finally the development of a user walkthrough, tasks and a survey are considered respective of the implementation (Section 6.7).

The process of visualization editing is evaluated through the use of the system within a user study in chapter seven (Results). The results of user tasks, the user survey, and user plot evaluation are all discussed within Section 7.2. The performance of the implementation is also considered within Section 7.3.

Chapter 9 provides a discussion of the results and major issues with the implementation in relation to visualization editing.

Finally Chapter 10 draws conclusions to what has been learnt from the project, as well as describing what work can be done in the future.

2.5 Summary

The domain of information visualization and specific applied domains such as visual analytics were introduced. Current focuses to the process of exploratory visualization within information visualization were identified, with a specific need for functions of editing to be applied. The need for visualization editing subsequently defined the aim of the project, with individual tasks that relate to the objectives discussed. Finally the structure of the thesis has been presented to provide an overview of the continuing chapters.

Chapter 2: Background

2.1 Overview

There are several important concepts within the domain of information visualization. These concepts include interactive techniques, and graphical representations of data. Separate graphical representations of data such as parallel-coordinate plots, scatter matrices, scatterplots and stacked bar graphs will be demonstrated and explained. Interactive techniques such as brushing, and multiple coordinated views will be explained to illustrate what the techniques do as well as how they affect visualizations. Finally the domain of visual analytics will be discussed to provide an overview of how users utilize visualization tools to interpret data.

2.2 Visualizations

2.2.1 Parallel Coordinate Plots

Parallel co-ordinate plots yield a graphical representation of multi-dimensional relations rather then just finite point sets; they are a system for doing and visualizing analytic and synthetic multi-dimensional geometry [3]. Alfred Inselberg published the concept for Parallel coordinate plots whilst working at IBM in 1985 after a series of technical reports from 1981. The concept of a parallel co-ordinate plot is that as opposed to an axis being placed orthogonally such as in scatterplots, they are placed equidistantly in parallel on a single plane, where we present a vector $(x1, x2, xd)$ as points on each axis $(a1, a2, ad)$, as illustrated in figure 1. Points on the plots axes are connected by lines to one another, a full representation of these connected lines is known as a polyline, which is a singular representation of a vectors points across all the axes.

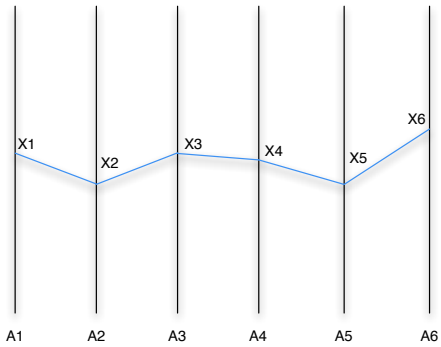

Figure 2-0-1: A parallel co-ordinate plot where points in a vector ($x1..xn$) will be plotted on multiple axes (a1 .. an)

Parallel co-ordinate plots represent data through individual scales for each axis, that is each axis has its own minimum and maximum data value that relate to the specific data attributes to be plotted on that axis. This is useful as it allows data of different unit dimensions to be plotted per axis meaning that the data is accurately plotted. Parallel co-ordinate plots also have an ability to represent nominal multidimensional data through mapping the values into small integers, this allows nominal data to be treated like ordinal data on an axis and connected to other ordinal data on different axes. Point and line duality is another important feature in the representation of the parallel co-ordinate plot; this is the representation of a set of points in a traditional two-dimensional plane where several points lie on one line. This line can be represented in a parallel co-ordinate plot as a set of connecting lines where the lines are either parallel or they intersect at one point. This means a line that has several points in an orthogonal two-dimensional plane can be represented as a point in a parallel co-ordinate plot and vice versa, this concept is illustrated in figure two.

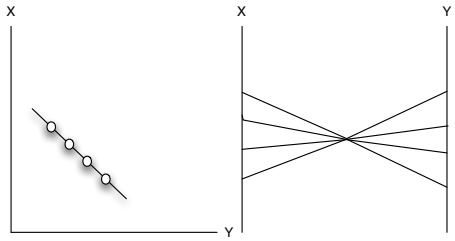

Figure 2-0-2: Point line duality illustrated from points on a line in two dimensional orthogonal space, that in a parallel co-ordinate plot form an intersection point

Advantages of parallel co-ordinate plots are clear in their representation of multivariate geometry in three-dimensional Euclidean space. Since parallel co-ordinate plots are two-dimensional geometric representations, conversions of data and graphs into dimensions higher then three is not necessary. Axes in a parallel co-ordinate plot can be rearranged in a number of layouts; since there are n dimensional data items there are $n!$ possible arrangements, however because there is repetition in the rearrangements, in order to ensure all axes are adjacent there are only $\lceil (n+1) \,/\, 2 \rceil$ possible reconfigurations. An example of this is having a three-dimensional dataset, this dataset can be re-arranged in six different configurations on a parallel co-ordinate plot, however there is only the need for two separate configurations to show the differing data points.

A problem with parallel coordinates is occlusion; this is where over plotting occurs within parallel coordinates, which is caused by a large set of polylines in the same area, or overlapping polylines where the values are mostly the same. Several approaches have been implemented to tackle the problem of occlusion. One solution is to implement cubic and quadratic curves instead of straight lines [4]. This approach allows otherwise overlapping straight lines to be distinguishable from one another because a curve is a product of the steepness of an original line between two adjacent axis, and the gradient of the previous line section, as is illustrated in figure three. Having more curves on screen is helpful as it is a better representation of the data but can provide problematic, as it becomes harder to differentiate the curves.

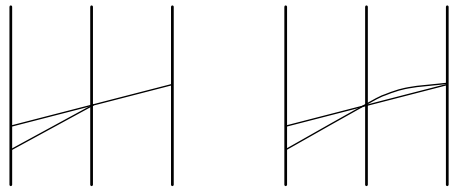

Figure 2-0-3: the left illustrates two straight lines overlapping with the right showing the same data but using quadratic or cubic curves

Occlusion can also be avoided by changing the opacity of polylines; this makes overlapping polylines appear darker giving a visual identification to polylines that overlap. The nature of this technique is user driven, as transparency adjustment is entirely dependent on the nature of the data set as well as the context it is being used within the parallel co-ordinate plot. Clutter reduction helps to overcome occlusion by using a sampling lens [5]; this technique allows the user to move a lens over data within a parallel co-ordinate plot to see an enhanced visual overview of the polylines. Lens sampling primarily works by having the user select a sampling rate of how much data they want to be sampled within the lens, this has been extended to be automatically based on what the user selected within the lens on the parallel co-ordinate plot.

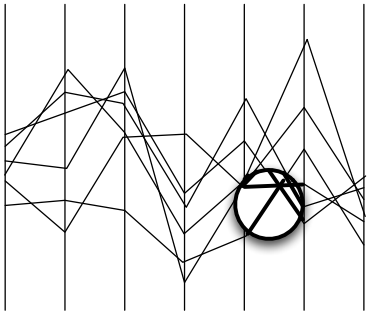

Figure 2-0-4: Clutter reduction illustrated in a parallel co-ordinate plot with a sampling lens highlighting some of the polylines with a default sampling rate of one

Highlighting is the most common form of brushing within parallel co-ordinate plots; this involves highlighting brushed polylines a specific colour. Parallel coordinates also support other common brushing operations such as delete, mask, zoom, and label. Polyline selection is an individual feature of parallel co-ordinate plots whereby users can view the data values of a singular polyline with a specific brush in a table or on the display. Singular polyline selection is beneficial as it provides an overview of the polyline in regards to the dataset, but focus in terms of its data values. Brushing can be applied across multiple axes in a parallel co-ordinate plot, this is commonly known as multi-dimensional brushing. Ranges across multiple axes can be defined to give a concentrated view of the data. Siirtola and Raiha discuss multi-dimensional brushing with reference to their tool Parallel Co-ordinate Explorer [6]. Parallel Co-ordinate Explorer displays range limits on axes as triangles that are modifiable, this allows the user to continuously redefine there brushing selection across all dimensions. Parallel Co-ordinate Explorer also implements multiple multi-dimensional brushes; this allows the user to make comparisons between subsets of polylines, although problems occur when trying to represent polylines that occur in both subsets of brushes. Singular polyline selection and multiple multi-dimensional brushing are illustrated in figure five, an extract from interacting with parallel co-ordinates.

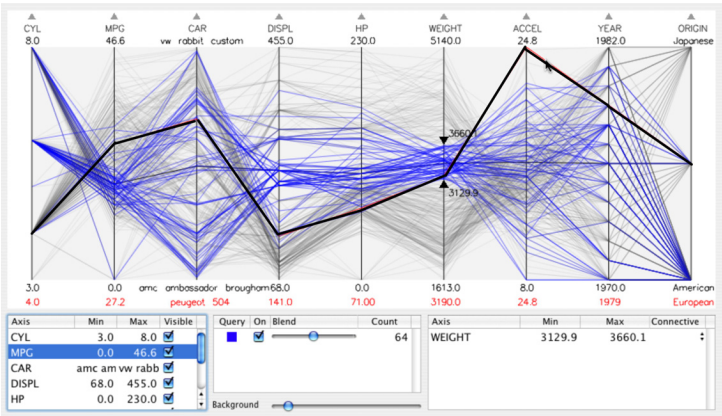

Figure 2-0-5: An extract from interacting with parallel coordinates (Siirtola & Raiha 2006) illustrating singular polyline selection and multi-dimensional brushing in the tool Parallel Coordinate Explorer.

Finally another form of brushing within parallel coordinate plots is angular brushing [7]. This treats the space between axes as a region of brushing interaction where the user can specify a subset of slopes. This subset visually alters the brushed selection of polylines by only displaying polylines that follow the subset criteria, as illustrated in figure six. Angular brushing is beneficial as the user can define a brush based on a combination of parameters. This is similar to composite brushes that are made of operations, however angular brushing works in respect to properties of the data within two dimensions.

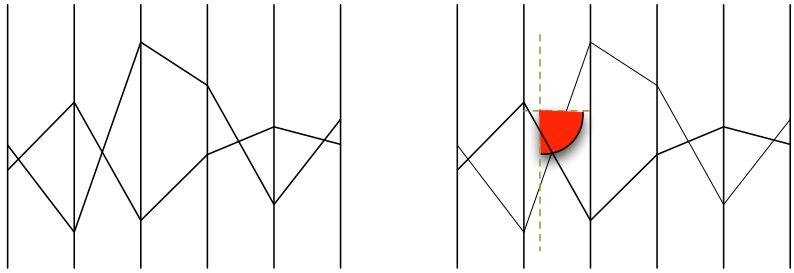

Figure 2-0-6: Angular brushing illustrated with a normal un-brushed parallel coordinate plot on the left, and a angular brushing based on a subset of slopes on the right

2.2.2 Scatterplots

Scatterplots are mathematical diagrams for displaying points based on two dimensions of data. Each dimension will affect how the point lies in the plot, so one dimension of data will affect the x and the other the y. Scatterplots are commonly used for finding correlations; these can be positive, negative or non. Positive correlations represent points plotted from the lower left to the upper right of the plot. Negative correlations represent points from the upper left to the lower right of plot. Non-correlations represent points all over the plot in no particular slope. Regression lines are another feature of scatterplots that help to identify the line of best fit in a plot or trend line. Positive correlations, negative correlations, non-correlations and regression lines are all illustrated in figure seven.

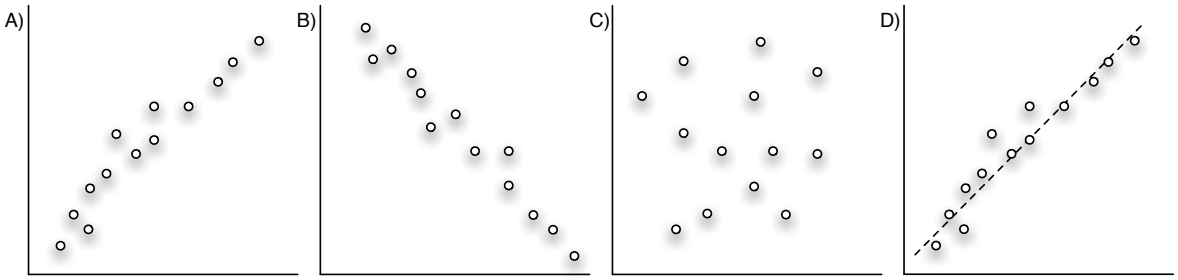

Figure 2-0-7: A) Positive correlation B) Negative Correlation C) No Correlation D) Regression Line

Scatterplots have been extended to be three-dimensional meaning that another dimension is available for the mapping of points. Typically scatterplots are visualized in three dimensions through a process called volume rendering that allows quadruples (x, y, z, w) to be mapped to a visual display. The main problem with this process is the depth perception from the output of two-dimensional devices. Pringer et al provide an implementation whereby users can interact with scatterplots in two-dimensions and the results will be displayed in three-dimensional scatterplots [8]. This allows users to interact with scatterplots comfortably but perceive results in an enhanced view.

2.2.3 Scatterplot Matrices

Scatterplot Matrices represent correlations between multidimensional data through a collection of scatterplots. If there are n dimensions within the dataset, then there will be n^2 scatterplots displayed in n rows and n columns. In terms of data in the scatterplots, a scatterplot in row i and column j will use data from the same dimensions; this allows the scatterplots to represent relationships between all the different dimensions in the data. An illustration of a scatterplot matrix is provided in figure 2-8.

Diagonal scatterplots within the scatterplot matrices represent the same dimensions of data for the x and y axis, for this reason a straight line is typically produced. Cui et al implemented a method that utilizes the diagonal space to display histograms, raster plots, and data driven plots. The histogram will display the data distribution of one dimension. Raster plots illustrate the trends of the data along a given order; this can be spatial order,

temporal order, or the order by one dimension. The data driven plots reveal the distribution of the data in two-dimensional space [9].

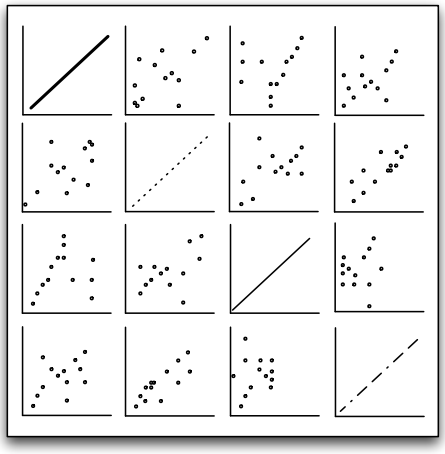

Figure 2-0-8: A four by four scatterplot matrix, resulting in 16 scatterplots that describe the relationships between the datasets dimensions

Common operations of brushing within scatterplot matrices include shadow highlight (drilling down). An extension to brushing within Scatterplot matrices includes dimension rearrangement and animation [10]. Dimension rearrangement involves moving the individual scatterplots into different row and column positions within the matrix allowing the user to see correlations and disparities between individual dimensions in the dataset. Animation allows the user to rotate individual scatterplots within the matrix.

2.2.4 Stacked Bar Graphs

Stacked bar graphs are based on traditional bar graphs that use the height of rectangles to represent the size of data typically along the y-axis. Stacked bar graphs use stacks of rectangles as opposed to single ones, these stacks are commonly known as segments of a whole stack. A collection of segments all stacked on one another represents the total of the data. This means that many dimensions of data can all be stacked on one another to represent a total. This gives an overview of the total of the data as well as a singular representation of the dimension in relation to that overview. Typically the individual stacks in a stacked bar graph are calculated as a percentage of there total. The total for a stack can be calculated as $percent = p/n$, where n is equal to how many stacks there are and p represents the height in pixels of the y-axis.

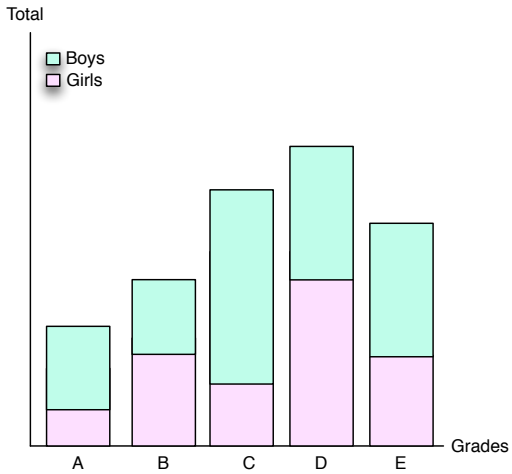

Figure 2-0-9: A stacked bar graph displaying grades for boys and girls in a class

2.3 Interactive Techniques

2.3.1 Brushing

Brushing is a collection of dynamic graphical methods for analyzing data in one, two, three, and higher dimensions [11]. The concept for brushing was defined by Becker and Cleverland who used it to link selections within scatter plot matrices. The idea in brushing it to use a brush to select a subset of data within a graphical plot, this makes the brush itself an important part of brushing. A brush can be defined as a shape that is superimposed on top of the screen; the brush may take multiple forms of shape most commonly a rectangle, point, ellipse, or allowing the user to specify the shape with something known as a freehand lasso, these shapes are illustrated in figure one. Brushes typically check for points through a binary operation where one will indicate a point is inside the brush and zero signifying a point is outside of the brush.

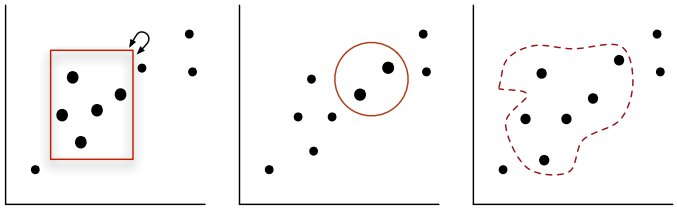

Figure 2-0-10: Brushing illustrated on scatterplots, in order from the left, rectangular brush, ellipse brush, freehand lasso brush

Users can move the brush across the graphical plot with a mouse and perform operations on the selection of data, this is commonly known as direct manipulation; the most common operations are highlight, shadow-highlight, delete, and label. Highlighting is a process in which the value of the brushed data is set to a colour. Shadow highlighting is the same as highlighting except for if there are multiple views with the same data then only the highlighted data in the active panel will be displayed across the other views. Deleting works in the same way as highlight except that the data is deleted instead. Label highlighting allows the user to toggle the visibility of plots labels for its axes; these methods are illustrated in figure two (with the exception of label highlighting).

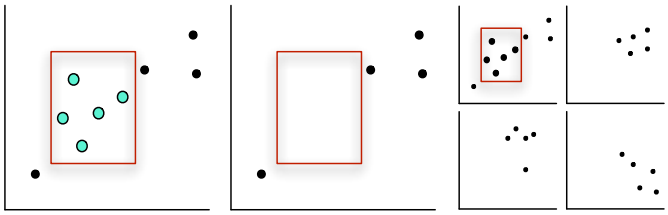

Figure 2-0-11: Illustrating-brushing operations, in order from the left: Highlight, Delete, Shadow-Highlight

Ward and Martin presented new direct manipulation techniques such as demand driven brushing and data driven brushing for the XmdvTool [12]. Demand driven brushing is where the brush is used to query the dataset, usually because of a demand or criteria from the user. Data driven brushing can be summarized by the presence of the data driving the brush specification when the user wishes to investigate something visually significant in the plot relating to the data. Ward and Martin also extended brushing to be n dimensional, composite and fuzzy. N-dimensional brushes can be used with other plots such as parallel co-ordinates and contain as many dimensions as that plot data has. Composite brushes are formed from combinations of brushes that are specified by rules such as AND, OR, and XOR. Composite brushes are different as they accommodate a ramping factor outside of the brush that instead of using the typical binary operation to check if points are inside the brush has a linear drop-off outside of the brushes boundary. Fuzzy brushes partially contain data points.

Dynamically linked brushing is the process of brushing the data in one view and linking this brushed data in other dimensions in multiple views. This is evident in graphical plots such as scatter-plot matrices that contain multiple views of the same data, interpreting this brushed data across multiple views can give the user a better understanding of the data and reveal trends. Dynamic linked brushing is part of multiple linked views and will be discussed further within that section.

Compound brushing is an example of where brushing has been extended to use a different model [13]. Compound brushing is a basis model that is useful for extending the typical model of brushing since it generically defines broad aspects of brushing within the model. Compound brushing consists of five types/entities: data, brushing selection, brushing device, brushing renderer, and transformation. The data is what data is investigated with the brushing techniques, this can be raw, meta, and derived data. Brushing selection presents the data that is covered by the brush. The brushing device is what type of brush is hosted to accommodate a specific type of plot, whether it is parallel, scatter, scatter-matrices, bar graphs and more. The brushing renderer is how the brush is graphically represented. Finally transformations are operations that are performed on the data or brushed selection in order to output another derived selection or result. Chen combined these entities to form higraphs to model brushing techniques and processes, where a simple example is shown in figure three.

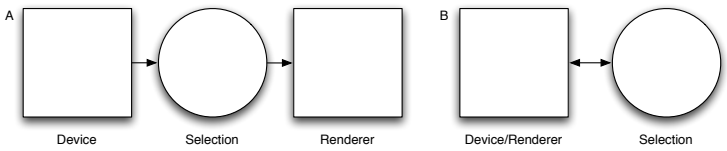

Figure 2-0-12: an example of higraphs based on the compound brushing model, (A) a device that generates a selection that is consumed by a renderer. (B) An entity that is used as a device as well as a renderer (Chen 2004)

Brushing has been extended to be ubiquitous, an interactive technique whereby anything in the visual display can be brushed [14]. Ubiquitous brushing has the advantage that any component in the visual display can be brushed over; this allows the user to understand the components meaning within the larger context of the visualization. This means that appropriate meaning must be given to components in the screen and the regions to brush components must be defined. Ubiquitous brushing presents the problem that the interface becomes transparent and the user may not understand what a particular operation will do. Ubiquitous brushing needs to be fast, this means that the appropriate use of advanced data structures needs to be utilized in order to keep a high refresh rate as well as a reactive system.

Brushing can also be manipulated indirectly with sliders and graphical components, this is known as indirect manipulation. Typically the process of using a slider generates dynamic queries; in definition this describes the interactive control of generated user query parameters that generate a rapid (within 100 millisecond update) animated visual display of results [15]. Dynamic queries allow a user to rapidly explore datasets with a constrained set of tools, allowing them to reveal trends, spot outliers, generate overviews, and many more data exploration tasks.

2.3.2 Coordinated Multiple Views

Coordinated Multiple Views (CMV) are an exploratory technique within visualization that enable the user to explore data. This is achieved by the user interacting with the data through multiple representations of data within different views. Multiple views are where data is displayed in multiple formats or views; this can be a tabular of visual representation. An example of multiple views is that where one view may be a scatterplot representation of data, another may represent the same data as a bar chart. Coordinated views are an extension to linked views, where the linked views relate to each other in some manner. In relation to the earlier example of multiple views, a coordinated extension of this would be the bar chart reflecting a selected subset of brushed points within the scatter plot (commonly known as drilling down). These examples are both illustrated visually in figure ten.

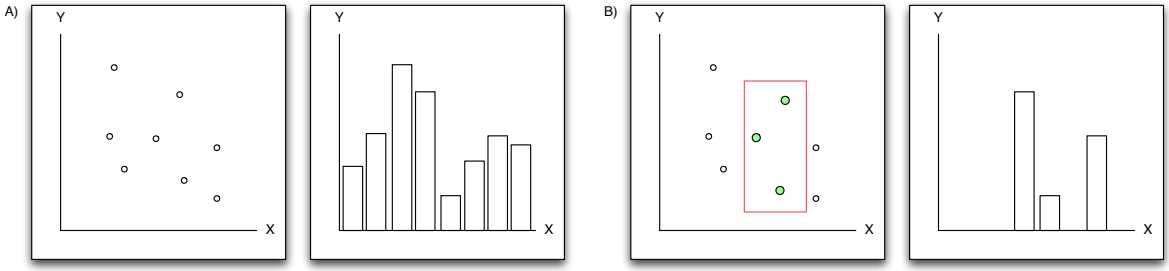

Figure 2-0-13: A) Illustrating linked views where a scatterplot in one window is a bar chart in another B) Illustrates the coordination of the same plots except the scatterplot has been brushed resulting in a updated view in the bar chart

Systems that only use two windows are dual view systems. There are some variants to dual view systems these being overview and detail, difference views, and master slave relationships. Overview and detail describes a system where one view is an overview and another the details. An example of this is Adobe Reader where pages of a document can be viewed as an overview to scroll through, with the page selected in the overview as the detail to read. Difference views merge two views together to visually represent the difference between two views. Master and slave relationships are relationships where one view controls another; for example navigating in one view will also navigate in another, this is called navigational scrolling and is discussed later in this section. This relationship is strictly one way where only interaction in the master view will affect coordination between views.

Design principles are important to consider in the construction of a CMV system, this is because CMV can be overloading both cognitively for users and resourcefully for systems. Aldonado, Woodruff, and Kuchinsky discussed approaches for design principles for CMV in relation to existing systems and experts experience in the field [16]. Eight principles are discussed for the design of CMV, where they are split into two general categories; the rules of selecting a multiple views and the interaction and presentation that come to light when using the system. Rules of selecting multiple views include diversity, complimentary, decomposition, and parsimony. Rules of interaction and presentation include space/time optimization, self-evidence, consistency, and attention management. These rules are all summarized within a table that is a direct copy from Guidelines for using multiple views in Information Visualization on page 118 [16].

Table 2.1: An overview of the eight design principles for designing Coordinated Multiple view applications, a direct copy from Guidelines for using multiple views in Information Visualization (14)

Rule	**Summary**	**Positive impacts on utility**	**Negative impacts on utility**
Diversity	*Use multiple views when there is a diversity of attributes, models, user profiles, levels of abstraction, or genres*.	Memory	Learning, computation overhead, Display space overhead
Complimentary	*Use multiple views when different views bring out correlations and/or disparities.*	Memory comparison context switching	Learning Computation overhead Display space overhead
Decomposition	*Partition complex data into multiple views to create manageable chunks and to provide insight into the interaction among different dimensions.*	Memory comparison	Learning Computation overhead Display space overhead
Parsimony	*Use multiple views minimally.*	Learning com putational overhead display space overhead	Learning Computation overhead Context switching
Space/time resource optimization	*Balance the spatial and temporal costs of presenting multiple views with the spatial and temporal benefits of using the views.*	Comparison computationa l overhead display space overhead	
Self-Evidence	*Use perceptual cues to make relationships among multiple views more apparent to the user.*	Learning Comparison	Computation overhead
Consistency	*Make the interfaces for multiple views consistent, and make the states of multiple views consistent.*	Learning Comparison	Computation overhead
Attention Management	*Use perceptual techniques to focus the user's attention on the right view at the right time.*	Memory Context switching	Computation overhead

Exploratory visualization (EV) has already been discussed in some respects with regards to Brushing, however it is important to understand how interaction within exploratory visualization affects CMV. Most interactive functions can be utilized within coordinated

views, the most common are navigational Slaving, coupling functions, and dynamically linked views. Navigational slaving is a common interaction technique within CMV; this is where movements in one view are shared in another. Coupling functions specify how an interaction is mapped over coordination between views. Dynamically linked views refers to the process of brushed linking where an interaction in one view will change that of others [15], however the subset that is brushed in one view is only displayed in others, this is equivalent to the shadow brushing operation discussed in section 2.1.1.

There are multiple options in the rendering of multiple views when parameters change, Roberts provides three general methods for rendering data in multiple views and how users will explore this data. These methods are replacement, replication, and overlay [17]. Replacement occurs when a user changes some parameter and the view is updated. Replication is where a view may be copied or specific information may be moved to a new window. Finally overlay is where new views would be placed over the other, this is the same as difference views in dual view systems however there may be more then two views being overlaid.

As has been discussed coordinated views can be linked with a relationship in terms of interaction. North and Shneiderman state four relationships in relation to their SNAP schemata [18], One to One, One to Many, Many to Many, and no relationship. Although these relationships are based on database design, when considered in a general approach these relationships can be applied generally to most CMV systems and models. One to one is having a parent-to-parent relationship where the data is the full representation on both views and any action performed in one view will be coordinated in the same manner. For example brushing over a selection of points in one view will simultaneously brush them in a linked view and vice versa. One to many is having a parent to many children relationship; this is where updates in the parent will be cascaded down to the children. As an example, imagine we have a dataset of animals in Britain that are categorically organized by species, so mammals, birds, fish etc. Species are plotted on a scatterplot whilst individual animals are plotted in a tree map. When a user clicks on a species selection in the scatterplot, only animals belonging to that species will be shown in the tree map. Many to many is a combination of one to many relationships, users point the relationship in the direction of there choice from plot to plot. Finally no relationship states that no relationships exist between views thus no coordination will occur. These relationships can represent coordination's between the linked plots, such as maintaining focus in one view whilst providing context in another, drilling down into the data, and details on demand. Both One to one and one to many relationships are illustrated in figure 11.

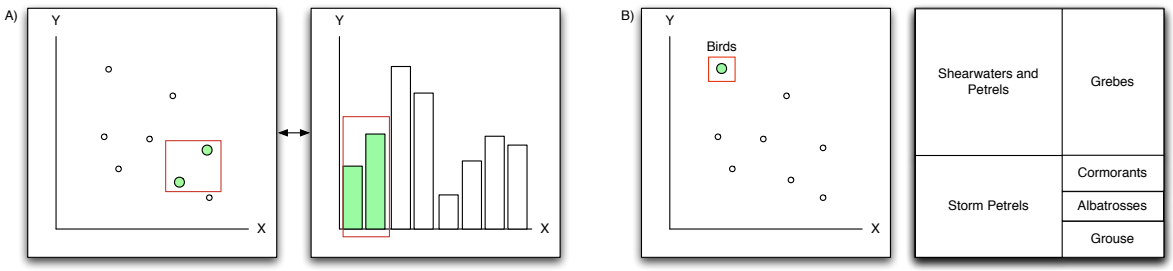

Figure 0-14: A) Represents a linked view between the scatterplot and bar graph, B) Represents drilling down where a point from a scatterplot represents a species where the tree view is updated

2.4 Visual Analytics

Visual analytics has been defined as the science of analytical reasoning facilitated by interactive visual interfaces [19]. Visual analytics is a cross-disciplinary research field that aims to create tools and techniques that facilitate people to derive insight and knowledge from massive, dynamic, ambiguous, and conflicting data. Visual analytics empowers people to gain knowledge through the process of explorative visualization that can involve factors such as proving a hypothesis and extracting the result.

Although there are overlapping principles, information visualization and visual analytics are separate domains. The key differences between visual analytics and information visualization are clear. Information visualization seeks to create novel graphical methods and models to represent data. Visual analytics is a merge of human interaction, data analysis and features of visualization, in ways visual analytics is finding the best-fit algorithm for the job of analyzing data.

Visual analytics starts with the raw data itself; raw data is a collection of meaningless numbers and characters that is made useful by analyzing, filtering and organizing from human input. Thomas and Cook state that our ability to collect this data surpasses the ability to understand it [19] since the nature of data is becoming more complex with the size and dimensionality. Approaches to analyzing data before it is graphically mapped have been developed in the form of automatic algorithms such as decision support trees, neural networks, and machine learning. These algorithms help to analyze raw datasets in relation to a predefined model to try and classify previously unseen data samples. It is a requirement of visualization tools to support large and complex datasets, with this requirement appropriate data structures and algorithms must be utilized to store and search the data based on the users demands.

Information overload specifically occurs when data is irrelevant to the task at hand, and is processed and presented in an inappropriate way. Keim et al state that one of the overarching driving visions of visual analytics is too turn the information overload into an opportunity by changing the way we process data and information transparently for an analytic discourse. This will foster the constructive evaluation, correction, and rapid improvement of our processes and models ultimately improving our knowledge and decisions [20].

Schneiderman describes a visual seeking mantra that is “over view first, zoom and filter, then details on demand”. Overview is gaining an over view of the entire dataset, zoom is zooming in on items of interest, filter refers to filtering out uninteresting items, and details on demand refers to selecting items of interest and obtaining the details about them [21]. Because of the increasing size and complex nature of datasets, the data is analyzed before and after it is visualized. Because of this Keim et al redefine Schneidermans mantra. The revision of this is “Analyze first, show the important, zoom filter and analyze further, details on demand” [2]. Analyze first involves investigating or finding the important information from the raw dataset, this important data is then visualized. Zooming and filtering operations are performed on the data except for with this we are also analyzing. Finally details on demand are displayed when something of interest is found.

Keim et al state that challenges within visual analytics include scalability, synthesis of heterogeneous data, interpretability, semantics, user acceptability, and evaluation. Scalability requires that systems can expand and accommodate for the rapid growth of data. Synthesis of heterogeneous data requires data from multiple sources to be joined together in one solution. Interpretability requires new graphical models to be created that

best represent the data that the user will understand and derive the most meaning from. Semantics require techniques to be developed to understand the complex relationships and correlations behind metadata from multiple data sources. User acceptability requires users to accept non-conventional methods of workflow to understand new models and systems that are implemented. Finally evaluation requires a deep analysis of the system itself to understand if it matches the best method of representing, interacting, and analyzing with the data.

2.5 Summary

Various visualizations such as parallel-co-ordinate plots, scatterplots, scatterplot matrices, and stacked bar graphs have been discussed. Design and interaction of Parallel-coordinate plots were established, with advantages and disadvantages discussed. Design of Scatterplots and scatterplot matrices were established with extensions to interaction of both discussed. Interactive techniques within visualization such as brushing and multiple coordinated views have been discussed to establish both techniques role within exploratory visualization. Brushing was discussed in terms of general techniques and models within the visualization process. Multiple coordinated views have been established in terms of definition, processes within visualization, and separate models that affect separate types of coordinated views. Finally the domain of visual analytics has been established to discuss how to best gain knowledge from data, and that traditional techniques and models may need to be refined in order to understand increasingly large and complex datasets.

Chapter 3: Related Work

3.1 Overview

In this chapter we provide an overview of the various software and technologies behind exploratory visual analytics within information visualization. First general commercial tools and there relation to the process of editing and creativity are defined and explained. Next low-level and high-level technologies and languages that are used to develop information visualization tools are discussed. Following this specific toolkits that have been developed specifically for the domain of information visualization are established and discussed. Finally an overview of existing information visualization software is provided with explicit advantages and disadvantages defined.

3.2 Editing and Creativity

Editing is the process of selecting and preparing written, visual, audible, and film media used to convey information through the processes of correction, condensation, organization and other modifications performed with an intention of producing a correct, consistent, accurate, and complete work [22]. In order to implement an editing tool in the domain of visualization, it is necessary to analyze editing tools in separate domains such as text, vector graphics, video, and animation. It is also necessary to understand how a user utilizes functions available within tools in these separate domains to understand the workflow that aids the user in creativity.

3.1.1 Text Editors

Text editors such as Microsoft Word™ and Pages™ allow users to create documents. Typically this is done through the process of drafting; meaning the user will build the document over time, starting from words, which become paragraphs that the user will then alter over time. This process is fundamental to how text editors aid the user in creating a document. Schneiderman writes that creativity utilizing software can be summarized with a Genex framework; this states creativity as a four phase process where users collect, relate, create, and donate. Within this framework Schneiderman states three areas to using tools such as text editors, low-level functions such as creating and changing words and there font, mid level functions such as reformatting paragraphs, and high level functions such as creating pre defined structures [23]. This three tier structure dictates the functions that are available to the user to support the completion of a task, for example when a user wants to create a bibliography, tools are available that allow citations to be added to a list in a pre-defined referencing format such as Harvard, once citations have been declared through the document, a bibliography can be generated automatically. Interactive Development Environments are source code editors that allow programmers to easily develop software. In terms of Schneidermans three tier model, the low level functionality is writing the code, mid level functionality is evident in error checking, formatting, and auto-complete, and high level functionality with the use of code snippets, compiling, and library inclusion (such as automatic class path generation). These functions aid users in developing software, as there are functions that are continuously at work in order to guide the user through the development process, an example of this is auto-complete; this feature makes the user aware of what methods and variables are available for use and contextually segregates the type of variables by colour coding. Features such as auto-complete allow users to creatively explore alternative ideas with ease, as well as find solutions to problems quickly and efficiently.

3.1.2 Vector and Video Editors

Vector editors are drawing tools that allow users to generate and edit vector graphics with which to build graphs, diagrams, animations and many other forms of visual media. Tools such as Adobe Illustrator™ help implement designs by building up vector shapes such as lines and polygons into objects. This object orientated approach to vector graphics gives users the ability to modify and perform operations on the separate components that create the object, this means that you can copy, cut, delete as well as visually customize what you are designing. Three-dimensional animation tools such as Blender and Autodesk Maya allow low-level tasks to be performed based on high-level functions within the software; an example of this is that some creative designers do not understand how to implement functions such as linear interpolation on an object, yet the software provides functions that allow this to be done at a higher level abstraction, where the user can choose what component of the object to apply the interpolation too. These tools are also expandable, as new technologies emerge, plug-ins will become available allowing the user to utilize this technology, an example of this is real-time performance based facial animation [24], whilst the basic version of Maya does not provide this feature, the developers of the emerging technology can implement a plug-in for Maya, that provides the user with the ability to use this new piece of technology but not have to understand its implementation to get the same results. Vector tools commonly provide informative metrics in the form of context specific popup menus relating to attributes of objects such as distances, scaling properties, and rotations, this makes it easier for the user to understand the relational properties of there actions and how they are being applied. Non-linear video editing tools such as Avid Media Composer™ and Adobe Premier Pro™ allow the user to compose video in a specified sequence. Linear editing of video used to be the case with film, as it would have to be spliced together, now that video can be captured in a digital format non linear video editors allow the creative freedom to edit video how users specify. Videos can be placed in one location but referenced from the software, allowing operations such as cutting, moving, and trimming to be applied to this video without altering the original source. Referencing video can also be useful in that it allows concurrent access to files so that they can be used within separate video tools at the same time, this helps separate teams in creating different products or ideas but utilizing the same resources.

3.3 Technologies and Languages

3.2.1 Application Programming Interface

Application Programming Interface (API) has a varying meaning in different contexts. In relation to graphics, Application Programming Interfaces are specifications of methods that can be utilized by programmers. The Open Graphics Language (OpenGL) is an example of a graphics API that is typically used to render two-dimensional and three-dimensional graphics on Graphical Processing Units (GPU). OpenGL contains specifications for geometric primitives, lighting, transformation matrices, texture mapping, and much more. OpenGL is generally used to create interactive applications; these can be in a wide variety of domains such as video games, simulations or visualizations. Within Information Visualization OpenGL provides the advantage that the tools will be portable and interoperable with the programmer having total control over the visualization methodology. OpenGL provides the disadvantage that it is completely code that is typically complex; this provides a steep learning curve for anyone who wishes to implement information visualizations.

3.2.2 High Level Programming Languages

In order to make programming easier, programming languages have been developed that do not require people to understand or have to code to the internal specifications of computers. These languages are high level programming languages; they are easier to understand where the code itself may be closer to the English language making them far less prone to error with the ability for rapid prototyping of ideas. Examples of high-level programming languages are Processing and Cinder. Processing is based on Java and is used to teach the fundamentals of programming in a visual context. Processing provides high level abstractions to link into low level APIs such as OpenGL, this means that users with low knowledge of graphical programming can learn in a simpler context. An example of this is illustrated in figure one, whereby both OpenGL code and Processing code is compared to create the same program; as can be seen the Processing code is much simpler with Processing also taking care of setting up the window context (only the draw display method is shown in OpenGL, there is a lot more code).

OpenGL

```
public void display(GLAutoDrawable drawable) {
    GL gl = drawable.getGL();
    gl.glClear(GL.GL_COLOR_BUFFER_BIT |
GL.GL_DEPTH_BUFFER_BIT);
    gl.glLoadIdentity();
    gl.glTranslatef(0.0f, 0.0f, -6.0f);
    gl.glBegin(GL.GL_QUADS);
    gl.glColor3f(0.0f, 1.0f, 0.0f);
    gl.glVertex3f(-1.0f, 1.0f, 0.0f);
    gl.glVertex3f(1.0f, 1.0f, 0.0f);
    gl.glVertex3f(1.0f, -1.0f, 0.0f);
    gl.glVertex3f(-1.0f, -1.0f, 0.0f);
    gl.glEnd();
    gl.glFlush();
  }
```

Processing

```
void setup() {
  size(300, 200,OPENGL);
}

void draw() {
  background(0);
  fill(0, 255, 0);
  rectMode(CENTER);
  rect(width/2, height/2, 50, 50);
}
```

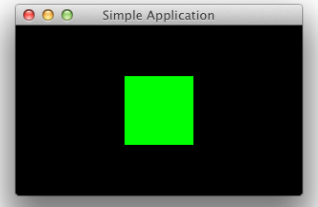

Figure 3-0-1: An illustration of low-level and high-level languages being used to create the same program

The main problem with high-level languages is that they still require the user to write code in order to create and interact with information visualization systems. There is not a general framework where the user can visually import, clean and map their data to generic

visualization types. This means that both sets of high and low level languages require a learning curve to use them to create interactive visualization systems specific to the users needs.

3.4 Information Visualization Models

Various models have been described to generalize the actions of a user within Information visualization. In this chapter general and adapted models are discussed to understand how the information visualization process can be generalized.

3.4.1 General Models

Model View Controller (MVC) is one of the most popular general models. The Model represents the system in its entirety, which can be divided into sub models, for instance a network model consists of sub models that represents aspects such as nodes and activities within the network. Views are visual representations of its model, for example a network tree can visually represent a data network. A controller is a link between a user and the system providing input and output for the views [25].

3.4.2 Novel and Adapted Models

Within Information Visualization a general model is the visualization pipeline; a five stage process that includes defining the problem, defining the data, adopting a processing language, processing the value or view, and interpret and deciding [26]. The visualization pipeline can be summarized as follows. Creating a hypothesis defines the problem. Once a hypothesis has been created raw data must be collected that can provide an answer to the hypothesis. Once raw data has been defined, operators must be specified that will provide effective functionality to the data. Once operations have been defined and a visual mapping made operations such as scaling, translation, rotation, and filtering need to be applied. The user must understand the view by analyzing and interpreting the view, and decide what further questions there are. Once further questions are established the process is iterated until the user has solved their hypothesis. This process is illustrated in Figure 3.2.

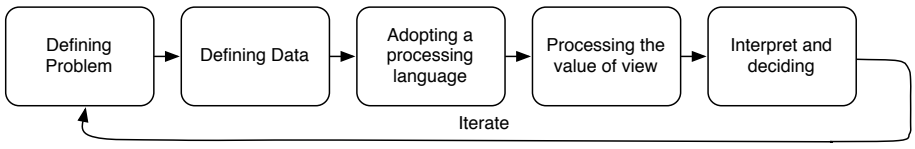

Figure 3-2: The visualization pipeline

Chi et al [27] specify an operator framework for visualization that attempts to specify all interactions within information visualization. The operator framework specifies two distinctive operators, view and value. Value operators refer to operations on the raw data itself such as mathematical operations or modifying or filtering the raw data to make a new subset. View operators only modify properties of the visualization with operations such as translation, rotation, and opacity; they do not affect the raw dataset in any way. The operator framework states five other operations between the view and value operators. All seven operators in the framework are summarized in Table 3.1 with relation to their specific functions and domains. The operations can be applied to information visualization systems in three contexts: within the internal infrastructure of an information visualization system, utilizing queries within data management engines, and aids for visualization systems and data management engines. The specific implementation of the information visualization framework will dictate what specific operators occur within it. This means that designers need to think more about the placement of operators within the visualization and how the operator's semantics apply to user interactions.

Table 3.1: Summaries provided of all the operations within Chi et als Operator Framework

Operator Name	**Operator Function**	**Operator Domain**
Data Stage Operators (DSO)	*Value filtering, sub setting, Difference or addition of two data sets*	Algebraic, Images, Point-sets, raw data, Web
Data Transform Operators (DTO)	*Calculating various transformation operations on the data, such as: computing textual vectors, surface extraction*	Textual, Grids, Points, Web
Analytical Abstraction Stage Operators (AASO)	*Selecting subsets or dividing regions*	Vector, Surface, Web
Visualization Transformation Operators (VTO)	*Searching through visual structures*	Dimension reduction, Clustering, Network
Visualization Abstraction Stage Operators (VASO)	*Visually cutting off or simplifying visual abstractions dependent on user interaction*	Grid, Network, Hierarchy
Visual Mapping Transformation Operators (VMTO)	*Visual mapping of data structure*	Point set, multi-dimensional surfaces, Hierarchy, Networks
View Stage Operators (VSO)	*Affect the visual environment*	Camera, Object Manipulation, View-Filtering

One of the specific advantages of the operator framework is that unlike the typical visualization pipeline it accommodates for multiple views and co-ordinations. This is accomplished by acting as a network with as many values and views needed as opposed to a singular pipeline. The operator framework provides a clear specification for the workflow of an information visualization application by distinguishing between view and value. This helps to identify components in information visualization that are interactive and editable. For example if the operator framework is applied to a scatterplot then several operators such as data stage operators, data transform operators, visual mapping transformation operators, and view stage operators can be applied to it. Essentially the framework can help to distinguish the semantic operation that a component has within information visualization editing.

Model-view-controller has been extended where the view is split into two domains, display views and picking views resulting in the Model - Display view - Picking view - Controller model (MDPC) [28] [29]. Display views refer to the typical view in the MVC model. Picking views are an invisible layer under the display view, whereby the specifications of interactions are described for the component. An example is illustrated in figure three, which shows a button with its display view and picking view. The picking view has two interaction states, a green selection and a dark grey selection. The green selection describes an interaction of being pressed. The dark grey selection describes the state of translation. When either the green or grey state is entered the appropriate interaction is performed.

Figure 3-3: The left is a display view; the right is a picking view

The goal of this model is to provide simplicity of design and implementation of coding; essentially the model improves the usability of programming. This model attempts to describe states of interactions by letting the user logically divide a component of visualization. This model is beneficial to the process of editing visualizations as it could be used to define a generic set of interactions for components within a system.

The P set model attempts to encapsulate how people interact with exploratory visualization systems [30] [31]. This is achieved by specifying interaction as four elements, visualization transformation, parameter results, visualization results, and derivations. Visualization transformations are identified by the transformations that are given as arguments, and the result that is generated. Parameters are defined from visualization types (such as scatterplots) and values (such as the data for a scatterplot). Unique collections of parameters define p-sets. Visualization results are defined from the p-sets and visual transformations that create them. Derivations are the main component of the model that contains specifications for p-sets and visualization results as well as a timestamp to identify the derivation itself. Figure four illustrates the model when applied to a brushing operation on a scatterplot. First parameters are selected from the scatterplot, from this a new p-set is created from the combination of the previous p-set. Finally this creates a new result and P-Set.

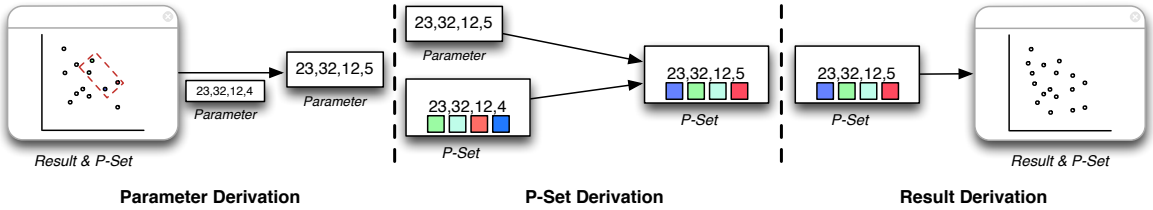

Figure 3-4: An illustration of how the separate derivations within the P-Set model are encapsulated

The model utilizes a library format so that users can include the model in their own visualization system. This makes it easier for users to utilize the model, as there are base types declared that correspond to elements of the model. This means the user just has to map their own specific transformations and parameters to these base types. In order to make the model results portable the Extensible Markup Language (XML) is used in order to format the model based on seven variables. This model provides an alternative approach to information visualization editing as it attempts to discover how parameters controlled by aspects of interactions affect visualizations. This can reveal user patterns and facilitate better design of information visualization systems.

3.5 Information Visualization Toolkits

Information visualization toolkits are essentially APIs that are specific to data structures, visual layouts, animations, and interactions within the domain of information visualization. They allow programmers to utilize high-level abstract representations of common features within information visualization, as opposed to having to program them from scratch such as in general low and high level languages such as Processing and OpenGL.

The InfoVis toolkit focuses on providing abstract methods for interaction within information visualization [32]. It is a library implemented in the Java language that abstracts processes relating to data structures, visualization, and interactions. Data structures are all stored within tables that use associative arrays. The tables also include meta-information about the specific columns. Visualizations transform semantic groups stored within the table. The toolkit includes three main types of data structure that map to visual structures. Scatterplots and time-series map to tables, node-link diagrams and tree-maps for trees, finally node-link diagrams and adjacency matrices are used for graphs. Interaction is supported for each visual structure through the use of permutations. Permutations specify and filter rows in the data table for a visual structure. Permutations are aided by the use of components within the framework. The framework supports a wide variety of components such as sliders, as well as specifying visualizations themselves as interactive components through specifying permutations for each visual structure. The InfoVis toolkit optimizes memory footprint, data locality, and filtering to be more efficient then the standard Graphics2D library that accompanies the AWT framework within Java. This allows users to be more expansive with there use of data when writing applications with the toolkit.

Prefuse [33] is a toolkit that uses the data state model [1] as a conceptual basis for implementing and structuring information visualization applications. The four stages of the data state model are used as logical separators within the framework as illustrated in Figure 3.4. An abstract data type named entities supports unstructured, graph, and tree data. Visual structures use entities in three ways. Node items represent singular entities. Edge items visualize relationships between entities. Aggregate Items represent aggregated groups of entities. These items do not affect the raw data as they are arranged in a local graph structure. This means that the framework is flexible in terms of how data is represented, as existing graph structures can be modified. Interactions are supported by Actions, a base class that handles events for filtering, colour, layout, animation, selecting and de-selecting visual data. The toolkit provides high-level abstractions for the use of force simulation, automatic layouts, interactive controls, colour maps, and event logging. In terms of efficiency, the toolkit uses the Java Graphics2D API; this means the toolkit does not provide any more efficiency in terms of rendering speed. Prefuses main area of focus is providing simplicity to programmers in terms of simple function calls. This is evidenced by the Degree of interest tree that was written with the toolkit as well as the user study that was performed. The toolkit manages to create a full interactive degree of interest tree in 48 lines of code where it could have taken a lot more time with low level and non-specific APIs. Whilst the users in the case study all built visualizations, with seven of eight finishing every task that was given to them.

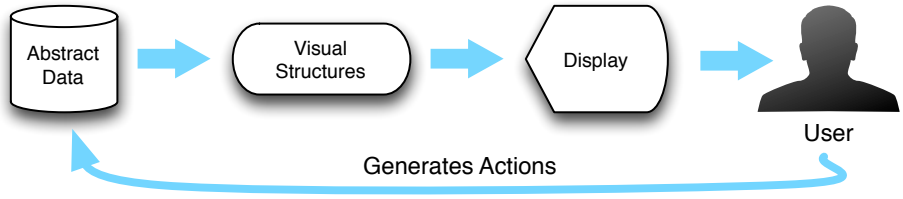

Figure 3-5: The data flow model adapted to how Prefuse uses it as a conceptual basis

Protovis is a toolkit that aims to make information visualization more accessible to web and interaction designers through enabling low-level control of the design [34]. Implemented using JavaScript and server side technologies, the framework is completely web based. Protovis utilizes a similar implementation as Prefuse in that it uses an adaptation of the data flow model. Data is stored as a concept called marks. Marks are rendered through the use of properties that are stored in a scene graph as new visual structures are created. Handlers are registered with marks to add interactivity. Protovis supports eight primitive Mark types, these are extensible where more can be added and different Marks can be combined. Protovis is not efficient, as benchmark tests would reveal 100,000 data points would require over 100mb of main memory. Protovis is simple as it allows a shallow learning curve, whilst giving users the ability to use simple primitives to generate custom visual structures.

D3 is an alternative to Protovis that provides a visualization kernel as opposed to a framework [35]. D3 is different from Protovis as it provides the manipulation of documents based on data by directly mapping data attributes to the Document Object Model (DOM). D3 focuses specifically on transformations, immediate evaluation, and native representation. D3 implements transitions as scene changes as opposed to the implicit methods in Protovis, this is much faster and does not require continuous rendering of the entire visual structure. Immediate evaluation helps users with debugging by moving internal control flow to the user code. Native representations provide specific functions for native primitives within SVG, for example Rect is a rectangle. Native representations mostly provide disadvantages within D3, as users cannot generate custom visual structures as they did with Marks within Protovis.

All the described toolkits help users by abstracting common Information Visualization tasks with simple function calls and objects within code. Summaries of the toolkits in comparison to one another are provided in Table 3.2. With the exception of Protovis all these toolkits generalize visualization structures. This means aspects such as interaction, visual structure, and data format are all pre determined denying the user the ability to generate and modify their own visual structures and visualizations in a visually orientated manner. Furthermore the focus of these toolkits is for developers to create new information visualization tools, as opposed to editing existing visualizations to explore the information visualization space.

Table 3.2: A summary of the Information Visualization toolkits discussed

Name	**Primitives**	**FPS with 20,000 data points**	**Rendering Engine**	**Platform**
InfoVis Toolkit	Tables, Trees, Graphs	60FPS	Agile 2D	Java Only
Prefuse	Node Items, Edge Items, Aggregate Items	Unknown	Java 2D	Java Only
Protovis	Marks (Line, Area, Bar, Dot, Image, Lable & Bar, Rule & Bar, Wedge	0FPS	SVG	Web
D3	Native Representations	5FPS	SVG	Web

3.6 Information Visualization Software

General and specific software tools and packages have been developed for the domain of Information Visualization. Various software and tools shall first be discussed in general. Following this data support, visualization structure support, interaction support, and finally editing support shall be discussed in relation to relevant tools.

Drawing with constraints provides an implementation named Briar, this software is a drawing program but includes concepts such as snap dragging and constraints on visual elements. Jigsaw is software that supports investigative analysis through parallel coordinated views of textual documents. XmdvTool [36] is a generic piece of software that supports multiple visualizations and interactions; it also has support for importing data and multiple coordinated views. Polaris [37] is a tool that rapidly visualizes table-based displays of multi-dimensional database data. Tableau [38] is a system expanding Polaris but attempts to generalize structures of visualization based on the type of data. Sage [39] is an automatic presentation system that generates complete or partial visualizations based on data. Sagebrush [39] allows the process of direct manipulation to edit visual structures that are generated within Sage. Sagebook [39] stores previously generated visualizations from Sage in an interface for users to browse and retrieve. Visage [40] is an extension to Sage that explores visualizations through parallel coordinated views to help users analyze and present their data. Selective Dynamic Manipulation (SDM) [41] is a set of techniques that enhance interactive capabilities of visualization components. Worlds within worlds [42] illustrates a technique that can represent multiple dimensions of data through the implementation of multiple worlds in one encompassing world, this is controlled with a data glove. Finally Flexible Linked Axis (FLINA) [43] defines axes as drawing objects and allows the user to edit and customize attributes of plots within the system. Screenshots from these systems are presented in Figure 3-5.

Data support has been classified as one of the most challenging tasks in visualization software (ref here). For this reason when building information visualization software a variety of sources have to be accommodated for, such as raw data which can be in any format of encodings. The XmdvTool accommodates for raw data as input through its own specific format that consists of ASCII alphanumerical fields separated by blank or new lines. Polaris and Tableau both use a custom database language called VizQL [44]. VizQL is based on Bertins semiology of graphics [45] and allows dynamic queries for visualizations by specifying its own relational algebra in context to visualizations. Raw data support is a lacking feature within visualization software because of the vast formats and encodings of data available, thus it is impossible to create a generic tool that reads and accepts all of them. Support for various types of data is another challenge of visualization software. Jigsaw, XmdvTool, Polaris, Tableau, Sage, SDM, Worlds within Worlds and FLINA all support nominal and ordinal data types. Spatial and temporal data types are only supported by World within Worlds, Tableau, Polaris, Visage, and SDM. The underlying data models for these tools represent how the raw data is stored internally within the program. The model component within the MVC model [25] is used to conceptualize a data model; Jigsaw and the XmdvTool both utilize the MVC model when storing data. Table based approaches are used for storing queries from Database Management Systems (DBMS). Polaris, Tableau, and Visage all use tables to store data internally from queries.

Basic types of visualization structures are scatterplots, bar graphs, line graphs and table data. Most of the software discussed can generate and visualize at least one of these structures with the exceptions of Briar and FLINA. Jigsaw, Polaris, Tableau, and SAGE

can represent more complicated structures such as trees and network graphs. Essentially the type of visual structure that can be represented by the software depends on the underlying formats of data that the tool supports. If the tool only supports nominal and ordinal data, then the choice of visualizations will be limited to general visualization structures. If the tool supports temporal and spatial types of data then the tool will be able to generate more advanced data structures to match the complexity of the data. SAGE supports the editing of generic structures as well as the generation of new structures by utilizing Sagebrush. Sagebrush provides a direct manipulation interface whereby the user generates graphics by using sketches from a library of existing designs. Nearly all elements of a visual component are customizable in Sagebrush, including the axes, spatial context, lines, and labels, with the ability to specify mappings for data for some of these components. Visualizations that render multivariate data such as Parallel coordinates can be represented by XMDV, FLINA, whilst scatter matrices can be represented by Tableau,

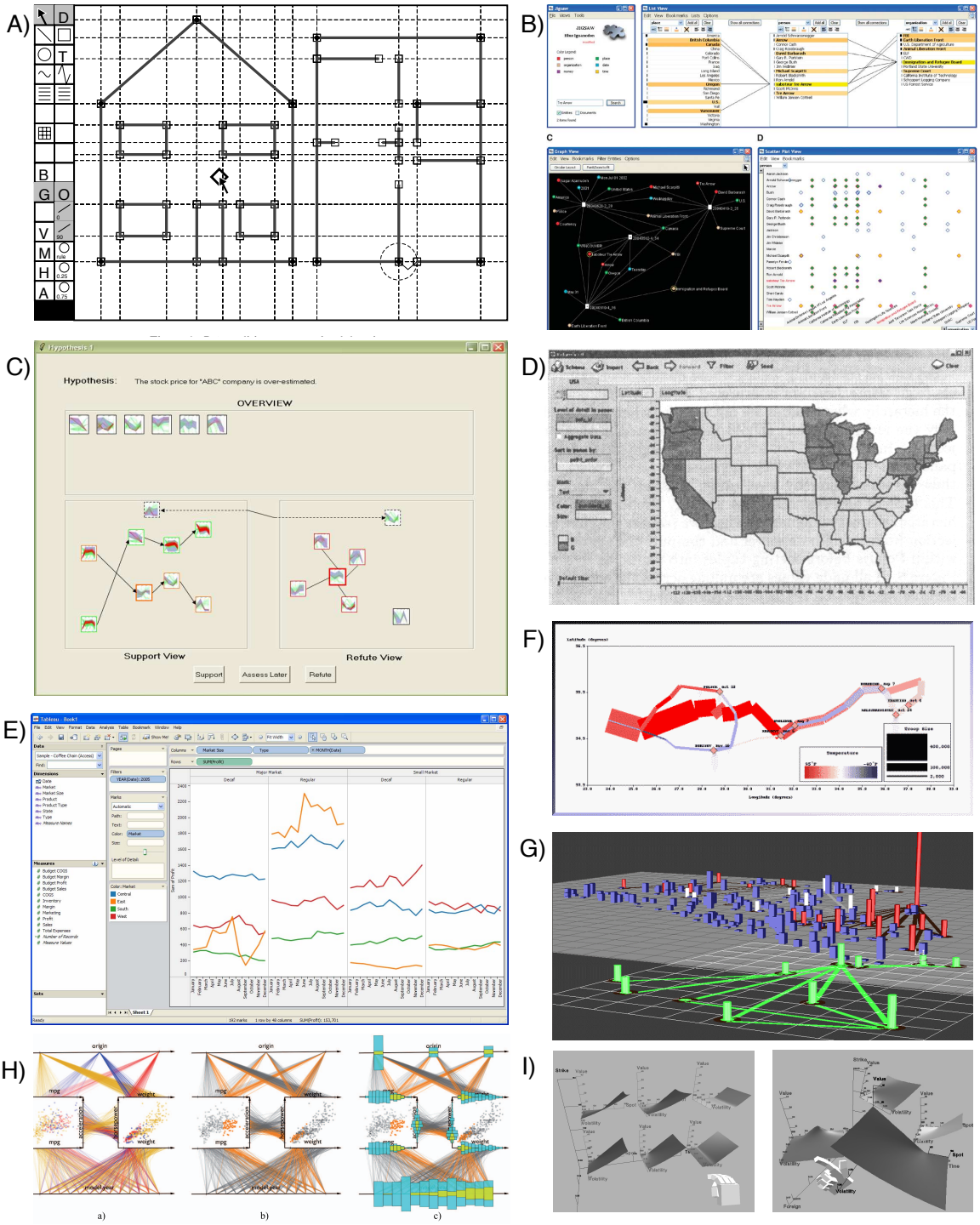

Figure 3-6: A) Drawing with constraints, B) Jigsaw, C) XMDV Tool, D) Polaris, E) Tableau, F) SAGE, G) SDM, H) FLINA, I) Worlds within worlds

XMDV, and FLINA.

The type of interaction that the tool provides depends on the visualization structures that are implemented. Worlds within worlds and SDM only support direct manipulation, this is because both pieces of software introduce novel interactive techniques and so do not focus on querying the data with existing methodologies. Polaris, and Tableau only support methods for indirect manipulation. This is because both tools generate queries from tables, input boxes, and sliders through the use of VizQL (that is not to say you cannot generate queries from VizQL with direct manipulation). Jigsaw, XmdvTool, FLINA, and Visage have support for both direct and indirect manipulation. These tools contextually support

these forms of manipulation, for example in Jigsaw direct manipulation is performed by clicking on words to view other instances of the same word, whereas in XmdvTool parallel coordinates are queried through specifying brushed sets with the mouse. Multiple coordinated views are supported by Worlds within worlds, SDM, and FLINA. Worlds within worlds naturally support multiple coordinated views through the technique itself. SDM supports multiple coordinated views through being able to copy data and move it to another place within the software. FLINA has an assortment of options for multiple coordinated views as users can specify nearly all attributes of plots. Polaris, Tableau, and Visage support copy and paste, and drag and drop. Polaris and tableau support local copy and paste and drag and drop within the software, whereas Visage supports local as well as interoperability with other software such as a mapping application called MATT.

Editing within visualization is a concept that has primarily only been addressed by FLINA. Editing concepts such as snap dragging and visual grid constraints are implemented in Briar, however this is only within a drawing context. SDM provides the options of directly editing elements in the display through rotating, translating and scaling visual elements using interactive handles. SDM maintains a context of data scaling through the use of separate scales allowing the overview of the data to be maintained whilst making individual changes to specific elements. As has been discussed Sagebrush allows the user to customize graphical elements, although users can construct visual structures in a generative way, it does not allow for various forms of interaction to be customized for the components. An aspect of editing is alternative designs for the user to choose from. Tableau uses a ranked order of data association to specify various visual depictions to users when they are choosing visual structures. FLINA is novel as it allows the user to start with the process of drawing. With this process the user can specify axes that join to make visual structures such as parallel co-ordinates, and scatter matrices. FLINA allows the user to specify visual elements of axes such as the colour and the label. History trails facilitate users in understanding what actions they have performed and provide the option of choosing to roll back to previous states. XmdvTool supports history trails through storing the actions.

3.7 Summary

In this chapter various software has been discussed within the domain of information visualization. Various general tools such as word, vector and video editors were discussed to provide an overview of the process of editing with common functionality being defined within the editing process. Software language such as OpenGL, and Processing were explained to demonstrate low and high level technologies that exist for total control implementations of information visualization software. Related conceptual models were defined and explained to demonstrate how information visualization software may adhere to guidelines and structures to be a good implementation. Various toolkits were discussed to understand how high-level abstractions of information visualization concepts such as interaction, animation, and visual structures could be used to quickly generate information visualization software. Finally examples of general and specific information visualization software were provided, with their relation to data, visual structures, interaction and editing defined. We have identified that while tools within Information visualization have many implementations for visual structures and interaction, focus has not been provided for the context of editing finite details of these implementations. This research addresses the context of an editing environment within information visualization to aid the user in understanding and gaining knowledge from their data.

Chapter 4: Earlier Prototyping

4.1 Overview

This chapter discusses the first version of Edivis during the period that Rick Walker implemented it in early 2012. The conceptual model pickup and put will be discussed with relation to the aims and objectives of supporting visualization editing. The software will be discussed in terms of its specification and architecture to provide an overview of work that has already been developed.

4.2 Put and Pickup Model

The original Edivis system was implemented as a proof of concept to support a visualization-editing model. This visualization model uses a pickup and put methodology that utilizes the MVC model [25] as a foundation for the system implementation.

Bostock et al define marks within the Protovis toolkit as a set of primitives that permit many combinatorial properties through non-defined links to graphs [34]. Edivis uses this concept of marks for visual parameters, and allows them to be controlled by the user. The put model controls how visual elements are put onto the display, and is defined in Figure 4.1. The model consists of four separate categories: components, labels and legends, functions and conditions, and locations. Components consist of three classes, dependent and independent elements, and labels. These components allow users to specify what instances of visual attributes to draw such as axes or text. The iterator enumerates over the component, and can optionally apply a function or condition to modify or enhance the component in some way. The location refers to the layout of how the components are put down, this can be manually user defined, data defined, or generically structurally defined.

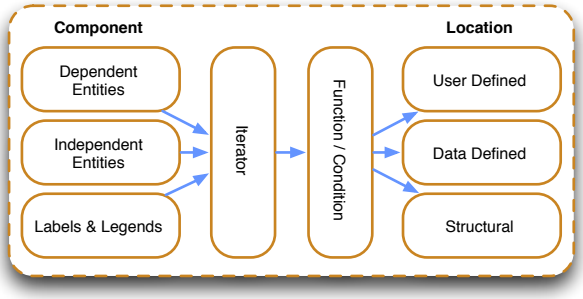

Figure 4-0-1: The put controller: this model defines which components to put, whether they are placed in the same subview or several views as controlled by the iterator, what elements are displayed, and there location.

The pickup model allows values from the display to be selected; this is used with the put model to define editing commands. The pickup model is different from brushing and other selection models as it maintains contextual selections under specific conditions and stores them for later use with the put model. The pickup model is defined in Figure 4.2. Selection contains three classes: user defined, data defined, and structural. User defined allows the user to select components with a brush such as a point, line, or rectangle. Data defined allows a selection based on data parameters such as a range of values. Structural selection allows components to be selected based on certain attributes of their structure, for example specific axes can be selected. Functions and conditions will be combined with the iterator to place constraints on the selected artifacts that have been picked up. This allows the function and iterator to annotate the data so that separate operations can be applied to separate parts. The component is the same as in the put model where there are three classes that define visual attributes.

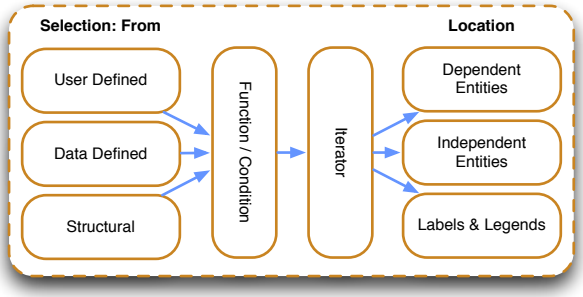

Figure 4-0-2: The pickup controller allows the user to select entities and attributes that can be changed and edited through the put controller. The pickup controller selects dependent, independent and label elements through user, data and structurally defined methods which are processed through the iterator with functions / conditions being applied.

4.3 Software Specification

The goal of Edivis was to provide a tool that illustrated the concepts of the Put and Pickup model. As has been stated the MVC model is used to separate the components of the system. The MVC architecture is useful for Edivis as it readily supports different views of the data model, and the ability to structure the display differently. The adapted MVC model that Edivis uses is presented in Figure 4.3.

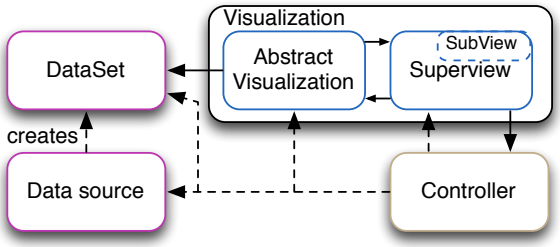

Figure 4-0-3: The model-view-controller pattern implemented within Edivis, super views and sub views allow the possibility for different visual representations and configurations of the dataset.

The software was built with Processing as it supported high and low level entities of the graphical pipeline, large screens, and PDF output. The first version of Edivis used a canvas to layout visualizations. This canvas supported a Zoom able User Interface (ZUI) that allows users to zoom in and out of the display. Users can also navigate around the canvas by dragging the mouse whilst holding the right button. The layout of visualizations is handled automatically through a force directed graph, although the specific position of plots can be manually defined. Data is handled within Edivis automatically; the data is manually cleaned and parsed into the software. Because the data is manually parsed into Edivis, there is no generic tool that supports various data types and only one mapping is presented in the form of floating point data. Another outcome of having the data parsed is that once Edivis loads, the user is automatically presented with a parallel plot containing the entirety of the data.

Interaction is presented in Edivis through the use of the put and pickup controller. Functions and conditions in Edivis relate to different layout types, and separate configurations. Users must specify components, and functions and conditions through the menu system. The menu system is implemented in the ControlP5 library (ref here) for Processing. The menu contains five categories, selection type, selection mode, layout modes, group modes, and axis ordering. Categories within the menu can be related to the put and pickup model. Selection Type, Selection Mode, and Group Modes all relate to functions and conditions for the pickup model. Layout Modes relate to components in the put model. Axis Ordering relates to functions and conditions in the put model.

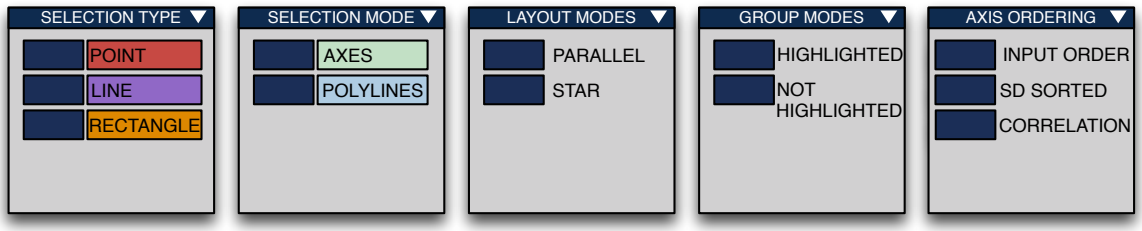

Figure 4-0-4: The menu system for Edivis, implemented in the controlP5 library. Selection Type Selection Mode, and Group Modes all relate to functions and conditions within the Pickup model. Layout Modes and Axis Ordering relate to components, and functions and conditions for the Put model.

Through supporting the put and pickup model Edivis allows the exploration of visualizations using multiple views. Users can pickup axes and polylines from plots using brush tools such as point, line, and area. Elements that have been picked up are placed into a put controller. The put controller will iterate over the selection of elements dependent on functions and conditions that have been specified in the menu. This permits Edivis to create separate views based on the same data using the put controller. Orderings of axis can also be specified by three separate functions, the input order (no change), the standard deviation, and by pair-wise Pearson correlation coefficient using a greedy algorithm.

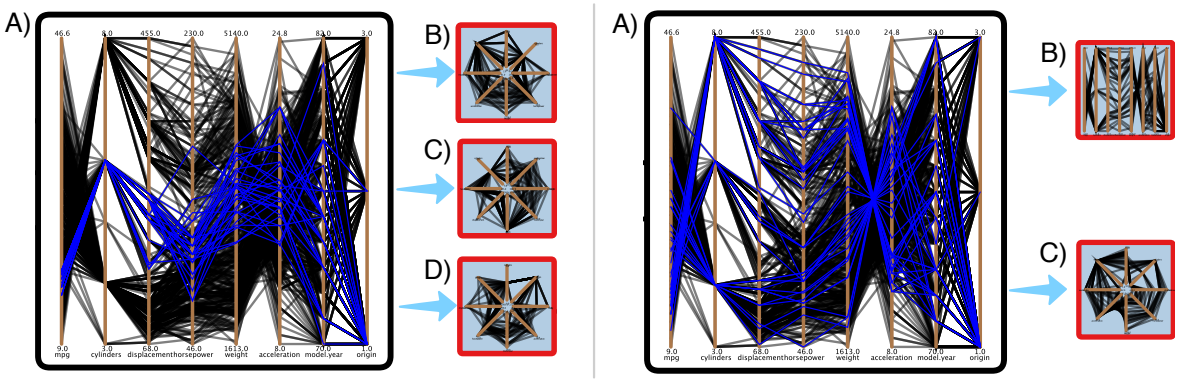

Figure 4-0-5: The left illustrates the concept of picking up polylines and re-ordering this selection. A is the plot that data is being picked up from, Plots B, C, and D all represent separate axis configurations of the selected data. B is the standard input order arrangement, C is standard deviation rearrangement, and D is a correlation rearrangement. The right illustrates the concept of creating separate views from the same data, A is the plot that data is selected from, B is a parallel plot of this data, and C is a star plot of this data.

4.4 System Architecture

The software implementation was prototyped within the Processing IDE. This means that the language is an abstraction on traditional java, where some features are handled automatically for the users convenience. The basic architecture of the program consists of four classes, a Main class, a PCP class, a data model class, and an Axes class. The main class handles the drawing and coordination of the program and holds instances of the PCP class within a data structure called a list. The main class also handles input from the keyboard and mouse, calling certain methods based on specific inputs from the user. The PCP class is responsible for the drawing and interaction of the parallel coordinate plots and star plots. The PCP class holds a reference to its parent if one exists so it can traverse a tree of parents to highlight the relationships between plots. The PCP class contains an array of Axes and a single Data Model. The Axes class is a data structure that details the position, and label of an individual axis within a PCP. The Data Model class specifies a list of indices that the plot contains in relation to the dataset. Storing the indices of the data as opposed to the data itself makes the program more efficient, when the data value is needed a get method returns the value based on given indices. The data model supports methods to compute statistics such as the mean, standard deviation, and pearson correlation for data contained within it. Various constructors exist within the data model itself so that inverted

models can be generated from external function calls. Enumarables exist to store states of user options through the program. There are five enumarable classes that correspond to the menu options, Axis Ordering, Highlight Type, Layout Mode, Selection Primitive, and Selection Type. An enumarable for Interaction Mode exists however this is not used within the system.

The Unified Modelling Language (UML) has been used to generate an overview of the system architecture. The systems classes Main, PCP, PCP Model, Axes, and all the Enumarable classes are represented as well as a overview of the system. These UML diagrams are in the appendix in section ?.

4.5 Summary

An overview has been provided of current work implemented for the concept of visualization editing. The Put and Pickup model has been introduced to discuss the concepts the model provides in relation to information visualization editing. The controller classes for both the put and pickup model were presented in relation to the model view controller architecture within the implementation of Edivis. Edivis was discussed in terms of its ability to support the put and pickup model through exploratory visualization. The specifications of the software such as multiple views, and the various forms of interactivity were also discussed. Finally an overview of the implementation in relation to the system architecture was provided.

Chapter 5: Design

5.1 Overview

This chapter will discuss the design of the extension to Edivis. First the five-design sheet approach shall be discussed. Next three different designs shall be summarized in terms of advantages and disadvantages. Finally a final design shall be discussed.

5.2 Five Design Sheet Approach

The Five Design Sheet (FDS) has been developed as an iterative approach to design [46]. The process includes five stages that generate five outcomes. The five stages consist of creating one brainstorm sheet, three design sheets, and one realization sheet. Creating the brainstorm sheet is freethinking, where any ideas are drawn on the sheet. Ideas from the brainstorm are refined to what is most useful and three Design sheets are created based on some of these concepts. Each design sheet contains a specific structure consisting of five components, Layout, meta-information, Focus, Operations, and Discussion. The design sheets provide alternative ideas for a software implementation, and are good discussion points for clients as they can be refined based on their needs. The realization sheet is the final stage and contains all the categories of the previous design sheets, with the exception of Discussion. Discussion is replaced with specific Details for the software such as algorithms, time constraints, reliabilities, and materials; this allows the software to be taken into the implementation stage. The FDS approach has been chosen, as various designs to the extension need to be considered and refined before a final option is chosen.

The first stage brainstorming was carried out where unrefined ideas were drawn on paper. The most important of these ideas that were carried forward into the designs were that of different types of plots, parallel coordinated views, and searching algorithms for data within every plot synchronously. An image of the brainstorm is provided in the Appendix.

5.3 First Design

The first design proposed an implementation that included a tabular based menu where users could select different tabs for different task related options within the system. The design supports three types of visualization, parallel plots, scatterplots, and bar graphs. These plots can all be generated from one another, based on the put and pickup model. The design also supports an action list so that users can rollback to a specific state. Layout options such as the ZUI and force directed graph remain, and the canvas is the main element on the screen for the user to interact with. Finally the design supports a data tool that helps the user to clean, parse and map their data into the software.

The designs advantages are that the options for the system are contextually divided into a tabular menu making the system easier to use. The support of a graphical data tool is also considered an advantage, as this is one of the most challenging aspects of any information visualization system. Disadvantages of the design are that the system does not help guide the user through the process of visualization through ‘pop out’ features, or helping hints and tips. The creation of a graphical data utility that supports the software is also considered a disadvantage. This is because it would be a challenge to support the increasing specification of types and formats of data within one tool, and map them to specific objects within the software. The first design sheet is illustrated in Figure 5.1.

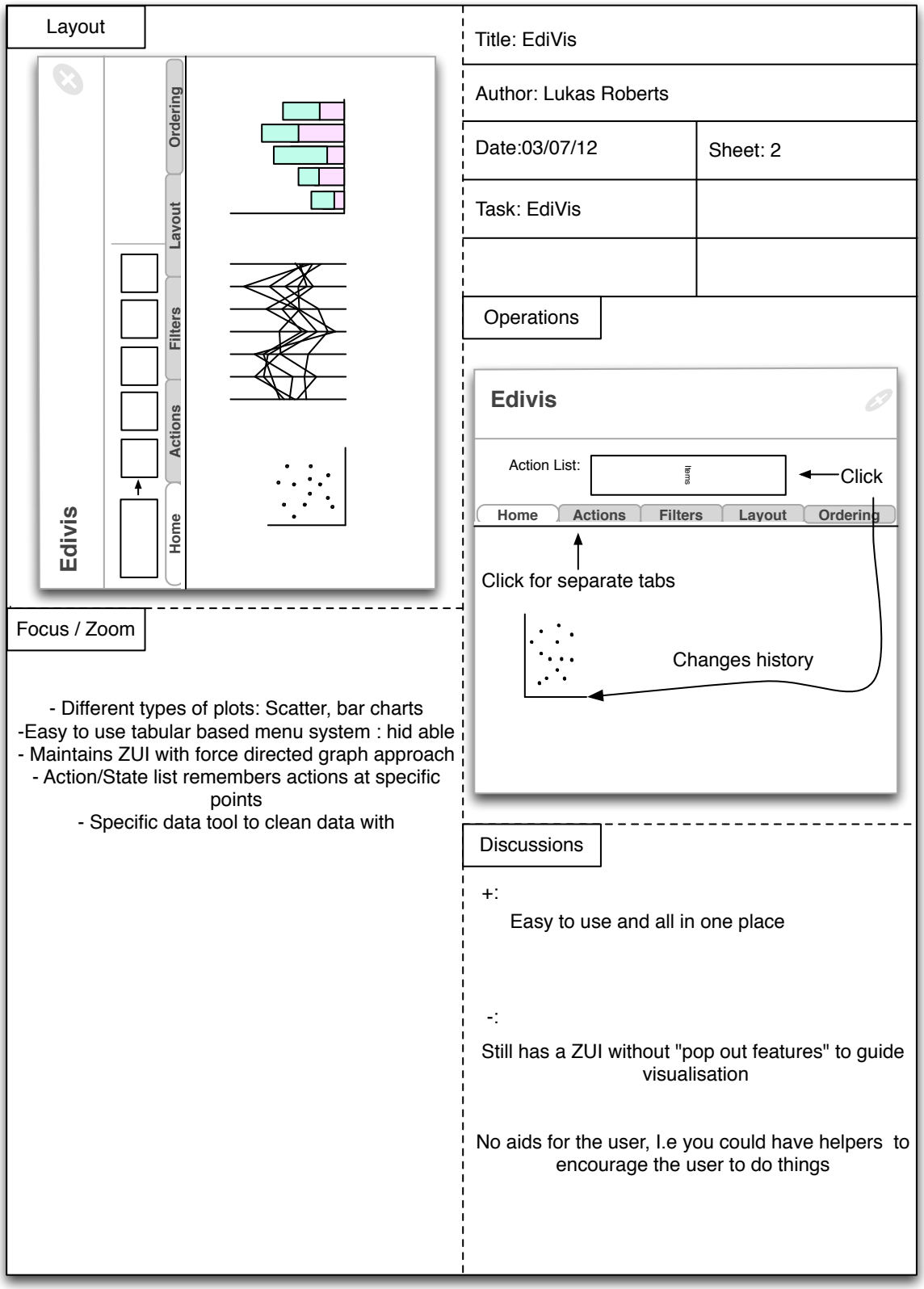

Figure 0-1: Design sheet one is illustrated through layout, meta-information, focus, operations, and discussions

5.4 Second Design

The second design proposes an implementation that uses a toolkit and multiple windowing system to separate the tasks available to the user. The toolkit contains tools that are relational to specific operations within the Put and Pickup model. These tools can be used on the canvas to perform actions, for example a mouse tool selects and move objects and plots, whereas an axes tool can visually put an axis onto the canvas. The menu system is located to the right of the visualization and is static in both size and position.

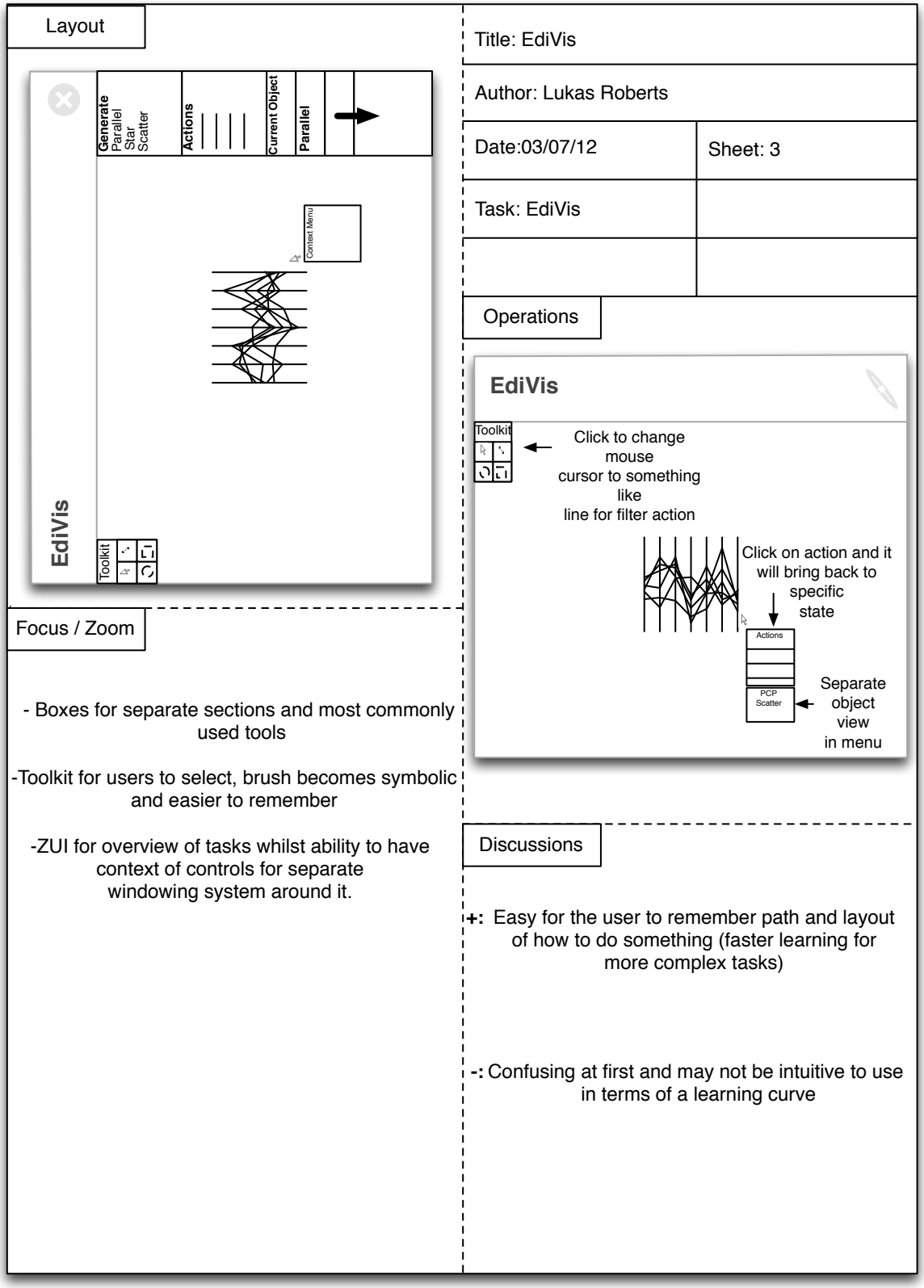

Figure 0-2

The menu system is based on the active object in the canvas. This means that if a parallel plot is selected, then the relevant options for this plot will be displayed in the menus. The system also supports context specific menus whilst the user is exploring visualizations on the canvas. Options for plots and other objects can be selected through right clicking them, which will bring up a list of options relational to the object in a menu on the canvas. There are also general options presented when right clicking on the canvas such as inserting structures such as parallel coordinates or scatterplots. Action lists are supported so that users can rollback to a previous state, this can be specific to an object providing it has been selected on the canvas.

Advantages for this design is that the tools represent a symbolic approach to learning the functions of the system. This process helps a user to learn a system quickly. Symbolic tools are also beneficial as other software packages may contain tools with the same symbol that provide the same functions. This means the user automatically knows how to use functions within the system from previous experience. This also provides a disadvantage in that it can cause confusion to the user if the same symbol does not provide the same function that it does within other software packages. Another disadvantage is that the software may not be intuitive to use, as there is not a literal definition of functions for the Put and Pickup model.

5.5 Third Design

Much like the first design the third design uses a tabular based menu system to separate its menu components. The menu is located in a collapsible ribbon dock. The menu provides options to modify the canvas and objects within it. The menu has five tabs, Viz Layout, ZUI Options, Viz Selection, Viz Extraction, and Draw. Viz Layout defines options for the layout of the canvas and plots. ZUI Options controls aspects of the ZUI. Viz Selection handles options for visualizations to place on the canvas. Viz Extraction lets the user control what components to pickup from a plot. Finally Draw contains options that let users choose tools and components to draw to the canvas. The canvas is still included in the design including the ZUI and force directed graph with the ability to put and pickup components. The design includes multiple forms of visualization such as parallel plots, scatterplots, star plots, bar graphs, and user defined structures. The design supports searching automatically for values that have been picked up in every other plot that exists on the canvas. This aids exploratory visualization as users can see likewise values that occur in separate plots. The concept of drawing is supported where users can select components and put them on the canvas. For example users can put down axes and link them together to create their own visualization structures. Drawing also allows users to add labels and notes that explain or aid the story of visualization. The design supports the concept of editing through giving the user the ability to change the plot colour. This can help users visually identify the meaning of certain plots and change the appearance to suit their needs. The design accommodates for exporting the visualization through saving the entire canvas or individual plots. This can be done through XML or vector and PNG output.

Advantages of this design are that it provides a generic approach to creating and editing visualizations utilizing the Put and Pickup model. The design also gives the user the freedom to explore data through visually defined structures and intelligent search algorithms. Editing visualizations is an advantage as users can reconfigure components of plots such as parallel coordinates and save them as there own for later use. Disadvantages of the system are that it may not allow separate types of data, as there is no tool or function to import this with. This means that the system may only be able to use floating-point data from Comma Separated Value (CSV) files. Because the design only supports floating-

point data, the ability to generate domain specific visualizations or complex visualizations such as trees, or networks is not possible. This also means that spatial and temporal data cannot be visualized. The third design sheet is illustrates in Figure 4.3.

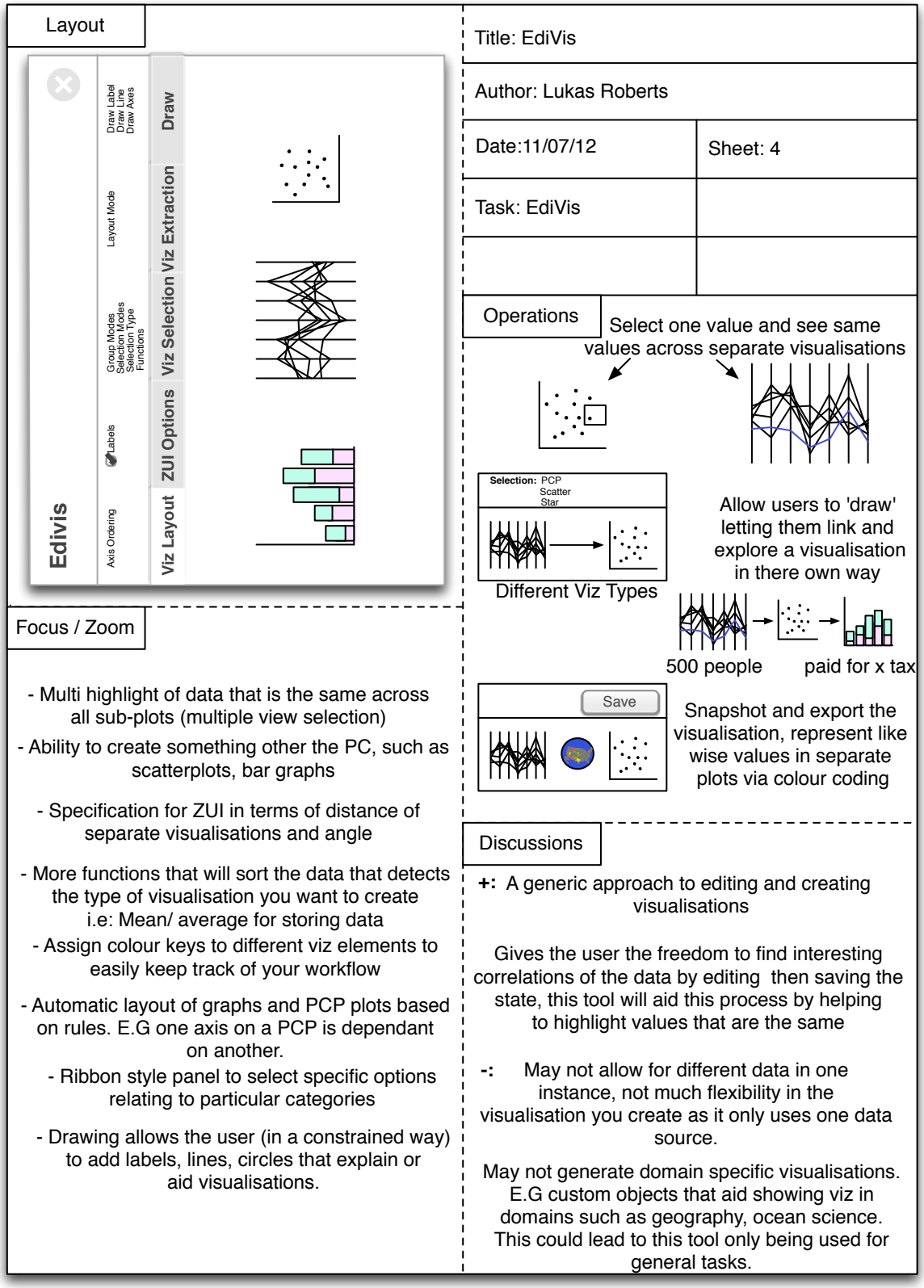

Figure 0-3

5.6 Realization Sheet

The previous three designs from sections 5.3, 5.4,and 5.5 were all considered in relation to how they utilized the Put and Pickup model and if they exploited features of visualization editing. It was decided upon to use the design outlines in section 5.5. This design was chosen as it fully utilized the Put and Pickup model for information visualization editing. It is the simplest design to implement as a codebase already exists with some of the features.

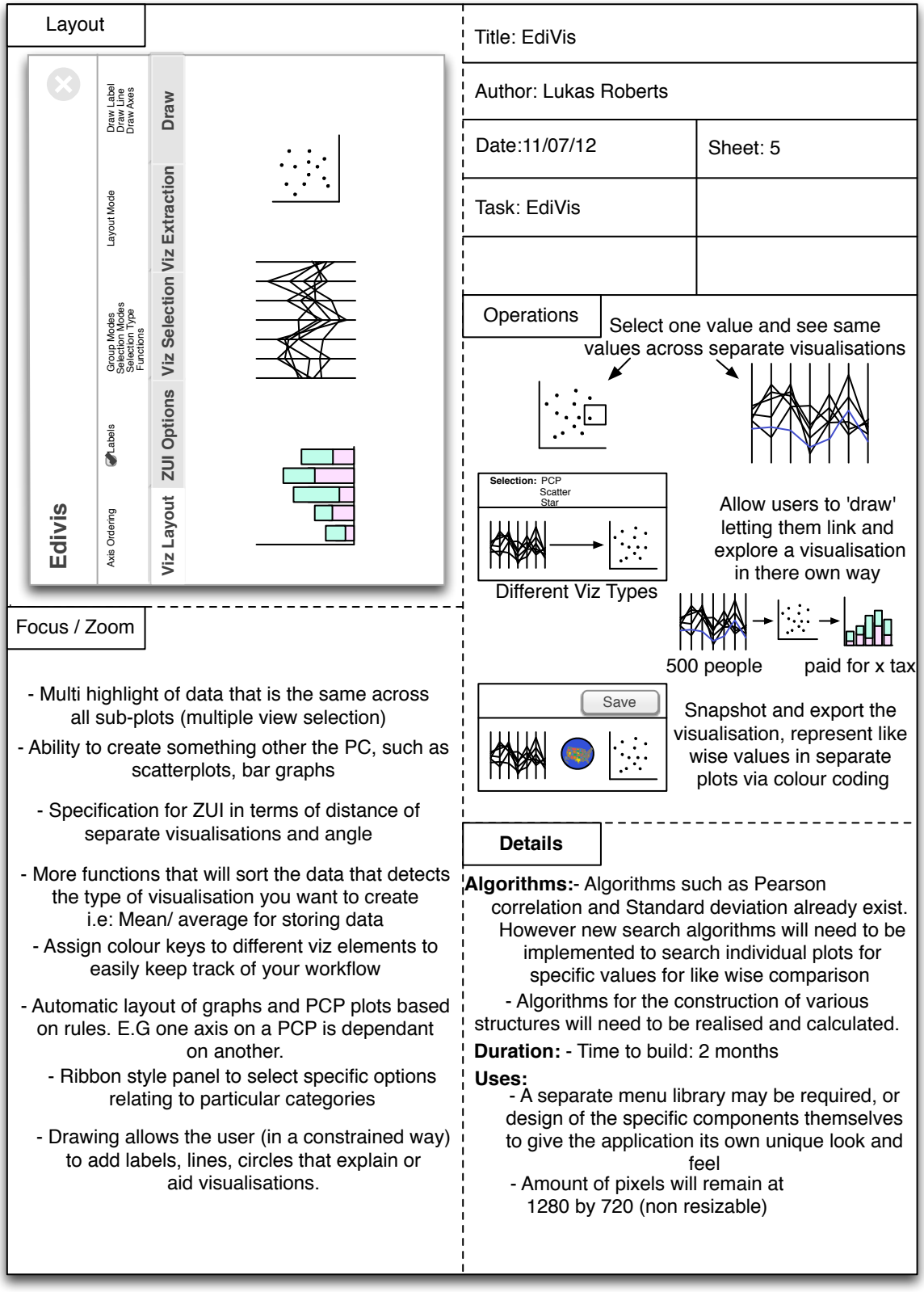

Figure 0-4:

This design is somewhat similar to the sheet presented in section 5.3; the main difference is that it does not include a tool for importing data. Building a data tool would be a challenge for reasons stated in 5.3; for simplicity of implementation this was decided against.

For the implementation of this tool it has been estimated that two months will be required. Novel algorithms will need to be created for structuring data for plot types such as scatterplots, as well as searching plots for similar values. The architecture of the existing implementation will need to be redefined to support multiple forms of visualization and a new graphical interface. This refinement in architecture may lead to the implementation relying on new graphical libraries. The implementation size will remain static at a screen resolution of 1280 by 720 pixels.

5.7 Summary

Various designs have been created using the five-design sheet approach. These designs have been evaluated in terms of applicability to the Put and Pickup model and visualization editing. Technical features of the designs were discussed in terms of advantages, disadvantages and simplicity of implementation. A final design was chosen based on design sheet 3 as it best matched the requirements and objectives of the project. This provided a realization sheet with details of the implementation in terms of build time, algorithm design, and dependencies.

Chapter 6: Implementation

6.1 Overview

This chapter shall discuss the implementation of the extension to Edivis based on the realization sheet design. First the technologies that have been used to implement the software will be discussed in relation to what they do, and why they were chosen. Control parameters and components that change the systems state will be discussed in terms of the architecture that is used. The implementation for creating visual extractions shall be explained, with visual structures such as scatterplots, scatter matrices, and stacked bar graphs taking focus. Implementation features of visualization editing shall be explained. Finally tools that aid in exploration of the dataset to find values of interest shall be explained in terms of development.

6.2 Implementation Technologies

6.2.1 Rationale

To implement the software the Java programming language has been used. Java has been chosen as it provides an Object Orientated Programming (OOP) environment that the concepts of the Put and Pickup model can be implemented within. Java also allows the inclusion of different graphical frameworks such as Swing, and AWT. Processing can be included as a library within a Java application, allowing a mixture of Java code that includes the high level abstractions of Processing. In order to develop this application in Java, the Netbeans IDE will be utilized to bring together all the libraries and code into one place. To increase performance with the rendering of points, lines, and other graphical primitives, OpenGL shall be used so that all drawing occurs on the Graphical Processing Unit (GPU). Specific Processing functions will still be used for the canvas implementation as it provides an easy way to specify control over the graphical pipeline, whilst using OpenGL as the rendering engine.

6.2.2 Porting The Prototype

In order to use Java it was necessary to move the current applications context from the Processing environment into the Netbeans environment. The process of doing this involved creating a project environment within Netbeans that supported OpenGL, and contextually separating the classes from the one Java file that Processing exports into several classes. In performing this task there were some dependencies that existed between classes. For example the PCP class relied on a method from the Run class, this method could be accessed because all the classes are located within one file. By separating these classes into separate files, local instances of the Run class would need to be held in the PCP class to access methods. The overall architecture representing the program after it had been ported into Netbeans is illustrated in Figure 6.1. The program is run from a Main class that holds multiple parallel coordinates plots within a list. Each PCP holds an array of Axes and one instance of a PCP model. All these classes can interact with the enumerable states that are stored within a separate package.

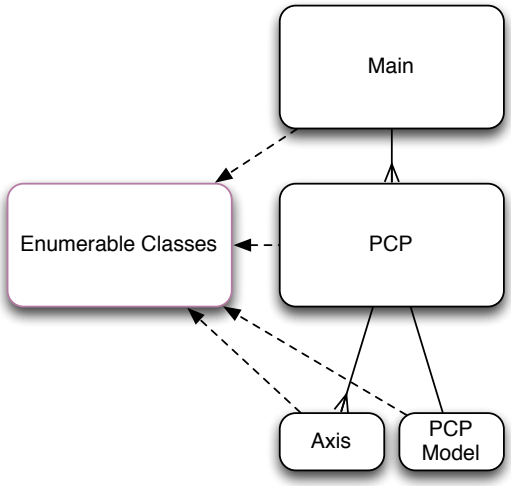

Figure 0-1: A diagram illustrating the system architecture that originally existed when the prototype was first ported to Netbeans. The main class coordinates the logic and listens for events from input devices. The main class holds several PCP classes within a list. The PCP holds an array of axes, and a single PCP model.

6.3 Specifying Options and Parameters

6.3.1 Graphical User Interface Architecture

For the canvas to be extended it was necessary to develop a new user interface to specify the options and parameters relevant to the Put and Pickup model, and visualization editing. The realization sheet from section 5.6 describes a ribbon style interface that is collapsible. The current menu is implemented within ControlP5 as separate accordions holding a list of buttons (see figure 4.4). The decision was made to drop ControlP5 as to implement the required interface would be tedious utilizing the library. Instead the Swing framework would be used. Swing provides the advantage that it offers a wide variety of standard components that can be customized to requirements. The major concern of using Swing was mixing lightweight and heavyweight components as this may cause issues, these problems would be dealt with when they were encountered.

To utilize Swing with processing a specific architecture would need to be implemented that took commands from Swing components and sent them to classes that used Processing. Whilst the initial prototype reference stated it used an MVC model this was not reflected in the code. There was no apparent segregation between controller code, and view code. This is most likely due to the nature of the Processing development environment where mouse and keyboard listeners exist within the class extending PApplet. For this reason, and due to the nature of Swing, classes that used Processing were split up into separate packages and classes. To support commands between Swing and Processing, a reference to the class that extends the PApplet object needs to exist within the classes that implement Swing components and vice versa. This two-way relationship means that both Processing and Swing have communication with each other, allowing both to be updated dependent on the source of user input.

The swing implementation contains four components, a dock, a menu, a data table, and a statistics panel, both the table and statistics panel require their own data model encompassing a total of six classes. The menu holds instances of both the object that extends PApplet (Run) and the dock; it is essentially the central class that holds the implementation together. The dock holds both the data table and statistics panel objects, and can reference the Run variable from its Menu object. Processing holds an instance of a menu object, this menu object can reference the dock object meaning processing can affect

both the menu and dock with one simple object. The relationships between these classes are illustrated in Figure 6.2.

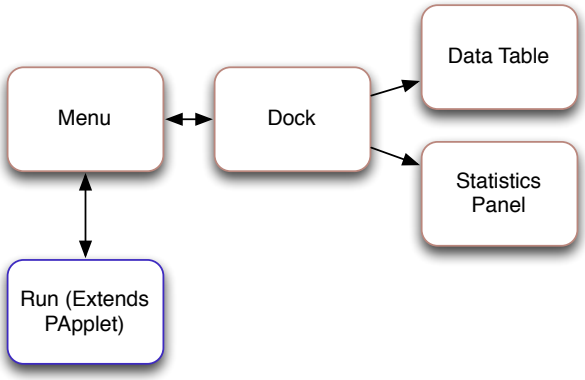

Figure 0-2: This illustrates the relationships between the GUI classes and Processing. Items colored in orange represent swing components, whereas the item colored in purple represents the Processing object.

6.3.2 The Put Model Interface

Options and components for the Put model are specified as a set of Swing J Check boxes located in the Dock class, and a set of J Radio Button Menu Items located in the Menu class. It was decided that users could put multiple plots onto the canvas at the same time, to achieve this a list of selected layout mode enumerable objects were held within the Run class. So specific plots could be added based on selections from multiple checkboxes, an indexing system was created that put specific indices into positions in an array based on the checkbox selected. This array is passed onto the Run class and iterated through, if the relevant number is found then a specific layout is added to the list of enumerable objects, the pseudo code for this is presented in Listing 6.1. The interaction for some of the components in the pickup, and explore tabs work on exactly the same indexing system described in Listing 6.1. The difference is that they run to separate methods within Run, and affect separate instances of different enumerable objects.

The next section of the put interface is located within the J Menu Bar component. The menu has a section for put, that contains three options; input order, standard deviation, and correlation. This interacts with the Run class with the same logic as Listing 6.1. The only difference in logic is that as opposed to passing down an array of options, only one value is passed based on what radio button is selected. Both the menu and dock interface for the Put model can be seen in Figure 6.3.

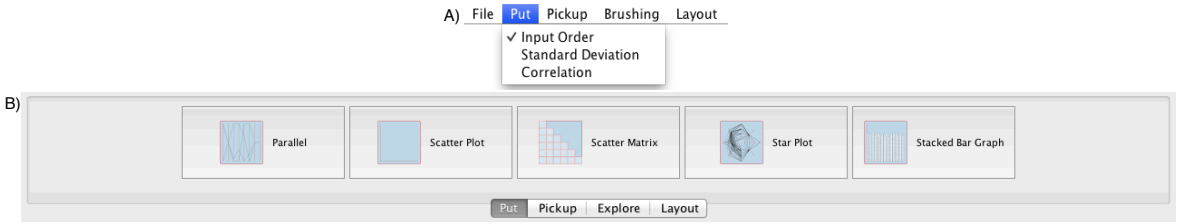

Figure 0-3: An illustration of the components for the Put model. A) Represents the specific options of axis ordering based on what the user is putting onto the canvas. B) The section on the dock that presents the user with the available visual structures they can put onto the canvas

```
if (parallelButton is selected or scatterButton is selected or smButton is selected or
starButton is selected or stackedBarButton is selected) {

            create array of integers as large as amount of buttons(5);

            if (specific button is selected) {
                set  array of integers at position relational to button to chosen index;
            } else {
                set  array of integers at position to zero;
            }

            call Run.layoutModes(pass integer array);
        }

  Method Run.layoutModes(input of integer array) {
        clear list of enumerable objects;
        for every item in integer array {
            LOOP
            if (item in integer array is equal to chosen index) {
                add relevant layout to list of enumerable objects;
            }
            ENDLOOP
        }
    }
```

Listing 6-1: This represents Pseudo code that was written to pass down control options to the Run class. The code in blue represents the linking method that exists between classes.

6.3.3 The Pickup Model Interface

Components that support the Pickup model are very simple and have not changed since the prototype. The only difference is that they have been moved into the swing framework as a set of radio buttons to specify options for data values (highlighted and non-highlighted) or components of a plot (polylines or an axis) to pickup. An illustration of the user interface for Pickup is presented in Figure 6.4.

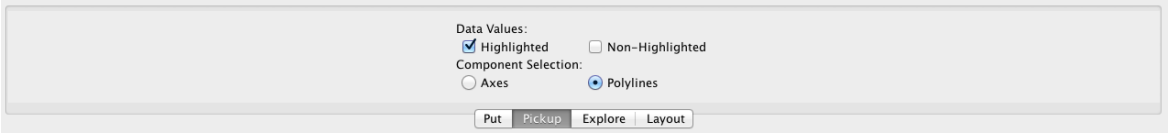

Figure 0-4: The pickup model interface: Users can select whether they wish to pick up highlighted or non-highlighted values from a plot, and what component of the plot they wish to select.

6.3.4 Explore Components

The explore interface simply contains a set of J Radio Buttons to select the tool to use; this has not been changed since the prototype with the exception of Smart Extraction Mode. Smart Extraction Mode is responsible for the creation of both the statistics panel and the data table. The data table is created automatically when the smart extraction J Check Box is selected, and deleted when it is unselected. The Statistics panel is created in the case that the J Button is pressed if there is not an instance of the panel already in existence. The specific implementation of the Smart extraction mode will be discussed in Section 6-6. The explore interface is illustrated in Figure 6-5.

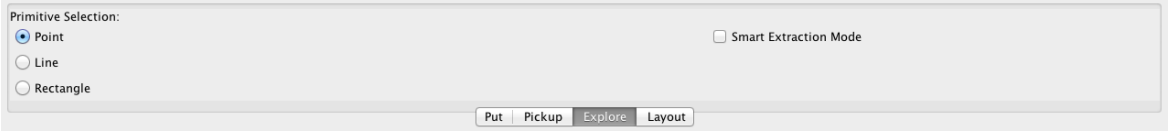

Figure 0-5: The Explore interface that consists of three J Radio Buttons and a J Checkbox to enter Smart Extraction Mode.

6.3.5 Layout Components

The layout interface was implemented using several J Radio Buttons, J Check Boxes, and J Buttons, and a single J Formatted Text Field, and J Slider. The radio buttons interact with Run using the same logic as Listing 6-1, except for the formatted text field. The formatted text field parses the value from the field, checks if it is not zero, and calls the method in Run that sets the size for all plots. The method in run checks that the value it has received lies between the maximum and minimum size for a plot, if the value is between the boundaries then all plots that are on the canvas are resized.

Dock:
```
if (source is equal to textbox) {
                if (identical Size checkbox is selected) {
                    set size to text box value;
                    if (size is not equal to 0) {
                        Call Run.setArbitarySize(pass in size);
                    }
                }
            }
```

Run:
```
Function setArbitaryPlotSize(input of size) {
set arbitaryPlotSize to size;
        if (arbitaryPlotSize is greater then 50 && arbitaryPlotSize is less then 3500) {
            set size to plots.size();
            if (size is greater then 0) {
                for every plot on the canvas{
                LOOP
                    Call resize method of plot(pass in arbitaryPlotSize);
                ENDLOOP
                }
                call layoutPlots();
                call updatePhysics();
            }
        }
}
```

Listing 6-2: Pseudo code that handles user input from the text box and passes the value to the Run class.

All interface components for editing the plot only accept input if the edit appearance checkbox has been selected. The colour buttons are customized J Buttons for which the backgrounds are set from within Run when a user clicks on a new plot. If the user clicks on the colour buttons a J Colour Chooser component is created that will only disappear once a color has been selected or the component has been disposed. The slider sends numerical values to the Run class between zero and ten. Finally the display borders checkbox sends a Boolean value to the Run class to change the state of whether borders should be displayed. An illustration of the Layout interface is presented in Figure 6-6.

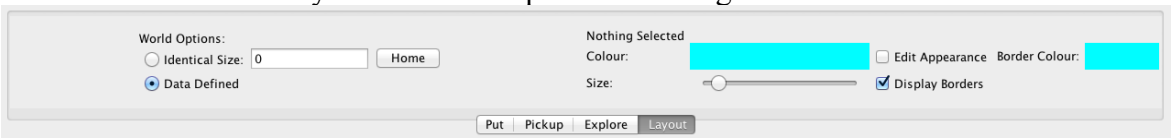

Figure 0-6: The layout interface that includes two specific categories of selection. The left side is relational to the drawing world where users can specify the size of plots. The right is relational to the individual plot, where users can select the colour, border colour, whether to display the borders, and the size.

6.4 Creating Visual Extractions

One of the goals of the realized design was to have multiple forms of visualizations as the prototype only supported Parallel coordinate plots, and Radial plots. The decision was made to incorporate scatter plots, scatter matrices, and stacked bar graphs. This section shall explain the implementation of the individual plots, as well as the architecture that is used to support them.

6.4.1 Plot Architecture

The code from the prototype supported both parallel coordinates and radial plots that were implemented within one PCP class. This was because a radial plot is the same as a parallel coordinate plot with the exception that axes are plotted at an angle between 0 and 2π. Because only one class existed that supported the structure of plots it was necessary to create a generic class that had the capacity to hold multiple types of visualization. To create this class it was a necessary task to identify similarities that all visualizations share. General similarities that visualizations share can be defined as location, size, data, colour, and references to other visualizations. Differences that plots share can be defined as the visual appearance and structure of the plot. Defining the similarities and differences of visualizations generated the creation of a Plot class that holds variables for the similarities of visualizations. This Plot class is similar to the concept of Marks [34] in that Plots know nothing about how visual structures are laid out but store common variables between visual structures. Using this concept provides freedom to implement multiple visualizations and separate the context between visualization and common methods that affect them appropriately. Similar variables such as location, size, and colour are held as floats. Using the same data model as the PCP Model class allows the Plot class to store data; the name of the class has been refined to Data Model. Parent references are stored as a Plot variable within the Plot class itself. The differences that have been stated between visualizations are handled within there own respective classes that specify the structure of the individual plot. This means that for every varying type of layout for visualization, there is a class that handles the structure and organization for that layout. All of these classes that handle the structure of visualizations have to be held within the Plot class as variables. An illustration of the plot class structure and the variables it holds is presented in Figure 6-7

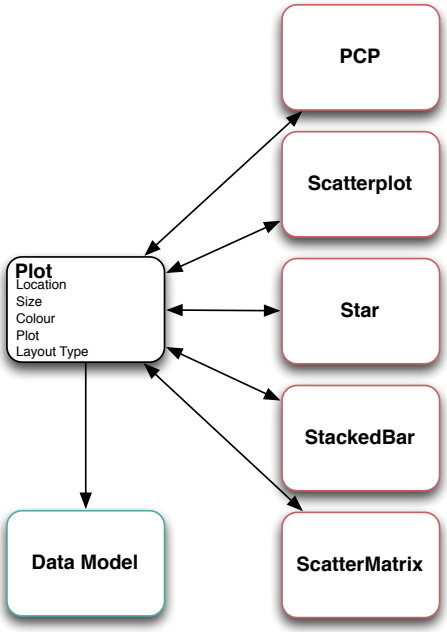

Figure 0-7: The plot class architecture contains local variables that are specific to location, size, colour, parent referencing, and layout type. Visualization objects are stored within the plot, where one is created based on the state of Layout Type. These plots have a two-way relationship with the plot class so they can access variables specific to location, size, and colour. The data model is also stored so visualization types and the plot class can both access data specific to the plot.

To choose between these layouts the enumerable class Layout Type is passed into the constructor of the Plot class and stored as a local variable. The relevant layout is chosen based on the state of Layout Type, where either a PCP, Star, Scatter, Scatter Matrix, or Stacked plot is created. Its important to note that all visualization classes exist as variables within the plot, but only one is chosen based on the Layout Type state. The logic for this process is illustrated in Listing 6-3.

```
set plotData as new DataModel(based on plot data passed through constructor);
     call plotData.sortColsBy(ordering);//sorts ordering of the columns
     switch (based on layout type enumarable) {
         case PCP:
             set parallelPlot as new PCP object;
             call parallelPlot.createAxes(input layout, input parent);//creates the axis
             break;
         case STARPLOT:
             set starPlot as new StarPlot object;
             call starPlot.createStarAxes(input parent);//same as for parallel
             break;
         case SCATTERPLOT:
             set scatterPlot as new ScatterPlot object;
             break;
         case STACKEDBARGRAPH:
             set stackedBarGraph as new StackedBarGraph object;
             break;
     }
```

Listing 6-3: Pseudo code that defines the logic for declaring the type of visual structure that exists within a plot. A plot can only have one type of visual structure that exists within it, so only one Layout Type is passed to the constructor. The rest of the visual structures that are not chosen will remain as null. The layout type is saved as a variable to define the visual structure of the plot.

Because variables exist within the Plot class that correspond to location and size, public methods exist that can change these variables. The Plot class provides draw, resize, move, draw to parent, and define colour methods for all visual elements that exist within the plot. The type of visual structure that the method affects is defined as the same logic presented in Listing 3. The context of methods will differ for separate visual types, as the function of resizing a scatter matrix is different to that of resizing a scatterplot. Resize, move, colour, and axis specific methods shall all be discussed within Section 6.5.

The draw and draw transparent methods call the corresponding methods that exist within the visual structure specified by the local Layout Type variable. Draw to parent checks if a parent plot exists, if one does then a line is drawn from the location of the plot to the location of the parent plot. Recursive methods exist that use the parent plot reference, these methods are specific to forms of parallel coordinated views or smart extraction mode and will be discussed in the relevant sections.

6.4.2 Handling Extractions

The constructor for a Plot requires parameters for location, size, colour, layout and data. To implement new visual extractions a refinement of the existing method that creates parallel coordinate plots was necessary.

6.4.2.1 Calculating Intersections

Plot data is selected by using a point, line, or rectangle from a polyline or axes. Within plots that support extractions there are methods to support the selection of data based on the component specified and the tool used.

Calculating whether a point lies on a polyline or an axis is handled by returning a vector V from a method named get distance. The method get distance works by first calculating the difference between two points (P and Q) as follows

$$d_x = P_x - Q_x$$
$$d_y = P_y - Q_y$$

Next the distance is calculated by the equation $d = \sqrt{{d_x}^2 + {d_y}^2}$

Next the cosine value is calculated as $c = dx/d$, and the sine value as $s = dy/d$. Where the point lies on the line is calculated as

$$m = (-P_x + x) * c + (-P_y + y) * s$$

Next we check that m does not lie outside 0 or the distance of the line, if it does then we set the vectors x and y values to either the first point or second. If the point lies on the line the vectors x and y values are set as

$$V_x = P_x + (m * c)$$
$$V_y = P_y + (m * s)$$

Finally the difference between a point R to the line is calculated by working out new delta values based on the difference between the point and vector position.

$$dx = R_x - V_x$$
$$dy = R_y - V_y$$

Where we set the vectors z value as

$$V_z = \sqrt{dx^2 + dy^2}$$

Once the vectors z value has been returned an if statement checks the value is below one, in which case the relevant row is selected. This test is done for every row of data within a plot through an iterated loop.

To calculate whether a line or rectangle intersects polylines, points, or an axis a library called Toxiclibs is utilized. Toxiclibs includes a variety of methods that allow basic geometric intersections to be calculated. To calculate whether a line intersects a component the logic is presented in Listing 4. In the pseudo code Line2D is a Toxiclibs data type that contains a method intersectLine to test whether a line intersects. Values p1 and p2 refer to mapped pixels that correspond to data values in the plot. The variable lineBeingDragged is the users line that is being dragged across the screen. The get type method is used to find out the type of line that is being created, if the type is the same as the Intersecting type then the row is selected.

```
if ((new Line2D(p1, p2).intersectLine(lineBeingDragged).getType() == Line2D.LineIntersection.Type.INTERSECTING)) {
                    select the Row(at index in loop);
}
```

Listing 6-4: Code that corresponds to utilizing the Toxiclibs library to calculate line intersections

The rectangle logic uses a Rect object from Toxiclibs as well as creating a Line2D object from screen pixels of data points currently being iterated over. The testing method checks whether the rectangle contains the screen pixels and also tests whether the rectangle intersects the given ray of the line. If either case is true a row is selected.

These tests are performed no matter what the component, whether it be an axis, polyline, or point. To calculate the index of the specific axis that is being picked up is performed in a separate test discussed in section.

6.4.2.2 Iterating Over Options

The data is stored in a local Plot variable in the Run class that is set whenever the user moves the mouse over a plot. When the user makes a selection on the plot the intersection testing as outlined in Section 6.4.2.1 is performed that selects rows in the Data Model class. When the mouse is released and some form of selection exists in the Data Model the method that creates an extraction is called. The method iterates through lists that contain user specified Layout Modes, Axis Orderings, and Highlight Types; there must be at least one enumerable object in each list for the method to proceed. Each enumerable object in the list is dealt with individually in relation to creating a plot. Within the method an if statement checks that the selected plot is not a scatter matrix, if it is the method will not proceed as extractions cannot be made from a scatter matrix. Scatterplots have there own specific format of creating extractions and will be discussed in there relevant section. If the Layout Type is not a scatter plot or scatter matrix then the plot is made through its default constructor.

6.4.3 Scatter Plots

To implement the Scatterplot class it was first necessary to organize the data, so that if scatterplots were generated from a plot with more then two dimensions, data would be organized as a linear correspondence between axis n and axis $n + 1$. The logic for this is presented in Listing 5.

```
set size to column count in selected plot;
      for (int i = 0; i < size - 1; i++) {
            create new column array list based on selected columns in selected plot;
            create new row array list based on selected rows in selected plot;
            create new DataModel(row array list, column array list, column names);
            setDataModels Columns(i, i + 1);
            create new Plot(location,size, colour, layout, DataModel)
      }
```

Listing 6-5: This pseudo code is the logic for creating scatterplots from adjacent axes. The size of columns within the selected plot is stored as the variable size. The count of size – 1 is then iterated in a for loop. In this for loop a new data model is created from the selected columns and rows. The data models columns are then set to the current index in the loop and the current index + 1. Finally a new Plot is created based on the data model.

Once the data model has been organized, it is passed through to a constructor in the Plot class. This constructor then creates a Scatterplot based on the data model. The scatterplot has two steps to creating its visual structure. The first involves creating the axes for the scatterplot. The second involves setting the axes data boundaries for the maximum and minimum data values in the data model. Once the axes are setup draw methods specify the points that are rendered within the scatterplot. The draw method renders points by using the processing map function to translate data points into vertices. The data point is mapped along either the x or y-axis between the data boundaries of an axis, and the start and end positions in pixels. If the boundaries of the axis are the same, then the position of the vertex is the start point of the axis plus the width or height of the axis. The boundaries and start and end points for an axis are illustrated in Figure 6-8, as well as the result of the scatterplot itself.

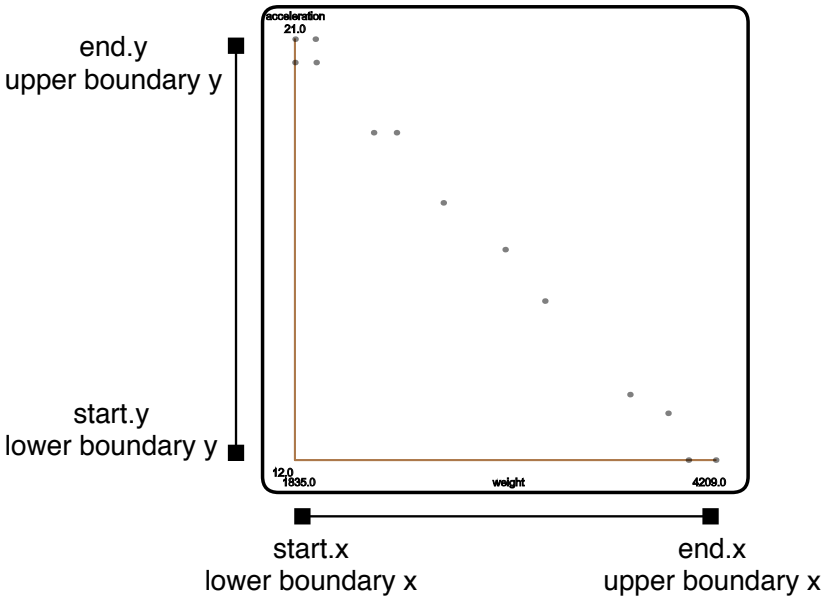

Figure 6-0-8: The scatterplot visualization type. X-axis lower boundary is 1835.0. X-axis upper boundary is 4209.0. Y-axis lower boundary is 12. Y-axis upper boundary is 21.0

6.4.4 Scatterplot Matrix

To implement a scatterplot matrix it was necessary to define two new classes, a class to hold scatterplots together in the matrix (ScatterMatrix), and an individual scatterplot class that exists within the matrix (ScatterPlotMatrix). The ScatterPlotMatrix class is identical to the previously defined scatterplot visualization class in Section 6.4.3; with the exception it does not have methods for interaction, and holds location, size, and data variables. ScatterMatrix contains a two-dimensional array of ScatterPlotMatrix. This relationship is illustrated in Figure 6-9.

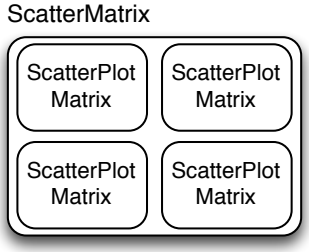

Figure 0-9: A ScatterMatrix holds a multidimensional array of ScatterPlotMatrix objects. The ScatterMatrix is a container for the objects

To define ScatterPlotMatrix objects two for loops are iterated. The logic in these two for loops is presented by the equation

$$\binom{A}{B} = \frac{A!}{B!\,(A - B)!}$$

Where A represents the outer loop index and B the inner loop index. This equation removes the duplicate and symmetrical relationships that occur when creating scatterplot matrixes based on a datasets dimensions. The pseudo code in Listing 6-6 displays the logic that implements this equation. Note that size subtracts two so that the scatter matrix does not account for linear relationships between dimensions.

Within the inner loop a new data model is created based on the selected rows and columns in the selected plot, where the columns are set as the loop index A and B. The data model is passed to a new ScatterPlotMatrix object that is then added into a ScatterMatrix with the loop indices A and B to identify its position within the two-dimensional array. The result of the scatterplot matrix object is illustrated in Figure 6-10.

```
set size to selectedPlot.selectedCols.size();
for (int a = 0; a < size - 2; a++) {
          for (int b = size - 1; b > a + 1; b--) {
               create array list rows from selectedPlot.SelectedRows();
               create array list columns from selectedPlot.SelectedCols();
               create DataModel temp from rows, cols;
               temp.setDataColumnsSize(a, b);
               create new ScatterPlotMatrix spm based on temp
               add spm to ScatterMatrix with indexes a, b - 2;
          }
}
```

Listing 6-6: Pseudo code for the logic that creates the scatterplot matrix.

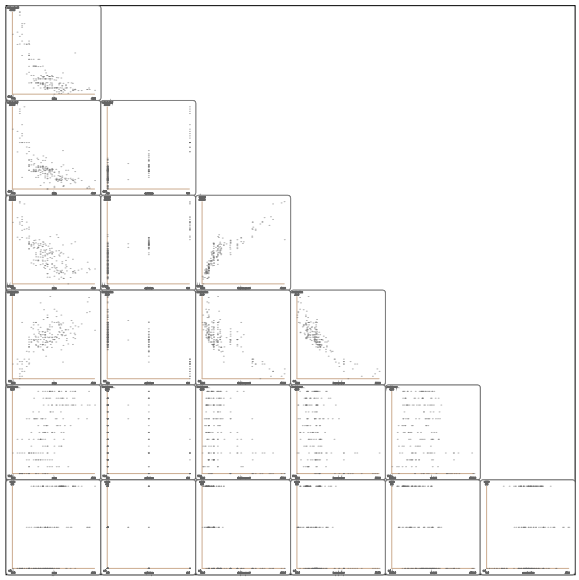

Figure 0-10: The scatterplot matrix visualization. Removing repeating and symmetrical relationships allows a singular display of all the relationships between a datasets dimensions.

An extraction can be made from a scatterplot matrix by dragging the individual scatterplot out of the bounds of the scatter matrix object. This is implemented in a three-phase process. First the indices of an individual scatter plot within the scatter matrix are stored if the plot is being moved and indices are not already assigned. The extractScatterPlot matrix method is called that checks whether the plot is outside the bounds of the scatter matrix. If the plot is outside the bounds then a new scatterplot is created as a copy from the scatterplot that has been moved. The scatter matrix is resized to its current width to realign the individual scatterplots in there respective locations. The logic for this is presented in Listing 6-7, whilst the process is illustrated in Figure 6-11.

Listing 7:This pseudo code is the logic for extracting an individual plot from a scatter matrix. First an object called a Checking Util is used to check whether the plot lies outside the boundary of the scatter matrix. The checking util holds three variables; a Boolean to state whether the plot is outside of the boundary, and the row and column of the outside plot. isOutside simply checks a scatterplot exceeds the boundary parameters and sets the Boolean of the checking util to true if it does. When a plot is declared outside the boundary a new scatterplot is created from the scatterplot variables and added to the canvas as its own plot.

```
Plots newPlot;
          CheckingUtil cu = isOutside(scatterplot row, scatterplot column);
          if (cu.isOutside) {//if outside boundary
              ScatterPlotMatrix spm = new ScatterPlotMatrix();
              spm = scatterMatrix.getScatters()[cu.row][cu.column];
              resize scatter matrix;
              newPlot = new Plot(spm);
              plots.add(newPlot);//stick into array list of plots on canvas
          }

function isOutside(){
CheckingUtil checking = new CheckingUtil();
        checking.setCheck(false);
        if (row not equal to -1 && col not equal to -1) {
            if (scatter matrix.scatterplot[row][colum] is outside boundary) {
                checking.setRowIndex(row);
                checking.setColumn(col);
                checking.setCheck(true);
            }
        }
        return checking;
}
```

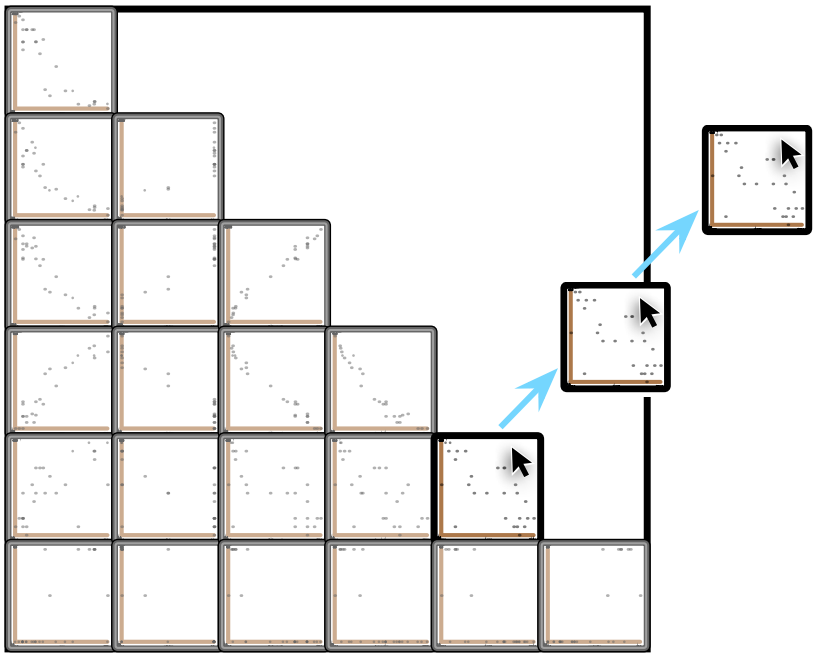

Figure 0-11:An illustration of the action required to extract a scatterplot from a scatter matrix onto the canvas as its own plot.

6.4.5 Stacked Bar Graphs

As stated in Section 2.2.4, individual stacks in a stacked bar graph are calculated as $percent = p/n$. To implement a stacked bar graph using multidimensional data there are two options. The first it too represent each column as a stack with the individual stacks representing the rows of the data. The second option is to represents a stack as row, with individual stacks representing columns. Essentially these two options are opposites of one another so it was decided that both should be implemented, where the user can decide what is most useful to them. The stacked bar graph is implemented as a structure that holds an array of stacks. Each stack object is responsible for drawing rectangles based on an array of floats stored within the class that correspond to the individual stack values in pixels. Individual stacks are calculated through mapping each data value between zero and the maximum data value for the column, and zero and the maximum height for the stack. An illustration of the result of the stacked bar graph is presented in Figure 6-11.

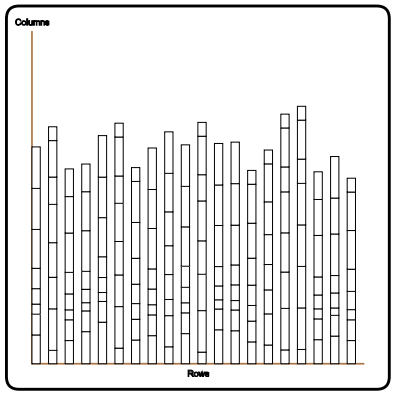

Figure 0-12:The stacked bar graph visualization

6.5 Editing Visualizations

One of the design goals of the realization sheet was to let the user visually edit the appearance of a plot. This section shall briefly discuss the implementation of plot editing for user defined size and appearance.

6.5.1 Resizing

Resizing a plot is implemented by recreating the visualization object with a new width; the height is set to the same value as the width as all plots have a uniform scale. The required width is calculated by taking the absolute value of the difference between the mouse x position V_X and the selected plot x position P_X. This is illustrated in Figure 6-12.

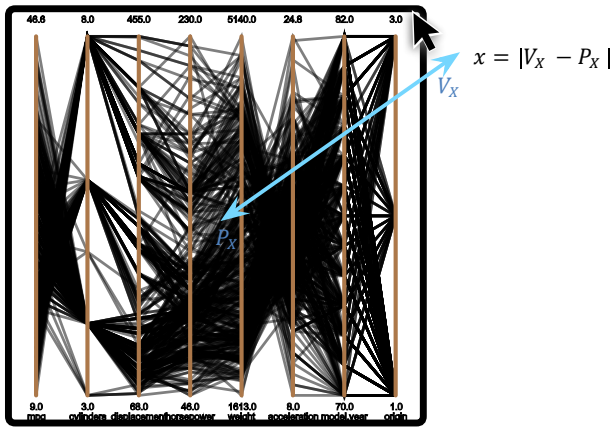

Figure 0-13: A parallel plot where the distance from the x position of the plot and the x position of the mouse is calculated. P_x Represents the x position of the plot, V_x represents the x position of the mouse. The equations specified that the value of x is equal to the absolute value of the difference between P_x and V_x.

All visualizations other then the scatterplot matrix are simple to recreate. The scatterplot matrix requires that every scatterplot within it be copied with its new position recalculated based on the new width and height. The logic for the resize method is presented in Listing 6.7.

```
set new sizes for plot;
if (this.layout != LayoutMode.SCATTERMATRIX) {
    new visual plot based on layout;
} else if (this.layout == LayoutMode.SCATTERMATRIX) {
    set scatPlot = new ScatterPlotMatrix();
    for (int i = 0; i < plot.scatterMatrix.getScatters().length; i++) {
        for (int j = 0; j < plot.scatterMatrix.getScatters()[i].length; j++) {
            scatPlot = plot.scatterMatrix.getScatters()[i][j];
            plot.scatterMatrix.getScatters()[i][j] = null;
            plot.scatterMatrix.getScatters()[i][j] = new ScatterPlotMatrix(new location calculated, old scatPlot parameters for copy operation);
        }
    }
}
```

Listing 8: Pseudo code for the logic to resize plots. All other plots asides from the scatter matrix are recreated. Every scatterplot in the scatterplot matrix is copied and its location is redefined based on the new width and height of the plot.

6.5.2 Appearance

Nearly every visual attribute of a plot can be modified in terms of its appearance from the layout menu. This is possible through storing the attributes as variables in the plot. Values that can be edited of a plot are the background colour, the border colour, and the specification for a border to exist. Furthermore individual axes of a plot can be edited in terms of width and colour. There is also the ability to spatially separate axes within a parallel coordinate plot.

Colour for a plot is set by passing a float value from the dock. This is performed automatically once a colour has been chosen from a J Colour Chooser component. Get methods also exist to update the colour buttons in the layout menu based on when a new plot has been selected with the mouse. The border of a plot is chosen to be visible by a Boolean operation; if the Boolean is true then the border is drawn, if it is false it is not. The Boolean value is specified from a J Check Box in the Layout menu.

Selecting an axis requires the user to hold down the a key and drag the mouse. Once an axis has been selected and appears in the layout menu, users can modify its size and colour. The axis is selected in the world with the point intersection method outlined in Section 6.4.2.1. Colour and size manipulation is achieved by storing the colour and size of the axis in the class itself, with relevant get and set methods to retrieve and refine the variables.

To spatially define the locations of a parallel coordinate plots axis the index to the current axis that the mouse is over is used to move its location. As the index of the axis is moved so must all other axes within the parallel coordinate plot. To achieve this two for loops are used, the first iterates through axes below the current index, the second iterates through axes above the current index. The axes are mapped between the differences from the outer most x position of the plot and the plots x position, and the outer most x location of the plot and the mouse's position. This is done for every axis except the furthest left and furthest right, where a separate distance is calculated for each axis. This is illustrated in Figure 6-13.

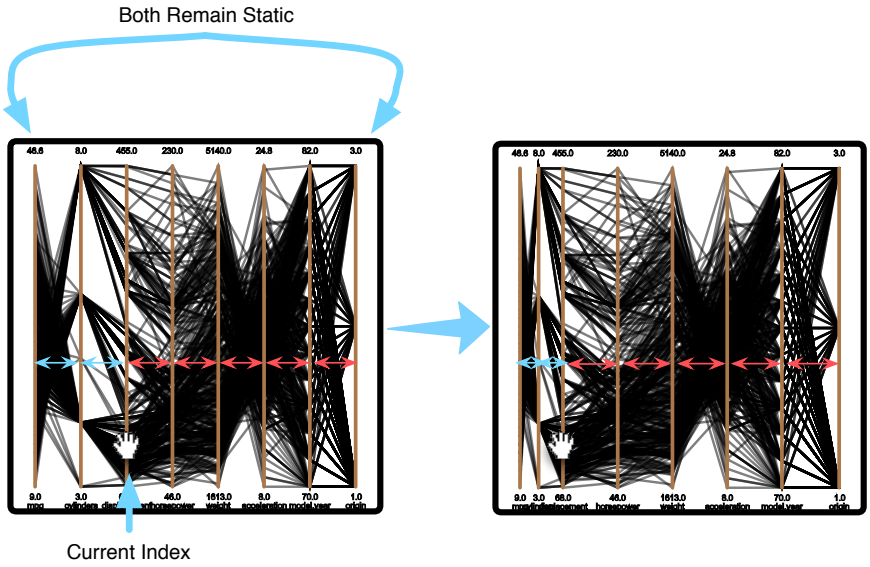

Figure 0-14:This represents the process used to move the individual axes in a parallel coordinate plot. The left plot illustrates a parallel coordinate plot before an axis is moved but an axis is selected. The blue markers between axes illustrate the distances that will be mapped before the selected axis index; the red markers illustrate the distances that will be mapped after the axis index. The right plot illustrates the result once the selected axis has been moved, where the distances between individual axes have changed.

Users may not be happy with changes they have made to a plot, for this reason an option to reset the plot appearance has been implemented. When a user adds a new plot, a data structure that holds the plots original appearance is created. This means for every plot on the canvas there is equally a structure that holds the original appearance in case the user decides to reset the plot. The original appearance objects are all stored within a list, that at all time corresponds to the size of the list that holds the plots. Resetting a plot is done through obtaining the original parameters from the relevant original appearance object in the list, and setting the current parameters of the plot to the original parameters. The logic for this is presented in Listing 6-8.

```
if (last plot mouse was over != -1) {
            plots.get(last plot reference).changeParameter(originalAppearance.get(last plot reference).oldParameter)
}
```

Listing 6-9: Pseudo code for resetting a parameter of a plot. An if statement checks there has been a relevant plot selected by checking if the reference is not equal to minus one. If a plot has been selected then this reference is used to identify the plot in the list and call the relevant method that changes a parameter. Since the lists are coordinated in size, the relevant original appearance parameter is selected from that list and changes the plot back to its original parameters.

Results for a variety of plots that have had the appearance and size configured are presented in Figure 6-14.

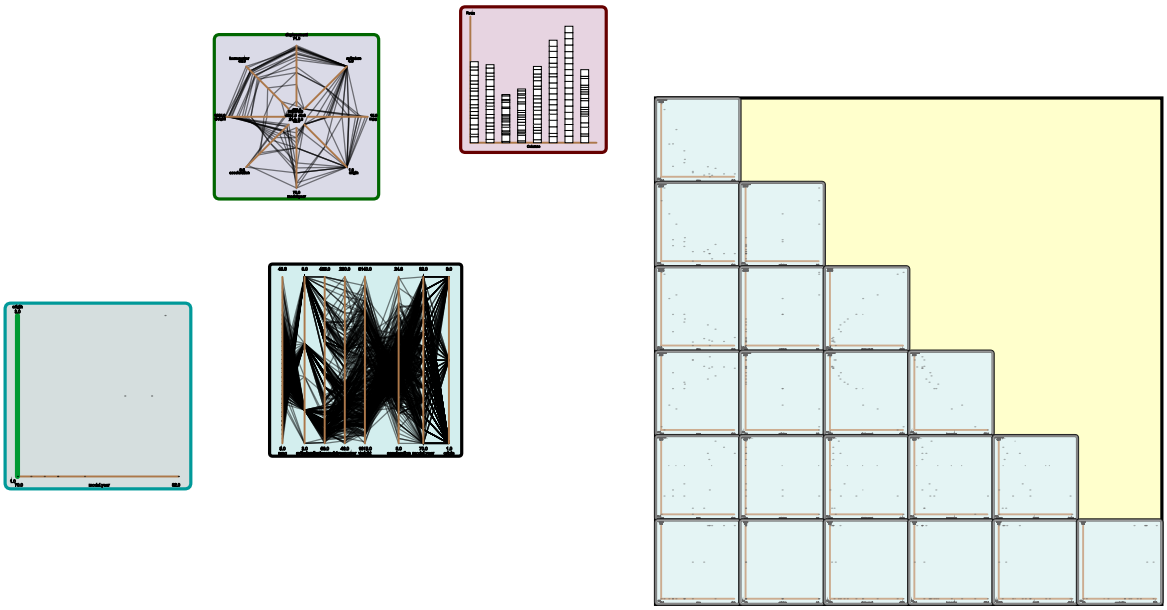

Figure 0-15: This illustrates many different types of plots with basic editing in terms of their size and appearance. All plots have some kind of colour changed whether it is the border or background. Specifically the scatterplot has the y-axis increased in size and highlighted in green, whilst the parallel coordinate plot has axes that have moved location.

6.6 Exploring Data

To address needs in visual analytics such as information overload [20], and the ability to understand relationships in data [19] Smart extraction mode was created as a method to browse brushed data from a plot. Smart extraction mode contains four features, data table searching, synchronous searching, numerical statistics, and regression lines. Data Tables store the brushed data as numerical values within a table, where the values can be individually selected. Synchronous searching allows values that have been brushed from one plot to be highlighted in other plots if they exist. Numerical statistics provides maximum, minimum, average, and standard deviation values for a brushed selection. Regression lines provide the line of best fit for a scatterplot. The implementation of these four features shall be discussed within this section.

6.6.1 Data Table Searching

Data values that have been brushed are stored within the data table by passing the selected plots data model to the data table model object when a selection has been made. Figure 6-15 illustrates the connecting classes the data model must be passed through to reach the data table model.

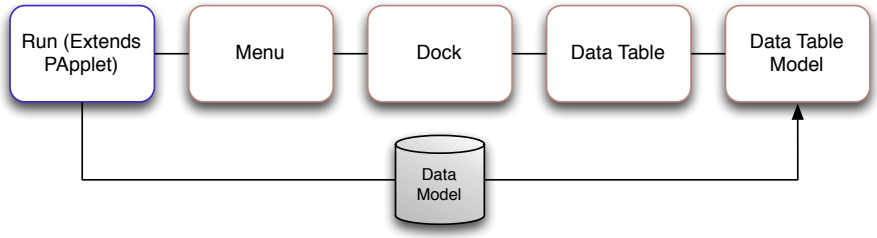

Figure 0-16:A representation of the class structure that the data model has to pass through to enter the data table model.

Once the data model has been passed to the data table model, data is copied from the selected rows and columns into the multi-dimensional object that exists populating the table with data. An array of indices is stored corresponding to the data in the table.

Values can be individually identified and selected within a plot by scrolling through the data table and selecting data values. Calling the filter data method that exists in the Data table continuously in the draw loop does this. Filter data gets the data row indices from the table and the indices for these in relation to the data, a for loop transfers the indices of the data that is selected in the table to a new array based on the position of the index in the data table. The logic for this is presented in Listing 6-9.

```
if (dataModel.getRowIndices() != null && table.getRowCount() != 0 && table.getSelectedRows() != null) {
            int[] indexes = table.getSelectedRows();
            dataIndices = new int[indexes.length];

            for (int i = 0; i < indexes.length; i++) {
                dataIndices[i] = dataModel.getRowIndices()[indexes[i]];
            }
            if (dataIndices.length >= 1) {//if there is actually something selected
                dataSelected = true;//then make the boolean true
            } else {
                dataSelected = false;
            }
        } else {
            dataSelected = false;
        }
```

Listing 6-10: This code presents the method filter data that organizes the data that has been selected into an array. First an if statement checks that some data exists in the table and rows are selected. The specific row indices in the table are stored into an array, and a new integer array with the same size is created. A for loop iterates over the selected indices and stores the actual data index of the selected row in the table into the new array. If this array contains any data then a Boolean dataSelected is set to true, if not it is set to false.

Figure 6-16 illustrates the flow that the data takes when it is filtered. After the data has been filtered and data is available, the Run method gets the data indices and calls the draw table based extraction method of the plot with the data indices as an input.

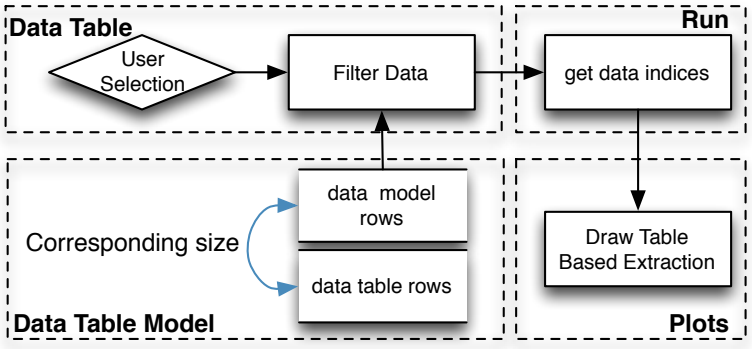

Figure 0-17: A representation of the process that the data takes in order to be highlighted both in the plot and the data table. The first action occurs within the data table where the user selects some data in the table, Filter data is continuously called and checks that data has been selected from the variable data model rows and data table rows. If data does exist then the run method gets the indices of the selected data and passes it to the draw table based extraction method in the Plot class.

The result of table based data searching is illustrated in Figure 6-17.

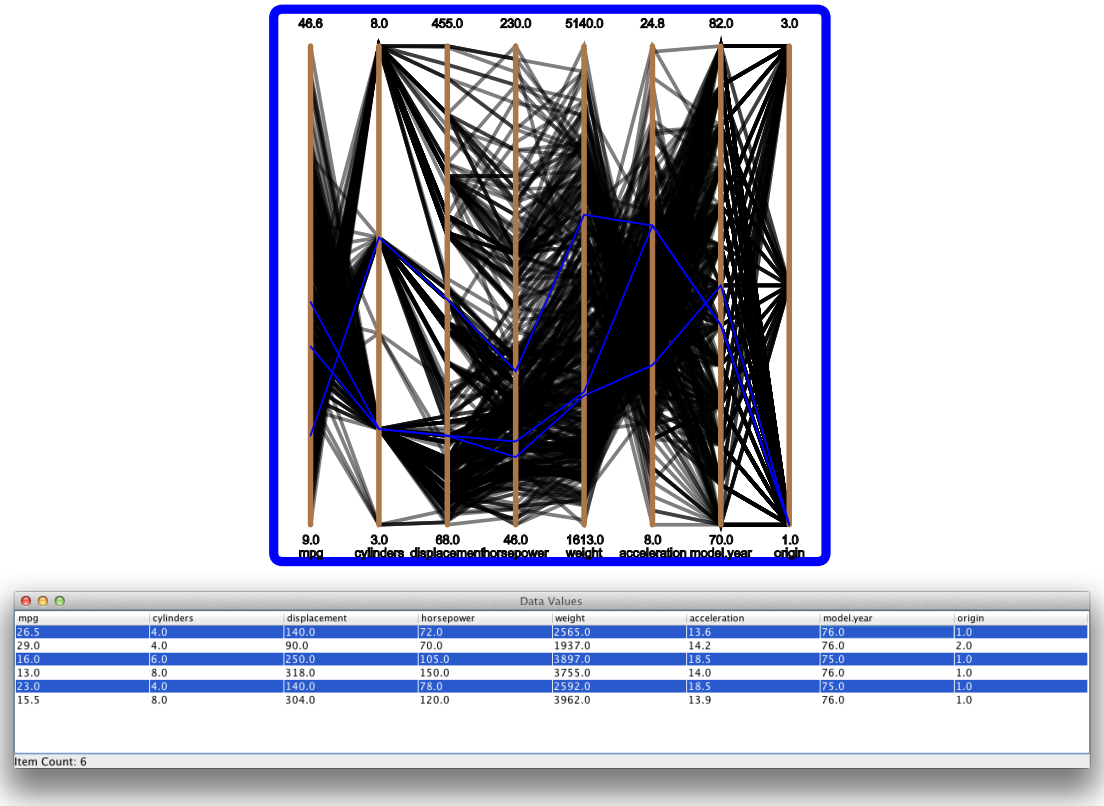

Figure 0-18: Data based table searching where three rows are selected from the table of brushed values and the matching polylines are displayed within the plot highlighted as blue.

6.6.2 Synchronous Plot Searching

The realization design specified that data values from a brushed selection from one plot on the canvas could be searched in all other plots to visually identify if they hold any of the same values. This could improve information overload [20] by visually 'popping' data points out that are the same, as well as supporting Schneidermans mantra [21] as users can gain an overview of data points that are the same and then zoom in to find the individual context. Using the indices retrieved from the selected rows in the data table, the aim is to search all the plots and check whether they contain the same values.

This is implemented by iterating over all the plots on the canvas and passing in the data indices to each plot to search whether the values exist. The search is implemented naively and simply using a logical equals test for two floating-point values, the logic is presented in Listing 6-10.

Listing 6-11:

```
Create ArrayList<Integer> highlightRows;//to store the same values
for (every row in this plot (i)) {
     for (every column in this plot(j)) {
          set float plotDataRow = data value at row i and col j in plot data;
          for (every row in data table indices (k)) {
               for (every column in data table indices(l)) {
                    set float plotDataRow2 = data value at row k and col l in data table data;
                    if (plotDataRow == plotDataRow2) {
                         highlightRows.add(data row in plotData at i);
                    }
               }
          }
     }
}
```

Once matching values have been stored in the array list, they are drawn as green points. The algorithm only highlights values within a scatterplot. This means you can make a brushed selection from any plot but currently only scatterplots will show similar values. An illustration of the search algorithm is presented in Figure 6-18.

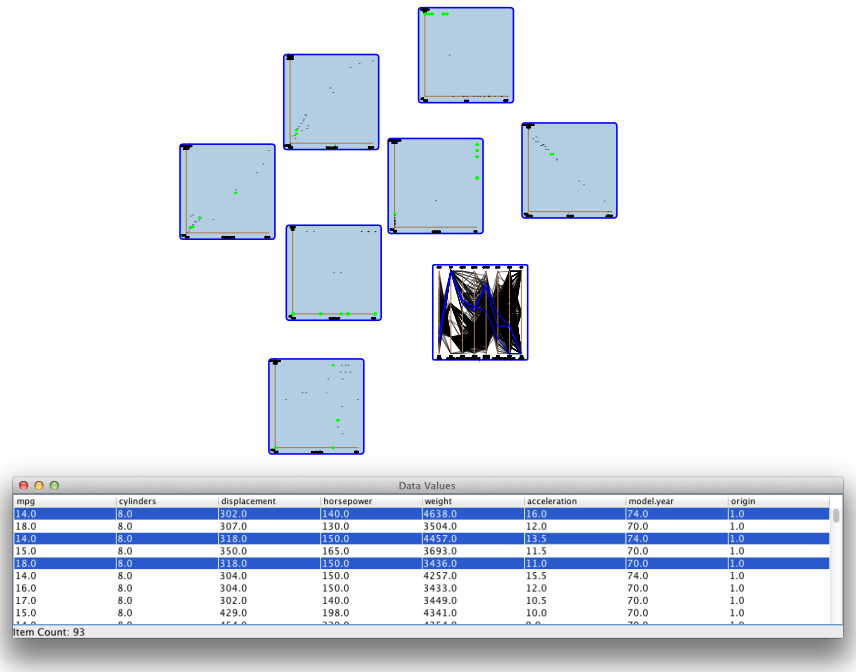

Figure 0-19: This figure illustrates synchronous plot searching. Rows are selected within the data table; values within this row are searched in every other plot to see if they exist. If the values do exist then points that match are highlighted in green.

6.6.3 Numerical Statistics

It was decided that users should have the ability to view statistics based on the brushed selection they had made. This is achieved by using a Statistic Panel class that is nearly identical to the Data Table with the exception of the format the data. The data is calculated once a brushed selection has been made, this includes the maximum value, minimum value, average value, and standard deviation for all columns within the data. This data is then passed to the Statistics Panel Model through the architecture presented in Figure 6-19.

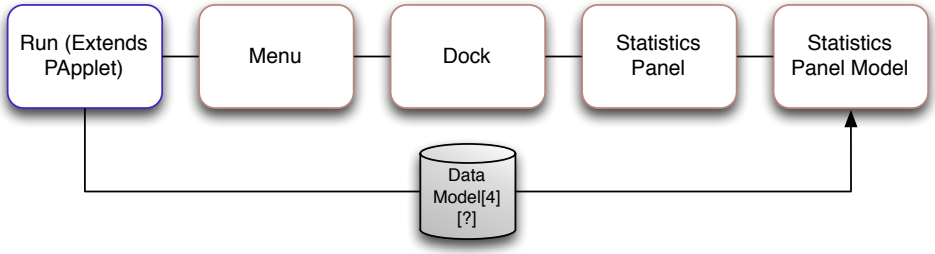

Figure 0-20:This figure illustrates the architecture that the data must pass through in order to reach the statistics panel model class. The data is a pre-calculated multidimensional array containing four rows that relate to the maximum, minimum, average, and standard deviation of each column.

The values are calculated through individual static methods that perform mathematical functions. getMaxForColumn takes a set of rows as an input along with an individual column index and calculates the maximum value for that column based on the rows. getMinForColumn, getMean and calculateStandardDeviation take the same inputs except calculate different values based on separate mathematical functions. The logic that calculates the statistics and passes them to the Statistics Panel Model is presented in Listing 6-11. The result of the statistics panel based on a brushed set of data is illustrated in Figure 6-20.

Listing 6-12: This pseudo code presents the logic that is used to pre calculate the values in the array. First an array is created, then an if statement checks data is selected within the data table; if it is then nothing is done. If no data is selected within the data table an if statement checks whether brushed data exists, if it does then a for loop iterates around how many columns exist, and mathematical statistics are calculated. The resulting array is then passed to the Statistics panel model.

```
float[][] values;
      if (data is selected in data table) {
      } else {
            if (brushed data exists) {
                  set float size as brushed selection column size;
                  set values as new float[4][size];
                  for (int i = 0; i < size; i++) {
                        values[0][i] = MathMethods.getMaxForColumn(brushed column at(i), brushed rows);
                        values[1][i] = MathMethods.getMinForColumn(brushed column at(i), brushed rows);
                        values[2][i] = MathMethods.getMean(brushed column at(i), brushed rows);
                        values[3][i] = MathMethods.calculateStandardDeviation(brushed column at(i), brushed rows);
                  }
            swing.getDockWindow().getStatisticsPanel().getDataModel().setValues(values);//pass them to the table
      }
}
```

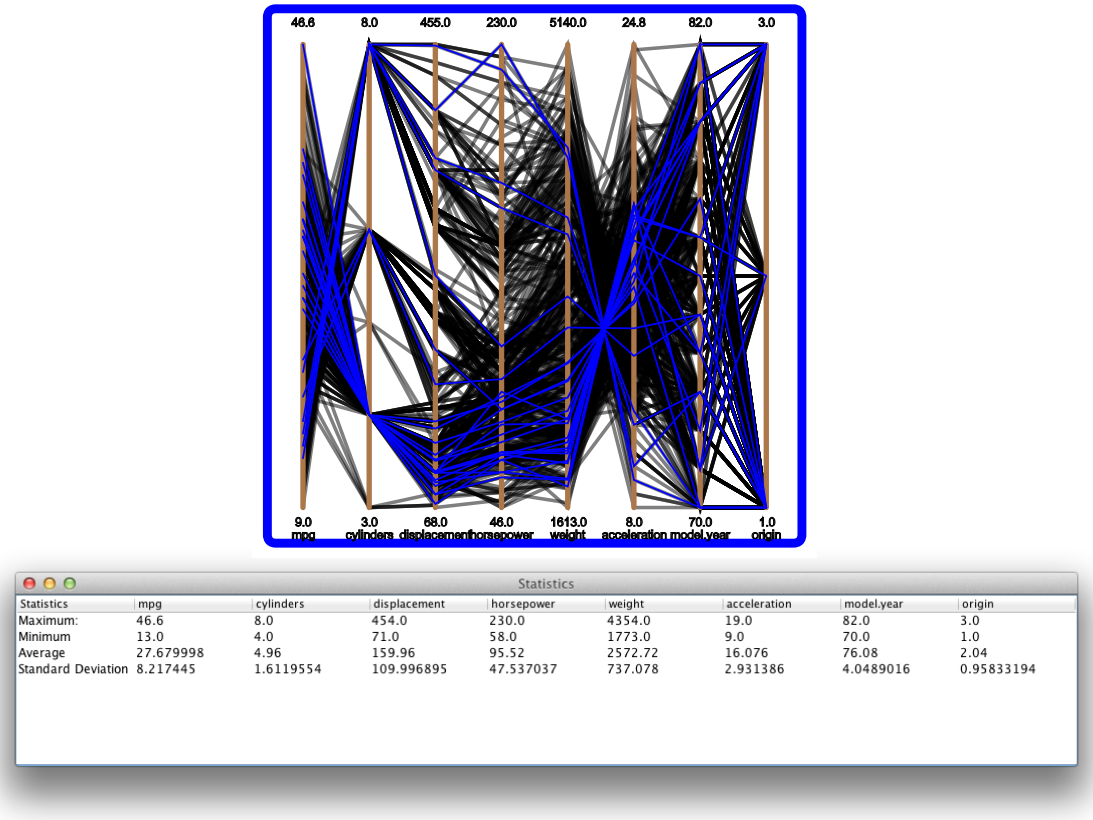

Figure 0-21: Statistics for a brushed panel are displayed in a separate panel. Every time a selection of polylines is brushed the statistics are recalculated and presented to the user.

6.6.4 Coordinated Multiple Views

The prototype had a simple form of multiple views implemented whereby the data of a child plot would be highlighted in its original parent. Since there are three new types of visualization, coordinated multiple views had to be considered and implemented for each type of plot in relation to one another. The eight design principles of coordinated multiple views have been considered [16], our multiple views use six of these being consistency, diversity, complimentary, parsimonious, self-evident, and attention management. These rules are summarized in relation to our implementation in Table 6-1.

Table 6-1: This table summarizes the relationships that the implementation shares with six of the rules that Aldonado et al define in relation to the use of coordinated multiple views.

Rule	**Implementation Relationship**
Diversity	*Separate types of visualizations are used that contain separate/relational data*
Complimentary	*Different plots can contain separate types of view that reorganize the axis as correlations using a Pearson correlation coefficient*
Parsimony	*Multiple views are chosen by the user, this can be minimal if they wish it to be.*
Self-Evidence	*The parent of a child plot highlights the data it contains when plots are browsed over with the mouse.*
Consistency	*There is only one multiple view operation that visually identifies data in parent plots, this is consistent.*
Attention Management	*Data is highlighted in parent plots in a separate colour and a trail is created for the flow the data has taken. This allows a user to move their attention to the separate plots that displays the relationship of extracted data that has been made to create the plot.*

In terms of relationships between linked views the use of one to many [18] is used to coordinate the views between parent and child plots. The operation that occurs between these plots is a highlight brushing operation [11] to display the identical data within a parent plot.

Data from Scatterplots is highlighted with context to multidimensional plots such as parallel coordinate plots and star plots by highlighting the columns the scatterplot relates to. Column indices are retrieved from the scatterplot and passed to the multidimensional plot where the axes are highlighted based on the column indices. Vertices are highlighted by taking the column indices and rows from the scatterplot and only drawing vertices for the rows between the column indices in the multidimensional plot. A result of a scatterplot to multidimensional plot highlight operation is illustrated in Figure 6-21. Scatterplots highlight points in relational scatterplots by passing the data columns and rows into the parent and then highlighting the rows within the parent scatterplot. The logic for drawing this data is presented in Listing 6-12.

Listing 6-13: This pseudo code presents the logic to draw data from scatterplots that exists within multidimensional plots. An if statement checks that the data that has been passed to the method is multidimensional, if it is then the normal draw is done. If not then a specific method is called that uses the data columns that have been passed in as indices for vertex points.

```
switch (layout) {
     set dataCols = columns indices passed into method;
     case MULTIDIMENSIONAL PLOT:
          if (number of dimensions in column data > 2) {
               //do usual draw
               multiDimensionalPlot.draw(row, dataCols);
          } else {
               //otherwise we're on a scatterplot so do specific draw
               multiDimensionalPlot.drawSpecific(row, dataCols);
          }
          break;
     case SCATTERPLOT:
          do usual draw;
          break;
}
method drawSpecific(input float row, input array columns){
Vec2D p1;
beginShape();
for (every column in columns) {
     p1 = map(row data value, axis[column].lowerBound, axis[column].upperBound, axis[column].start , axis[column].end);
     vertex(p1.x, p1.y);
     }
endShape();
}
```

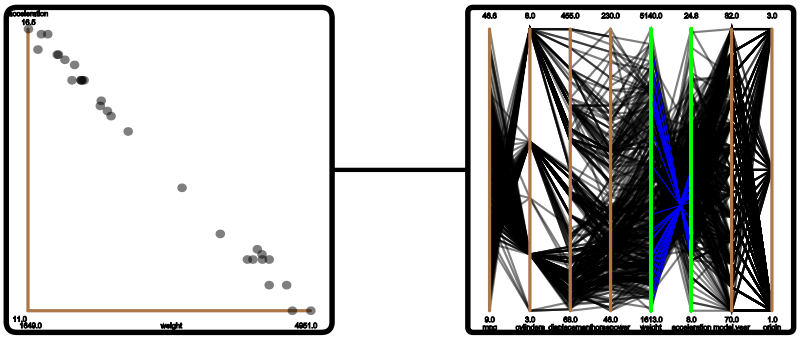

Figure 0-22: The highlight operation between a scatterplot and multidimensional plot illustrated where only the axis and vertices that the plot is relational to have been highlighted.

Scatterplot matrices have the same highlight operation as a scatterplot. The difference is that the structure of the multidimensional array of scatterplots that exist within the scatterplot matrix must be searched to find which one the mouse is over, and then retrieve the column indices based on that plot. Once the column indices have been retrieved the operation for drawing is the same logic as presented in Listing 6-12. A scatter matrix highlight is presented in Figure 6-22.

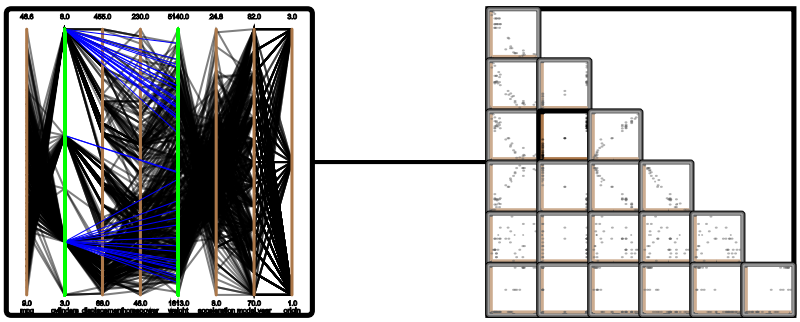

Figure 0-23: The highlight operation for a scatterplot matrix across multiple views. The individual scatterplot within the scatter matrix must be found when the mouse is over it. This scatterplots columns and rows are passed to the multidimensional plot and are highlighted.

Stacked bar graphs are highlighted within multidimensional plots based on the configuration of axes. If stacks represent an entire row then that row is highlighted within the multidimensional plot. If stacks represent columns then the relevant column to the stack is highlighted within the multidimensional plot. This is illustrated in Figure 6-23.

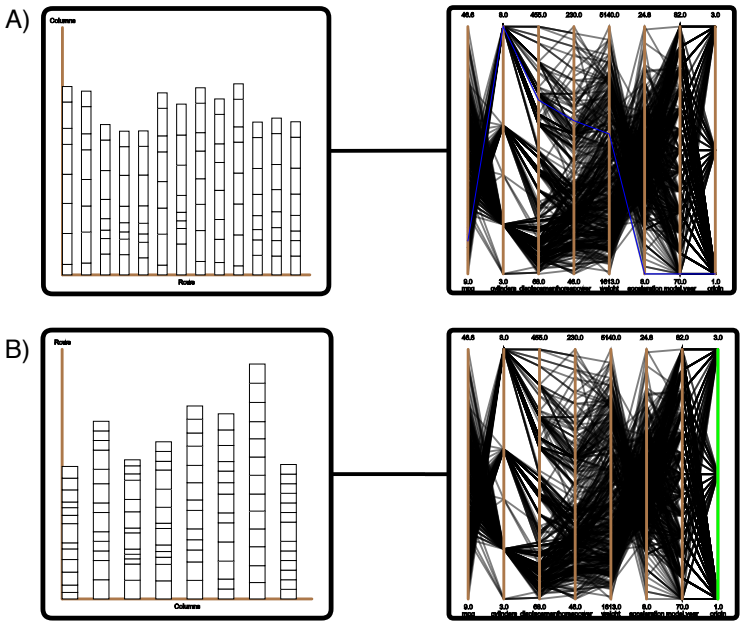

Figure 0-24: An overview of the different highlight operations for a stacked bar graph to a multidimensional plot. A) When a stack represents a row the individual row in the multidimensional plot is also highlighted. B) When a stack represents a column the column is highlighted in the multidimensional plot.

6.7 User Study

To address the specific features in the software and the process of visualization editing, it was necessary to define a format for a user study. The study was split into three tasks, following a walkthrough for the system, completing a set of tasks within the software, and finally completing a survey. These tasks would need to be completed in that order.

6.7.1 Walkthrough

For the user to understand the features of the software it was necessary to write a guide that would provide a structured approach to learning the system. In creating the guide it was necessary to think about the flow of the software and the best approach to learning it in a stepwise way. The most fundamental idea behind the software is making extractions and interacting with the canvas, this is the first area that is introduced to the user in the guide. The next fundamental piece of implementation is specifying the options within the menu for extractions and interaction capabilities; this is explained to the user by explaining each tabular menu interface in the docked menu. Every component is explained in relation to how it will affect the canvas or plot. Exploratory visualization is addressed through smart extraction mode; this is another fundamental software feature that is explained to the user. Features of smart extraction mode such as the data table, statistics panel, and synchronous searching are all explained to the user.

The walkthrough contains both a descriptive specification of features as well as asking the user to engage with these features by revealing results themselves. This allows a better learning curve for the user, as tasks are succinct with explanations allowing the user to understand the result of an action. Screenshots and diagrams illustrate results of processes that should happen when users perform an action, this helps the user to evaluate whether what they have done is correct.

Finally a quick reference to all the shortcuts and actions available to the user within the software was presented so the user could quickly lookup actions for certain commands. The walkthrough is presented in the Appendix Item ?.

6.7.2 User Tasks

To understand how well the system was implemented it was necessary to set users tasks that corresponded to new features within the software. Five questions were defined in terms of separate functions they would exploit within the software; questions shall be discussed individually with results defined.

6.7.2.1 Question One

The first question involves utilizing smart extraction mode to derive an average from a brushed selection of data. Users must first enter smart extraction mode, select the statistics panel and then brush a selection specified to locate the numerical result. This question was defined to evaluate how well users took to using the statistics panel within smart extraction mode. The route of the question is not apparently obvious as it is openly worded as

What is the average weight for cars with three cylinders?

Users may think to take an extraction route, or simply try and calculate the average based on the upper and lower limits of the axis. Both results would be wrong since a brushed selection is necessary to calculate the values for the averages of cars with three cylinders. The result is illustrated in Figure 6-25.

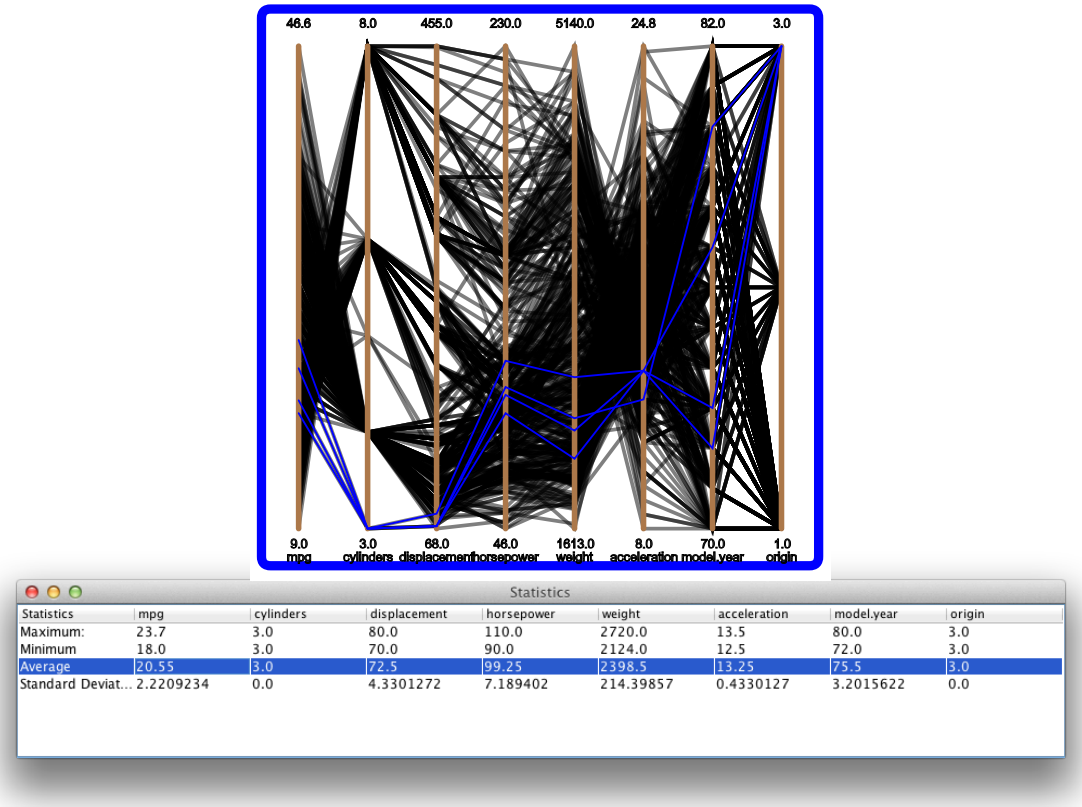

Figure 0-25: The answer to question one. Users must enter smart extraction mode make a brushed selection of cars with three cylinders and identify the average weight from the statistics panel, which is 2398.5.

6.7.2.2 Question Two

The second question is designed to make the user evaluate what type of visualization structure would best represent a relationship between weight and displacement for an extraction. The mention of spotting a linear trend helps to identify that an extraction of a scatterplot is needed. When the user investigates further they will find that it is impossible to make an extraction of a scatterplot from the axes of weight and displacement, as they are not organized linearly. Thus a visualization structure is required that represents all possible configurations of axes, a scatterplot matrix. The result of this question is presented in Figure 6-26.

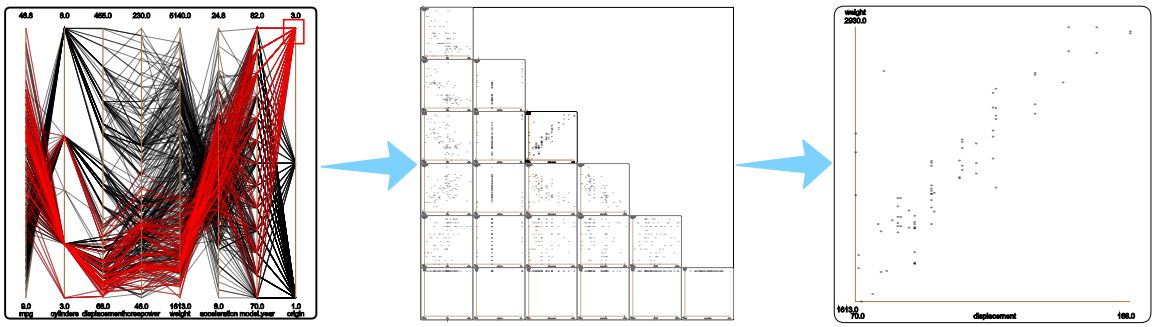

Figure 0-26: The answer to question two. Users must select cars with an origin of three with a rectangular or line brush tool to create a scatter matrix. They must then identify the plot within the scatterplot matrix that represents the relationship between weight and displacement. Once they have found the plot they can identify that a linear trend is present.

6.7.2.4 Question Three

Question three is defined to evaluate the menu system relating to the put interface. Users must create a parallel coordinate plot extraction based on cars that have six cylinders with the correlation option selected in the put menu bar. This will create a parallel coordinate

plot where the axes have been rearranged by correlation. The menu is evaluated as users will have to use it to define the arrangement of axes in a plot, if they do not think of using it, then the interface is not well designed. A result of the process to answering question three is illustrated in Figure 6-27.

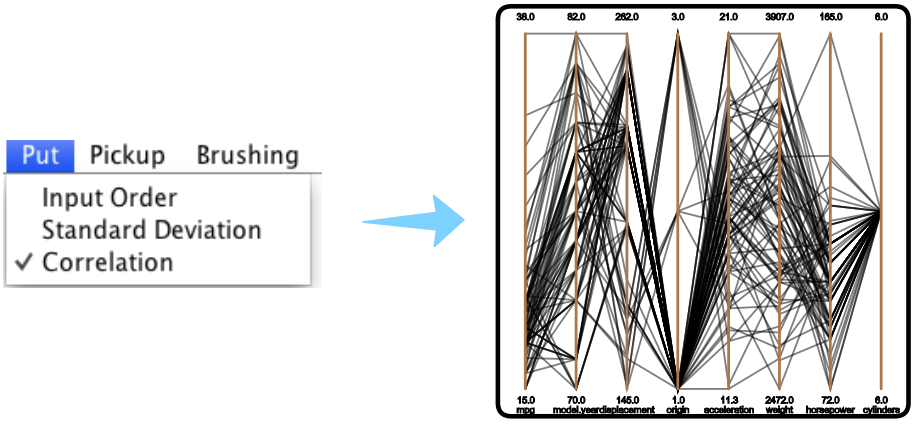

Figure 0-27: The answer to question three. Users must select the correlation option within the put menu bar, then make an extraction of a parallel coordinate plot based on cars with six cylinders. When the user looks at the axis arrangement they will see that MPG and displacement are not located next to one another, proving there is no correlation.

6.7.2.3 Question Four

The fourth question is designed to make the user find a result, and then visually edit the appearance of the plot. The task of finding the answer is straightforward and is nearly identical to the task presented in question one, except for a different brushed selection and separate statistic. Editing the plot is the purpose of the question, which requires the user to modify colour properties of both the border and background colors of the plot. Once this is done the user must output a PDF format image of the visually altered plot on the canvas. Outputting a PDF gives the opportunity to the user to understand how results can be exported from the software and used in separate contexts with other applications that support the PDF format. The result of question three is illustrated within Figure 6-27.

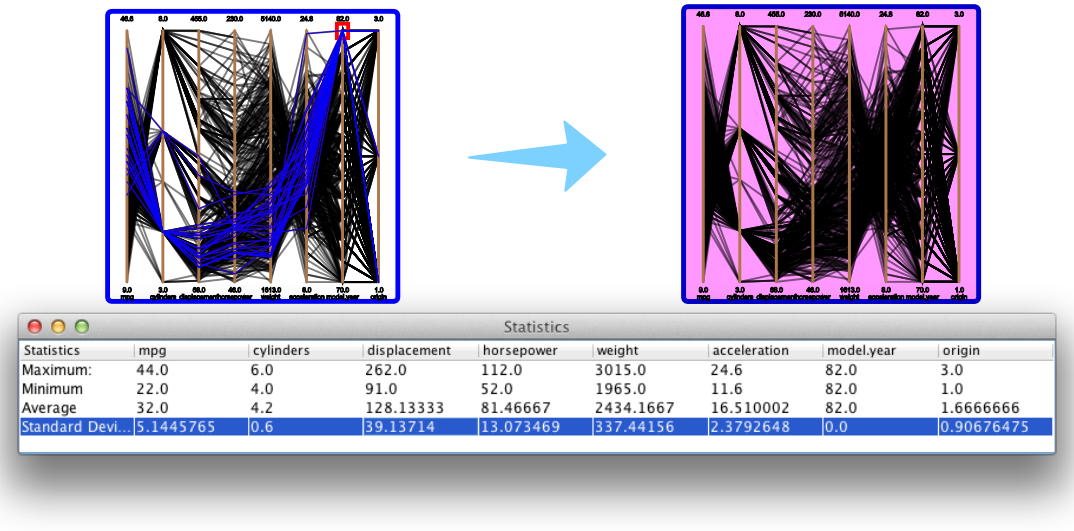

Figure 0-28: This presents the answer to question four. Users must use smart extraction mode with the statistics panel to make a brushed selection of cars from the year 1982. Alternatively they can make an extraction of another plot based on this and then enter smart extraction mode and perform the same process, selecting it in the parallel coordinate plot is a simpler option. Once they have found the statistic they must exit smart extraction mode and edit the background and border colour of the plot to pink.

6.7.2.4 Question Five

The final question is used to address the synchronous plot-searching feature within smart extraction mode. Users are told to make an extraction of scatterplots based on four

cylinders, the question is asked to evaluate whether any of these plots share values with cars that have three cylinders. The question requires users to brush the values of three cylinders within the parallel coordinate plot, and select the values in the data table. The question is designed to evaluate whether this specific feature of smart extraction mode is easy to use. The result of question five is illustrated in Figure 6-29.

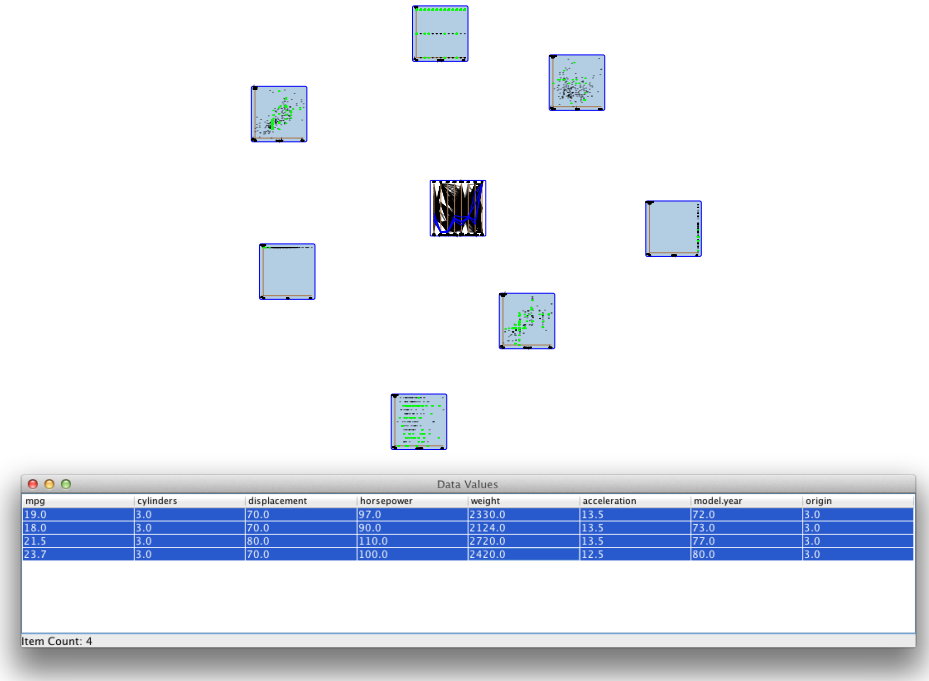

Figure 0-29:An illustration of the result for question five. Users must make an extraction of scatterplots for cars with four cylinders. They must then enter smart extraction mode and brush three cylinder cars from the parallel coordinate plot. Selecting the values within the data table reveals the result of if any brushed values occur within the scatterplots.

6.7.2 User Survey

A user survey has been implemented to ascertain what users thought of the software based on the tasks they have performed. The survey uses the System Usability Scale (SUS) [47] with a five point likert scale that ranges from strongly disagree to strongly agree. The system usability scale is used as it offers a simple format that is robust and reliable with the output of a single number that represents the usability of a system. Alternative biasing is used through the questions as it offers a technique that supports opposites in questions that test whether the user is actually answering the questions correctly. The survey for the software includes two sections. The first section includes ten questions relational to features in the software such as smart extraction mode, and visualization editing. The second section includes five questions assessing the performance of the five different visualizations within the system. To answer the user survey properly, users must perform the tasks so that they can form an opinion of what the questions are asking.

Both the user tasks and user survey can be found in the Appendix as item ?.

6.8 Summary

The software was implemented according to the design structure outlined in the realization sheet. Assessments of various builds of the implementation resulted in new features such as smart extraction mode being defined and implemented. The tabular interface was refined and implemented by using the Swing framework. The Swing framework has provided an array of standard components for the user to specify options in context to the put and pickup model, and visualization editing. New visualizations such as scatterplots, scatterplot matrixes, and stacked bar graphs have been added to the system to provide new visualization types for the user. Adding new visualizations refined the implementation of the architecture for plots, with the ability to support the inclusion of more visualization types. Editing has been implemented so the user can manually specify the appearance and size of any visualization. Smart extraction mode has been created, and implemented based on the need for purely exploratory visualization of existing plots on the canvas. Smart extraction mode provides several methods for the user to find specific values of interest based on direct and indirect manipulation techniques. Data table searching stores the data that has been brushed so users can visually select rows and coordinate them with the plot. Synchronous searching has been implemented to search for values that have been selected within the data table in other plots. Numerical statistics are provided to the user so they can easily extract information about brushed selections. Coordinated views have been refined to support the inclusion of the new visualization types. Other implementation features such as regression lines have not been discussed due to being buggy and incomplete. Finally the implementation of the user study was provided in terms of creating the walkthrough, tasks, and survey. The implementation was considered complete.

Chapter 7: Results

7.1 Overview

Having provided an extension to the existing prototype of Edivis to enhance features of visualization in terms of editing and exploratory analytics, it is necessary to consider how well the aims and objectives of the project were met.

The aims of the project were three-fold. That is to provide the user with more generic visualization structures that better represent relationships within the data. Provide a framework for the user to edit the appearance of visualizations. Finally to provide an approach to exploratory visualization that enhanced and aided the process for the user.

To address these aims a user study was performed that accommodated for these aims and addressed specific features of the implementation. The results of the user study shall be summarized with possible interpretations provided of how users responded to features that have been implemented. The results of the user study will be used to ascertain how well the aims of the project have been completed, and how effective features of the implementation are.

7.2 User Study Results

Based on the implemented user study presented in Section 6.7, black box testing was performed with the use of eight participants. Seven of the participants were male with one female. Five of the eight participants are currently on a postgraduate degree, whilst three are on an undergraduate. Each of the participants was considered knowledgeable in the domain of information visualization and understood the functionality parallel coordinate, star, scatter, scatter matrix, and stacked bar plots provided.

7.2.1 Task Results

The raw task results are presented in Table 7-1, where one represents a correct answer, and zero represents a wrong answer.

Table 3:This table contains the raw results of the user study tasks, where one is equal to a correct answer and zero is equal to a incorrect answer.

User	Question 1	Question 2	Question 3	Question 4	Question 5	Total
User 1	1	1	1	1	0	4
User 2	1	1	1	1	1	5
User 3	1	1	1	1	0	4
User 4	1	0	0	1	1	3
User 5	1	1	0	1	0	3
User 6	1	1	1	1	0	4
User 7	1	1	1	1	1	5
User 8	1	1	1	1	1	5

The raw data by itself is relatively meaningless, for this reason a stacked bar graph representing questions each user has answered as stacks is illustrated in Figure 7-1. Furthermore averages for the individual questions themselves were calculated and are presented in Figure 7-2.

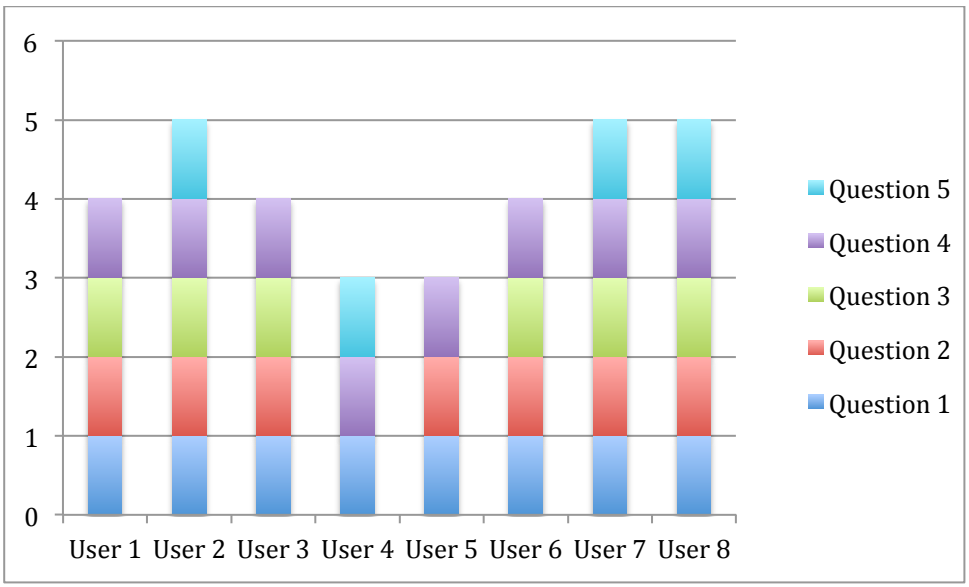

Figure 0-1: This represents the questions that users answered correct. Individual questions are colour coded so that an insight can be gained into the questions the user did not answer.

Figure 7-2 illustrates the total average for questions that the users answered correctly.

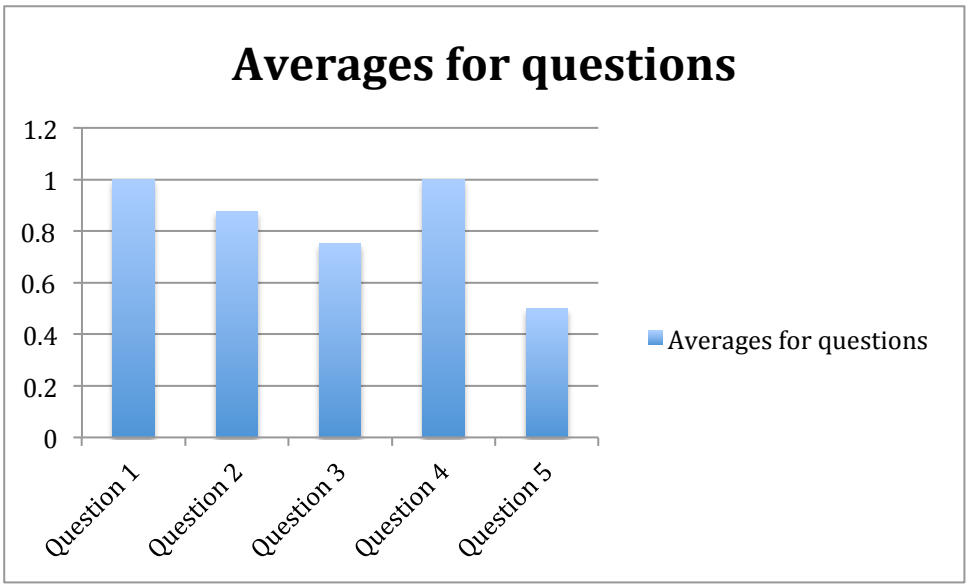

Figure 0-2: A bar graph that represents averages of questions users answered correctly. All users answered questions one and four correctly, ninety percent answered question two correctly, three quarters answered question three correctly, and half answered question five correctly.

Overall it can be seen that tasks relating to generating statistics, extractions, and using the interface were all relatively easy, with a high percentage of users completing the task correctly. Question five that relates to the feature of synchronous plot searching within smart extraction mode can be seen as confusing as half of the users did not answer the task correctly.

Finally Figure 7-3 presents the total amount of marks that users gained. This shows that an equal amount of users achieved a score of four and five, whilst two users received a score of two. This reveals that nobody had less then three correct answers with all users gaining above 60% for the test. The task results may be biased as if users got stuck they were assisted immediately with help. In most cases the answer was not revealed but an inclination towards the route of achieving the answer was given. If the user could still not work out an answer after being given help then the route to the answer was revealed.

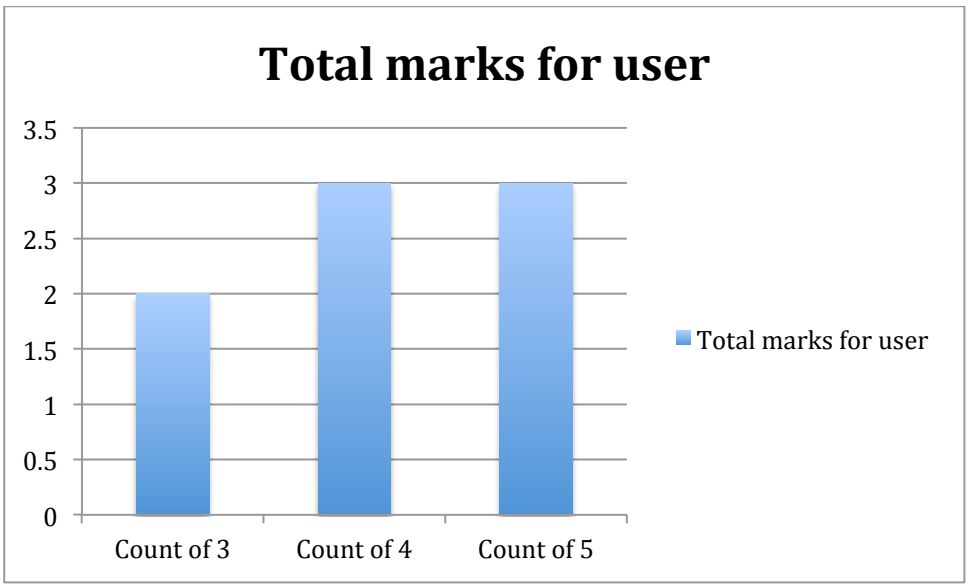

Figure 0-3: A bar graph illustrating the number of users who passed with certain points. The number of points is plotted on the x-axis, whilst the number of users is plotted on the y-axis

7.2.2 User Survey

To calculate the system usability results the SUS calculation methodology [47] had to be applied on the raw data, as without applying this methodology the raw data is meaningless. Once the calculation has been applied it revealed a total that corresponded to the usability of the system. Figure 7-4 presents a bar graph that illustrates these results for each user, whilst Table 7-2 presents the individual numerical values for each user.

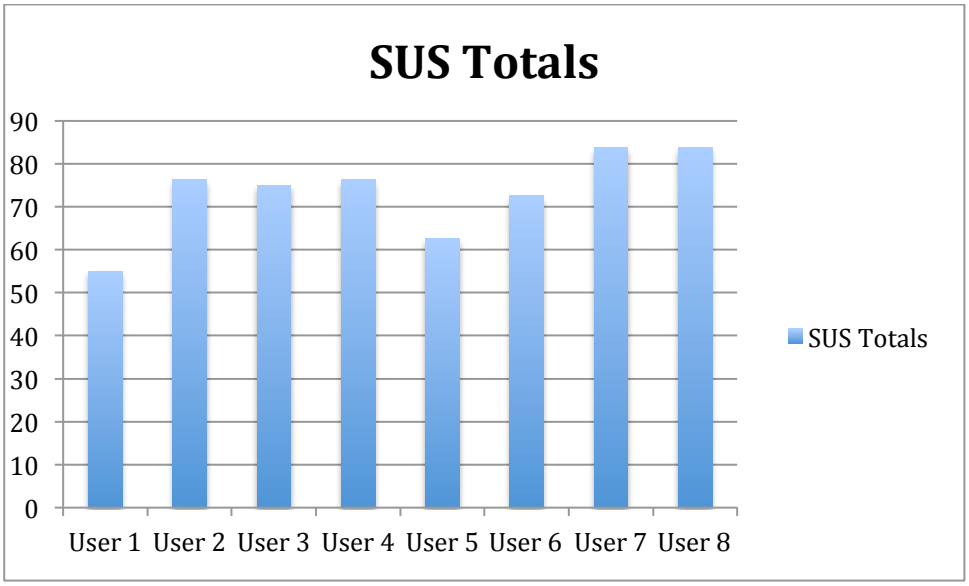

Figure 0-4: A bar graph that illustrates the SUS totals of the calculated results. The y-axis represents the different percentages, whilst the x-axis plots the individual users.

Table 4: The numerical percentages of calculated SUS totals for the user survey. These results correspond to the bar graph that is illustrated in Figure 7-4.

Users	SUS Total
User 1	55
User 2	76.25
User 3	75
User 4	76.25
User 5	62.5

User 6	72.5
User 7	83.75
User 8	83.75
Average system usability:	**73.125**

The SUS totals indicate that the system is highly usable based on an average result of 73 percent (rounded down) from eight users. The result is partially biased as users are performing a pre defined set of tasks that are relational to specific features of the implementation. Therefore this usability test is not a reflection on the general usability of the system, but a specific set of implemented features. The result is not completely biased as most of the user tasks exploit other features that exist within the software, providing the user with a partial view of the systems general usability. The questions for the user tasks are open ended, allowing multiple routes to be performed to achieve the same result. This open-ended approach means the test is relatively unbiased as users can define their own structure to perform tasks.

7.2.3 Plot Evaluation

Users were asked to rate the plots in terms of usefulness, Table 7-5 displays these results.

Table 7-5: The raw data to the plot evaluations are presented. Users were asked to evaluate the usefulness of plots from very useful (five) to not at all useful (one)

Users	Parallel	Star	Scatter	Scatter Matrix	Stacked
User 1	3	5	5	1	1
User 2	4	5	5	1	1
User 3	5	4	5	2	1
User 4	5	3	3	2	1
User 5	4	4	4	4	5
User 6	5	4	4	3	3
User 7	5	5	5	1	1
User 8	5	5	5	1	1
Plot Total	36	35	36	15	14
Plot Average:	7.2	7	7.2	3	2.8

Table results are partially biased as the user tasks only used plots such as parallel, scatter, and scatter matrix. It is partially unbiased as some tasks were open-ended allowing the user to define their own choice of plot; tasks that were not open-ended were tasks two and five. Figure 7-5 illustrates a graph of the average totals of the plots. These results reveal that users evaluated parallel coordinate plots, star plots, and scatterplots to be useful, whilst scatter matrix and stacked bar graphs were not useful. Task two requires the user to create a scatter matrix to represent a relationship between data, thus it is odd that users did not evaluate them to be useful.

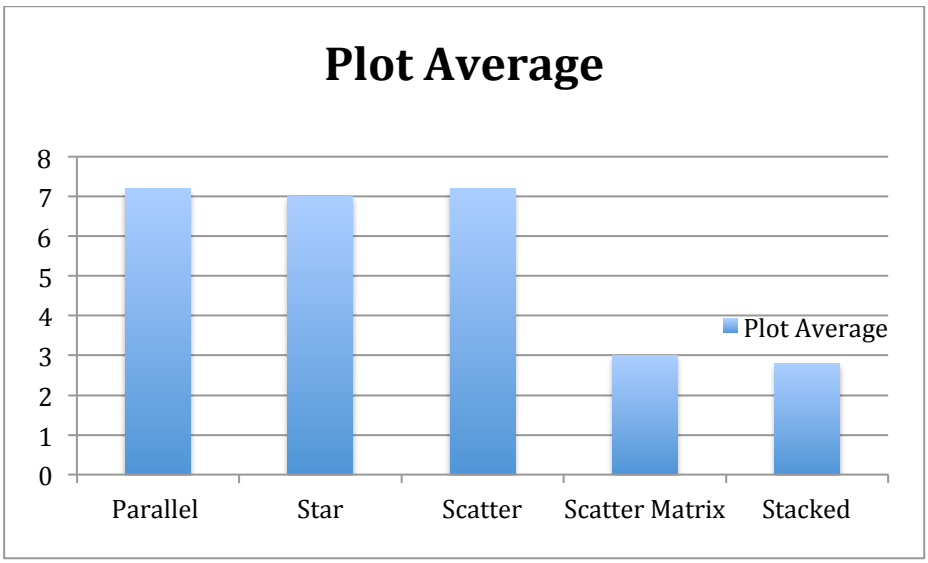

Figure 0-5: A bar graph illustrating the average totals for the user evaluation of separate plots within the software. Parallel coordinate, star, and scatter plots were all considered highly useful, whilst scatter matrix and stacked were not considered as useful.

7.3 System Performance Results

In order to evaluate the system in its entirety it is also necessary to retrieve results for the performance of the system. Since the nature of the system is interactive is it necessary to evaluate the frames per second (FPS) that the application maintains under specific situations. Interactive frame rates between 50 and 60 FPS that do not have a response time of more then 20ms are considered interactive. The system has been tested within three tests to evaluate the interactive FPS. The first test includes how many plots can be added to the canvas before frame rates drop. The second test includes how many points can be on the canvas before the frame rate drops. Finally the third test involves testing how many polylines can be put on the canvas before the frame rates drop. The tests were all performed on a Mac Mini that has a 2.7GHz Intel Core i7 processor, with 8GB 1333MHz DDR3 RAM, and an AMD Radeon HD 6630M 256MB graphics card.

7.3.1 Multi Plot Rendering

The first test involves adding as many separate types of plot to the canvas to asses the FPS of the system. Figure 7-6 presents a line graph that details the FPS for how many plots are on the canvas. Figure 7-7 presents the output of the canvas at the final point of 56 plots.

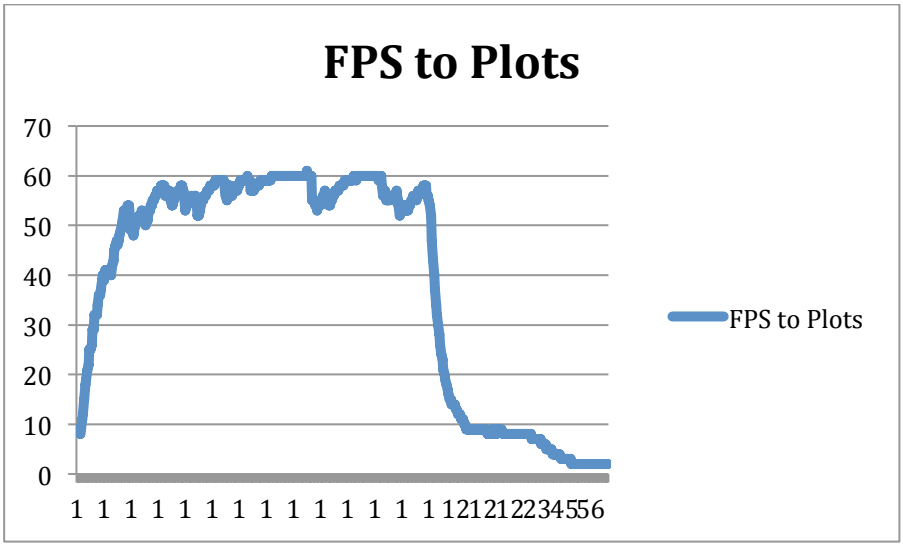

Figure 7-0-6: A line graph representing the FPS for the number of Plots that are displayed on a canvas. The FPS drops to around two FPS with a total of 56 plots on the canvas

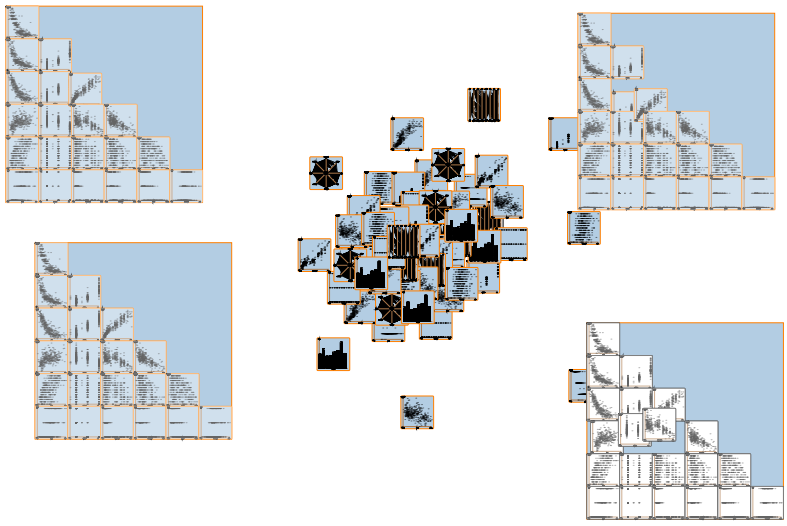

Figure 0-7: The result of 56 plots being laid onto the canvas; the system was still responsive (it rendered this vector output) however interactive capabilities on the plot were reduced to two FPS.

These results are completely unbiased as they are outputted directly from the application into a Comma Separated Value (CSV) file. The raw data that correspond to Figure 7-6 is presented in the Appendix as Item ?. From analyzing the line graph in Figure 7-6, it can be seen that as soon as new plots are added to the canvas the FPS drops at an exponential rate, from an interactive rate of 50-60FPS to less then 10 FPS. This continuously drops as more plots are added to the canvas until the system is reduced to 2FPS with 56 plots on.

7.3.2 Polyline Rendering

Rendering polylines is equivalent to rendering simple connections of vertices as no OpenGL constant is specified within the implementation. This test checks the frame rate for how many polylines appear on screen. A line graph is illustrated in Figure 7-8 that represents that results of this test, the raw data is available on the CD attached to this thesis.

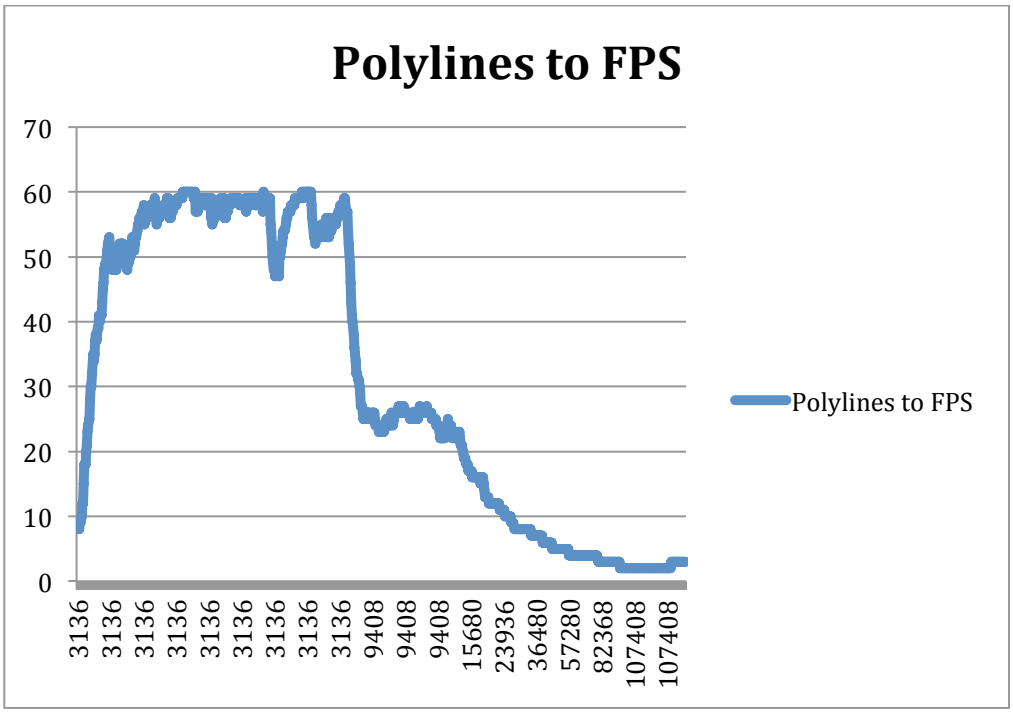

Figure 0-8: A line graph displaying the results of the FPS to the amount of polylines on the canvas. The polyline count is plotted along the x-axis, whilst the FPS is plotted on the y-axis.

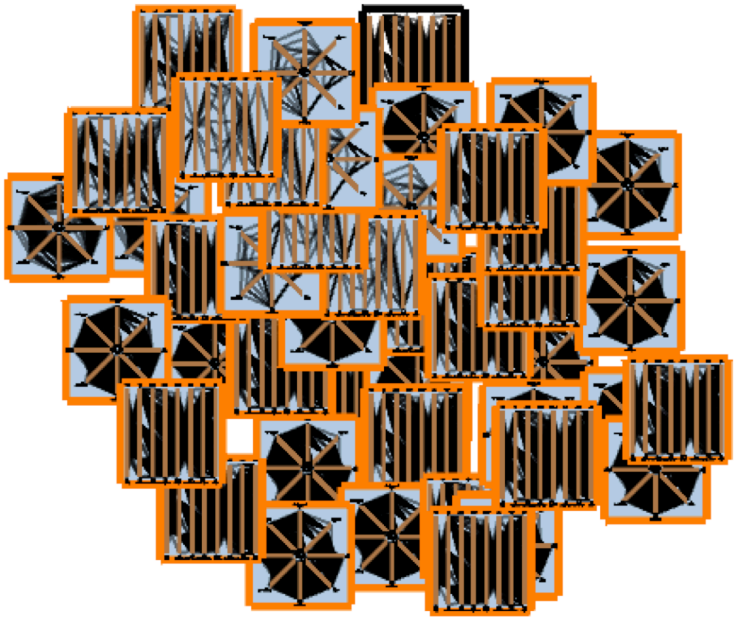

Figure 0-9: An output of the result of rendering 107,408 vertices to the canvas at once. The FPS was reduced to one and the system completely irresponsive.

The results of this test allow 107,408 vertices to be rendered to the screen before the system completely stopped at one FPS. It is interesting to note that the system will still run smoothly whilst rendering 9408 vertices to the screen allowing a small number of parallel and star plots to be rendered on the canvas. As the number of vertices increase the system performance gradually slows down to become irresponsive.

7.3.3 Point Rendering

The implementation of scatterplots and scatterplot matrixes specify the use of the OpenGL Points constant. To evaluate the performance of the system in terms of point rendering it is necessary to perform a test to check the FPS in correspondence to how many points are on screen. Figure 7-10 illustrates a line graph that corresponds to the FPS for the amount of data points on the canvas.

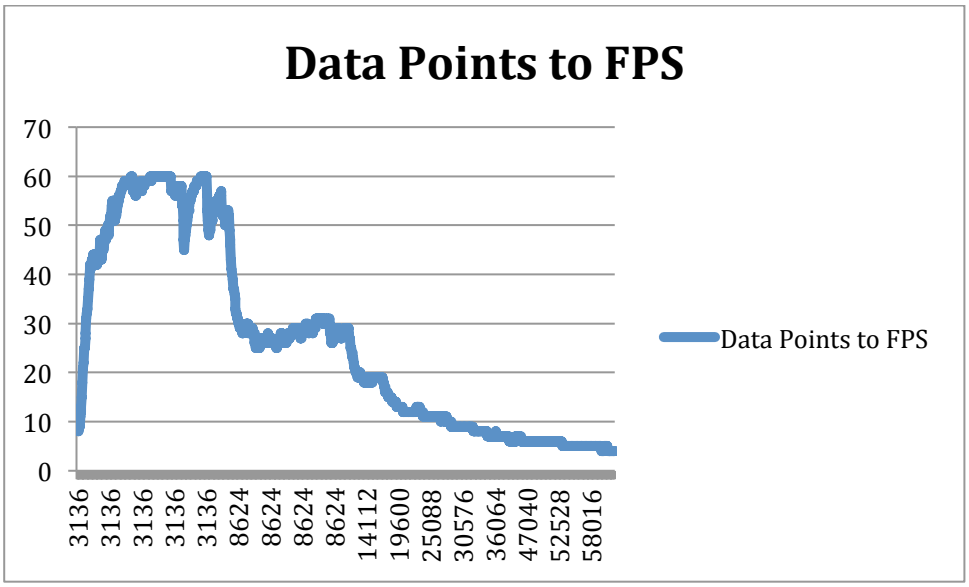

Figure 0-10: A line graph displaying the results of the FPS performance of the system in relation to how many points are being rendered on the screen. Data point values are plotted along the x-axis whilst the FPS relevant to the data value is plotted along the y-axis.

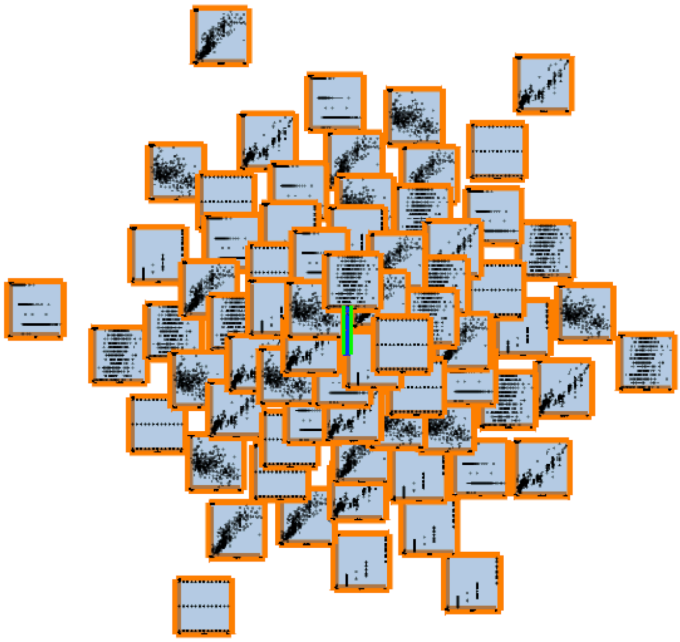

Figure 0-11: The result of 58,016 data points being rendered to the canvas across multiple scatterplots. Scatter matrix plots were not used in this test as they are rendered in an iterative way for each object within the plot the plot itself.

The result is completely unbiased as only scatterplots are used to render points, if scatter matrixes were used, the process of rendering points would affect the result as the implementation of rendering points in a scatter matrix is defined by a separate process to that of scatterplots. The rendering of points maintains interactivity until the amount of points increases to 14,112, from here a linear drop in FPS is evident until 58,016 points are rendered onto the screen making the system unusable at a frame rate of four. The maximum number of points that can be rendered to the canvas is approximately half of the amount of vertices that can be rendered; this may be due to using the OpenGL constant Points as opposed to using standard vertices (which would not render anything to the plot).

7.3.4 Data Searching

It was necessary to evaluate the search algorithm, to do this results would be needed that corresponded to how long it took to search for values in other plots based on the number of data points being searched within the data table. A slow search algorithm can render a system unusable and reduce the FPS of the system, if the response time of a visual search algorithm takes more then 20ms to display its results it is considered slow. Figure 7-12 illustrates a line graph that plots the results that the search algorithm takes in milliseconds against the amount of points being searched. The search was performed with eight scatter plots with a brushed selection of all 392 rows within the parallel coordinate plot.

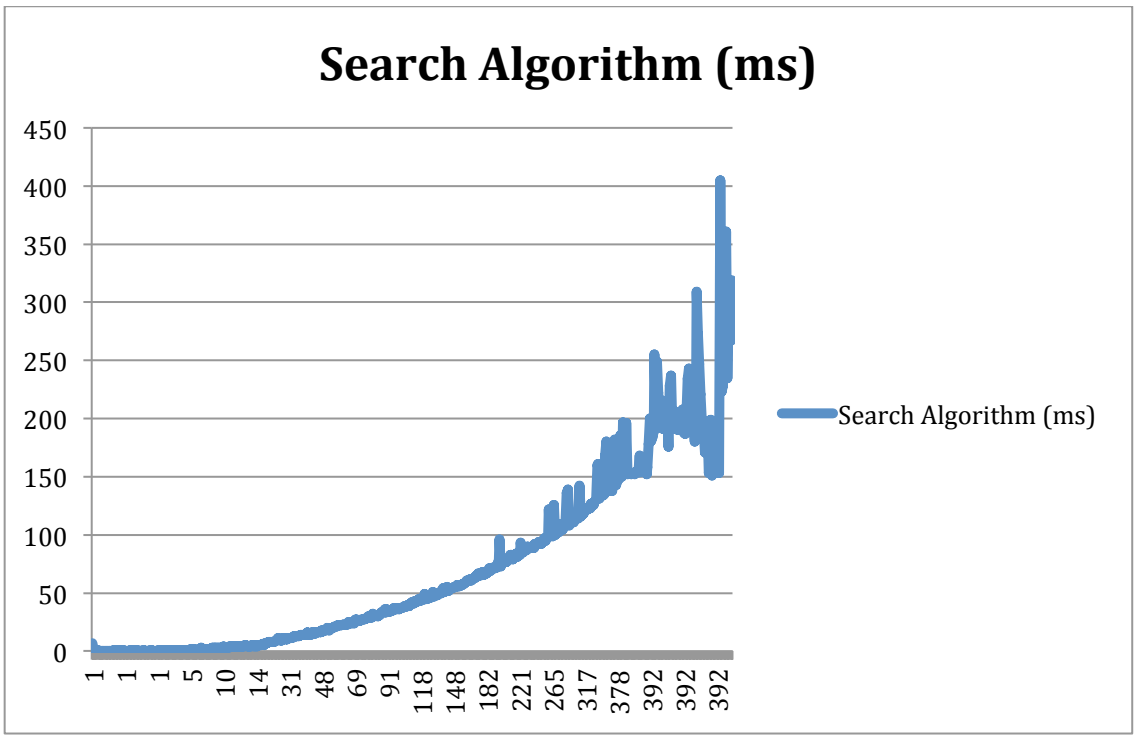

Figure 0-12: A line graph illustrating the time (in ms) that the search algorithm takes to complete based on the number of points it is searching. The time is plotted on the y-axis, whilst the number of data points is plotted on the x-axis.

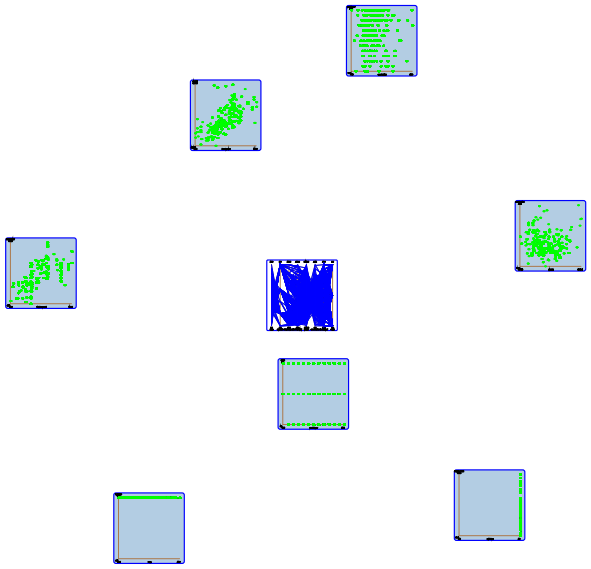

Figure 0-13: This is the result of all the points being searched within eight scatterplots. 392 rows are highlighted with each row being searched in each plot to check if it contains a matching value.

The results of the search illustrate that as the number of points goes over around 55 the time the search takes to complete goes over the boundary of 20ms, this progressively gets worse as the number of points increase. The results have a consistent linear trend until 392 points are searched; this is due to trying to export a PDF and PNG of the results. This shows that interaction with the canvas whilst points are being searched drastically affects the time the search takes to perform. The result is biased as only eight scatterplots are used within the search, if more scatterplots were used then the search time would have been higher from the start and been more drastic as more rows of data were searched.

7.4 Summary

The usability and performance of the extended software were both tested with results defined. The usability of the system defined three results; task results, survey results, and plot evaluation results. The task results illustrated what tasks users excelled and struggled with in relation to the software implementation. Survey results defined the usability of the system through using the system usability scale; it was revealed that the system had a good average rating of usability. Plot evaluation results demonstrated the plots that users preferred for accomplishing tasks; parallel coordinate, star, and scatter plots were all rated highly in terms of usability, whilst scatter matrixes, and stacked bar graphs were rated negatively. The system performance was evaluated utilizing four specific tests, point and polyline rendering, multiple visualization type rendering, and search algorithm performance. Point rendering revealed that only 58,016 points could be rendered to the canvas before the system stopped being usable. Polyline rendering revealed that 107,408 vertices could be rendered to the canvas before the system was no usable. Multiple type rendering illustrated that only 56 plots could be on the canvas at one time before interactive frame rates dropped. Finally search algorithm performance results demonstrated how inefficient and naïve the search algorithm is in its performance.

Chapter 8: Discussion

8.1 Overview

Chapter seven defined a set of results relating to the usability and performance of the implemented software. An evaluation of how well the software has extended the put and pickup model shall be discussed. To evaluate the software the results shall be utilized in respect to the implemented features of the software. Four main features of the implementation shall be discussed, the menu interface, the different visualization types, visualization editing, and finally smart extraction mode.

8.2 Put and Pickup Model

The software extended the put and pickup model by implementing new visualizations, and techniques. New visual structures successfully extended the put model by creating an array of separate types of visualization structure that are possible to put on the canvas, as opposed to just parallel coordinate plots.

With new visualization types to put on the canvas, the relevant pickup interactions had to be implemented. Pickup interactions were specified for the axes and data points on scatter plots, allowing extractions to be placed onto the canvas from this visualization. It is possible to pickup any data from a parallel coordinate, scatter, and star plot to put onto the canvas in a chosen visualization; this is illustrated in Figure 8-1. Scatterplot matrix extractions were handled by dragging an individual plot outside of the scatterplot matrix boundary, this converted the individual plot to a normal scatterplot. This allows individual plots within a scatterplot matrix to benefit from all the pickup operations available in scatterplots by redefining the format of plot. Further implementation to various pickup operations that are possible with visualizations would be an advantage to extending the model.

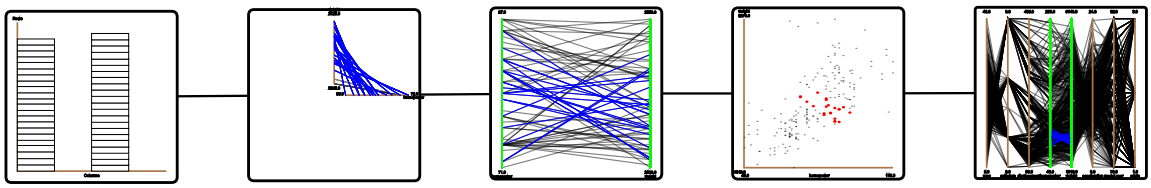

Figure 0-1: Multiple visualizations that have been put onto the canvas from picked up data selections. It is possible to put any data visualization structure onto the canvas, however it may not be possible to pickup data from the structure.

Drawing is the main feature of the put model that has not been extended due to the complexity of implementation it would require. So that drawing specific components onto the canvas could occur, the software would need to be redesigned completely to allow drawing interactions. This was not possible for the defined time period of implementation.

Finally synchronous plot searching was considered an effective extension to the pickup model as values that had been picked up were evaluated as to whether they exist in other plots on the canvas.

8.3 Software Extension

The implemented extension to the software provided many features that through combinatorial properties could define new derivations of existing data. It is necessary to discuss each of the implemented features respective of the results to define advantages and disadvantages. Finally a holistic evaluation shall be provided for an overview of the software as a whole.

8.3.1 Menu Interface

A docked menu interface was implemented to replace the old ControlP5 interface. The menus options were redefined to better correspond to the separate options in the put and pickup model. As a wealth of new implementation options were added to the software the menu interface provided users with a quick and easy way to specify these. The new menu interface was received positively in the user test. Users found options and components easy to find and use as they kept with the standardized look and interaction of the Swing framework. No users asked for help with navigating the user interface. One comment from a user was that the interface should support more visual feedback to the current active options in the software, this would help provide the current context of the application to the user at all times. Another comment was relational to the menu structure in that there should be an export option to specify a file directory and format to output plots on the canvas.

The dock and menu does not affect the performance of the visualization system; this is because the context of Swing components is handled in there own separate threads with relevant event listeners. The dock could be improved by managing the space that the data table and statistics panel occupy. Whilst these windows are user defined in terms of size and location, they need to remain at the forefront when performing tasks such as data searching. The dock requires an option to stay focused instead of being defined by the mouse location; this would allow the visual inclusion of the dock to be user defined as opposed to software defined.

8.3.2 Visualizations

User results defined that the parallel coordinate, star, and scatter plots are most useful. This merits that stacked bar graphs and scatter matrixes require further research and implementation within the software. Parallel coordinate plots and star plots have not been extended within the implementation; features that multi-dimensional plots would benefit from are multi-dimensional range brushes [6], angular brushing [7], and cubic or quadratic curves [4]. These features could be user defined whereby they could choose to alter the polylines from normal straight appearances to quadratic curves, or make brushed selections based on an angle or specific range selections. This set of features would enhance the concept of visualization editing, as components of the visualization structure can be manually defined. These specific features were not focused on, as one of the main objectives was to introduce new visualizations into the system.

New visualizations such as scatterplots and scatterplot matrixes were one of the key implementation features that are used to define relationships for the data. Previously the system was limited to parallel coordinate and star plots. Scatterplots work well as they are implemented in terms of appearance, interaction, and extraction. Users found it a natural process to use scatterplots, as they needed no instruction, and could interact and extract new data from the plots easily. Scatterplots would benefit from automatic regression lines so that the software can spot linear trends automatically and identify them to the user. An implementation of regression lines was attempted; however the mathematics that was defined for clipping the line for the plot boundaries did not translate from the test application that was made, into the system implementation.

Scatterplot matrixes were implemented to display all possible relationships across axes in a multidimensional plot. Users found these plots difficult to use as it was not immediately apparent what the plot did, and how to use it. Making extractions from a scatterplot matrix is a difficult process that is not immediately apparent, whilst interaction is not defined for a

scatterplot matrix. Scatterplot matrixes could be improved through defining interaction capabilities, and a more logical extraction implementation.

Stacked bar graphs proved to be the least effective visualization method, as they do not offer any insightful information to data and interaction capabilities are not defined within the system. Stacked bar graphs need to be reevaluated in terms of applicability to the implementation and if they are required, this will redefine their purpose, interaction, and extraction capabilities.

The performance of visualizations in terms of rendering vertices was defined in Section 7.3.2, and rendering point vertices in Section 7.3.3. Rendering above 10,000 vertices or points made the system drop to below interactive frame rates, for an interactive visualization application this is not ideal. The Info Vis toolkit [32] can render 20,000 points at a frame rate of 60FPS based on efficiencies within the framework, the software will need to improve the interactive frame rate to compete with other packages. To improve the performance of vertex rendering, vertex buffered objects could be used to store the vertices within the system. Parallelized algorithms could be utilized when calculating new extractions for plots. An off-screen buffer could calculate the locations and appearance for all the plots on the canvas and render an image when attributes change. Finally the physics system could be removed as it only serves the purpose of laying out plots automatically and produces a large performance overhead in terms of consistent recalculation to define the location of particles.

8.3.3 Editing Visualizations

Editing aspects of visualizations such as size, and location was a natural process for the user as the plot continuously updated when these actions were performed. Editing the appearance of a plot was defined through selecting the plot and then editing the parameters through the dock interface, using standard swing components such as a J Colour Chooser. This process was easy to use and provided a simple implementation for which plots could be customized. Visually editing the space between axes only worked for parallel coordinate plots, and specifically the first parallel coordinate plot that within the system. The mathematics that define this process need refining so that it works across every parallel coordinate plot. Further features such as axis relabeling could be added. Features such as notes and drawing that were specified in Section 5.5 were not added due to time constraints on the implementation. These features would make a welcome addition to the software, however the software may need to be redesigned to support the nature of these new components.

The performance of editing the appearance of visualizations cannot be evaluated as the process is defined in a sequential manner. The performance of editing the size of the plot will be affected by the interactive frame rate the system maintains whilst rendering the plots. Thus the amount of vertices on screen will affect the interactivity to resizing plots.

8.3.4 Exploratory Visualization

An approach to facilitating exploratory visualization was implemented using smart extraction mode. Extractions also aided exploratory visualization through drilling down into specific selections of data. Data table searching worked well in that users found benefit in visually searching for values from a brushed selection. The statistics panel was of benefit to users as average, maximum, minimum, and standard deviation values were calculated automatically. Finally synchronous plot searching aided the exploratory visualization of finding likewise values in other plots on the canvas. Users agreed that whilst the concept of smart extraction mode was good, it needed more work. Synchronous plot searching was the main feature that users found confusing, finding it difficult to find

the route to an answer through using it. This could be improved by adding options in the dock that activates this feature so that users understand what context they are searching for data values within.

Results for the performance of the searching algorithm were defined in Section 7.3.4. Because the algorithm is implemented naively in a linear way the more points that are searched the higher the search takes to complete. For true synchronicity to occur, plots should be searched in parallel. This would involve redefining the current algorithm to a parallel algorithm for the use with a GPU based language such as CUDA or OpenCL.

8.3.5 Holistic Evaluation

The implementation of the extension to Edivis could be considered a success, as all the objectives and aims of the project were met. This is based on the usability of the implemented features within the software, as evident by 73% average system usability from eight users.

The shortcoming of the system is the performance, whilst this was not one of the aims of the project, interactive applications need to sustain interactive frame rates to be used on a wide scale. The implementation only utilizes the cars dataset; a data-parsing tool within the software that accommodates for mappings of separate data types would be an advantage. If separate data types were accommodated for such as temporal and spatial, this would increase the number of visualizations the software could support.

Iterative implementations.

8.4 Summary

The software has been evaluated in terms of how well the put and pickup models were extended, and how well the aims and objective criteria was met in terms of usability and performance. The project was considered a success in terms of extending the implementation to enhance the number of visualizations, and exploratory and editing techniques. This was evident through performing a user study that defined the usability for the software based on the extended implementation. Each extended implementation feature was discussed in terms of its advantages, disadvantages, and performance to identify a criterion for improvement. General pitfalls in the software were also identified with a need for better data parsing, and drawing tools needed to exploit true visualization editing capabilities.

Chapter 9: Conclusion

9.1 General

This project has provided

9.2 Further Work

Optimizing the efficiency of the software would make an improvement to the usability and performance. Through techniques outlined in Section 8.3.2, the software could be optimized to handle large datasets, thus allowing the software to be utilized for relatively large data exploration tasks.

To implement specifications of the Put model such as drawing, the software would need to be redesigned. Redesigning the software may improve efficiency and provide new approaches that better use the put and pickup model. It would be beneficial to keep certain features of this implementation such as the Plot architecture, data model, and the interaction specifications for each visualization structure.

Developing a data parsing tool, and expanding the type of data the software supports would be beneficial. This would naturally expand the number of visualizations that the software can support. It would also be beneficial to implement support for tree and network data structures within the software, to compete with other visualization software packages such as Tableau [38], Sage [39], and the InfoVis toolkit [32].

Bibliography

[1] Stuart K Card, Ben Shneiderman, and Jock D Mackinlay, *Reading in information visualization: Using vision to think*. San Diego: Morgan Kaufmann, 1999.

[2] Daniel A Keim, Florian Mansmann, Jorn Schneidewind, and Hartmut Ziegler, "Challenges in Visual Data Analysis," in *Information Visualization, 2006. IV 2006. Tenth International Conference on*, Konstanz, 2006, pp. 9-16.

[3] Alfred Inselberg and Bernard Dimsdale, "Parallel Co-ordinates: A tool for visualizing multi-dimensional geometry," Department of Computer Science, IBM Scientific Centre, Los Angeles, Technical IEEE, 1985.

[4] Martin Graham and Jessie Kennedy, "Using Curves to Enhance Parallel Coordinate Visualisations," in *Information Visualization, 2003. IV 2003. Proceedings. Seventh International Conference on*, Edinburgh, 2003, pp. 10-16,16-18.

[5] Geoffrey Ellis and Allan Dix, "Enabling Automatic Clutter Reduction in Parallel Coordinate Plots," in *IEEE Transactions on visualization and computer graphics*, vol. 12, Lancaster, 2006, pp. 717-724.

[6] Harri Siirtola and Kari-Jouko Raiha, "Interacting with parallel co-ordinates," *Interacting with Computers*, vol. 6, no. 18, pp. 1278-1309, March 2006.

[7] Helwig Hauser, Florian Ledermann, and Helmut Doleisch, "Angular Brushing of Extended Parallel Coordinates," in *Information Visualization*, Vienna, 2002, pp. 127-130.

[8] Harald Pringer, Robert Kosora, and Helwig Hauser, "Interactive Focus+Context Visualization with Linked 2D/3D Scatterplots," in *Coordinated and Multiple Views in Exploratory Visualization*, Vienna, 2004, pp. 49-60.

[9] Qingguang Cui, Matthew O Ward, and Elke A Rundensteiner, "Enhancing Scatterplot Matrices for Data with Ordering or Spatial Attributes," in *Visualization and Data Analysis, Part of IS&T/SPIE Symposium on Electronic Imaging*, Worcester, 2006, pp. 0R1 - 0R11.

[10] Niklas Elmqvist, Pierre Dragicevic, and Jean-Daniel Fekete, "Rolling the Dice: Multidimensional Visual Exploration using Scatterplot Matrix Navigation," *IEEE Transactions*, vol. 14, no. 6, pp. 1539-1148, June 2008.

[11] Richard A Becker and William S Cleverland, "Brushing Scatterplots," in *Technometrics*, vol. 29 Issue 2, NJ, 1987, pp. 127-142.

[12] Matthew O Ward and Allen R Martin, "High dimensional brushing for exploration of multidimensional data," in *VIS '95 Proceedings of the 6th conference on Visualization '95*, Washington, 1995, p. 271.

[13] Hong Chen, "Compound Brushing Explained," in *Information Visualization*, vol. 3, Cary, 2004, pp. 96-108.

[14] Jonathan C Roberts and Michael A.E Wright, "Towards Ubiquitous Brushing for Information Visualization," in *Information Visualization, Tenth International Conference on*, Kent, 2006, pp. 151-156.

[15] Ben Shneiderman, "Shneiderman, Ben. "Dynamic queries for visual information seeking," *Software, IEEE*, vol. 11, no. 6, pp. 70-77, 1994.

[16] Michelle Q. Wang Baldonado, Allison Woodruff, and Allan Kuchinsky, "Guidelines for Using Multiple Views in Information Visualization ," in *Proceedings of the working conference on Advanced visual interfaces*, Pale Alto, 2000, pp. 110-119.

[17] Jonathan C Roberts, "Exploratory Visualization with Multiple Linked Views," in *Exploring Geovisualization*, Kent, 2004, pp. 141-170.

[18] Chris North and Ben Shneiderman, "Snap-Together Visualization: A User Interface for Coordinating Visualizations via Relational Schemata," in *Proceedings of the working conference on Advanced visual interface*, Chicago, 2000, pp. 128-135.

[19] James T Thomas and Kristen A Cook, "A visual analytics agenda," *Computer Graphics and Applications*, vol. 26, no. 1, pp. 10-13, 2006.

[20] Daniel Keim et al., "Visual Analytics: Definition, Process, and Challenges," *Lecture notes in computer science*, vol. 4950, no. 1, pp. 154-175, 2008.

[21] Ben Schneiderman, "The Eyes Have It: A Task by Data Taxonomy for Information Visualization," in *Visual Languages, 1996. Proceedings*, Maryland, 1996, pp. 336-343.

[22] Alexander, Williams, Sean Mamishev, *Technical Writing for Teams: The STREAM Tools Handbook*, John Wiley & Sons. Inc, Ed. Hoboken: Institute of Electrical and Electronics Engineers, 2009.

[23] Ben Shneiderman, "Creating creativity: user interfaces for supporting inno- vation," *Computer Human Interaction*, vol. 7, no. 1, pp. 114-138, March 2000.

[24] Thibaut Weise, Sofien Bouaziz, Hao Li, and Mark Pauly, "Realtime Performance Based Facial Animation," in *Siggraphh*, 2011, pp. 1-9.

[25] Trygve Reenskaug, "Models-Views-Controllers," *Technical Note, Xerox PARC*, vol. 32, no. 55, January 1979.

[26] Jacques Bertin, *Graphics and Graphic Information Processing*, Jacques Bertin, Ed. New York: Walter de Gruyter, 1981.

[27] Ed Huai-hsin Chi and John T Riedl, "An Operator Interaction Framework for Visualization Systems," in *Information Visualization, 1998. Proceedings. IEEE Symposium on*, Minnesota, 1998, pp. 63-70.

[28] Stephane Conversy et al., "Conversy, Stéphane, et al. "Improving modularity of interactive software with the MDPC architecture." Engineering Interactive Systems (2008): 321-338.," , vol. 4940, Toulouse, 2008, pp. 321-338.

[29] Stephane Conversy, "Improving Usability of Interactive Graphics Specification and Implementation with Picking Views and Inverse Transformations," in *Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on.*, Toulouse, 2011, pp. 153-160.

[30] T.J Jankun-Kelly, Kwan-Liu Ma, and Michael Gertz, *A Model for the Visualization Exploration Process*, Robert Moorhead, Markus Gross, and Kenneth I Joy, Eds. Boston: IEEE, 2002.

[31] T.J Jankun-Kelly, Kwan-Liu Ma, and Michael Gertz, "A Model and Framework for Visualization Exploration," in *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, Mississippi, 2007, pp. 357-369.

[32] Jean-Daniel Fekete, "The InfoVis Toolkit," in *Information Visualization, 2004. INFOVIS 2004. IEEE Symposium on*, Austin, 2005, pp. 167-174.

[33] Stuart K Card, Jeffrey Heer, and James A Landay, "Prefuse: a toolkit for interactive information visualization," in *Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, 2005.*, New York, 2005, pp. 421-430.

[34] Michael Bostock and Jeffrey Heer, "Protovis: A Graphical Toolkit for Visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 15, no. 6, pp. 1121-1128, December 2009.

[35] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer, "D3: Data Driven Documents," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 12, December 2011.

[36] Di Yang, Zaixian Xie, Elke A Rundensteiner, and Matthew O Ward, "Managing Discoveries in The Visual Analytics Process," *ACM SIGKDD Explorations Newsletter*, vol. 9, no. 2, pp. 22-29, 2007.

[37] Chris Stolte, Diane L Tang, and Patrick Hanrahan, "Computer systems and methods for the query and visualization of multidimensional database," 7089266, August 8, 2006.

[38] Jock D Mackinlay, Pat Hanrahan, and Chris Stolte, "Show Me: Automatic Presentation for Visual Analysis," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 13, no. 6, pp. 1137-1144, August 2007.

[39] Steven F Roth, John Kolojejchick, Joe Mattis, and Mei C Chuah, "SageTools: An Intelligent Environment for Sketching, Browsing, and Customizing Data-Graphics," in *Conference companion on Human factors in computing systems*, New York, 1995, pp. 409-410.

[40] Steven F Roth et al., "Visage: A User Interface Environment for Exploring Information," in *Information Visualization'96, Proceedings IEEE Symposium on*, San Francisco, 1996, pp. 3-12.

[41] Mei C Chuah, Steven F Roth, Joe Mattis, and John Kolojejchick, "SDM: Selective Dynamic Manipulation of Visualizations," in *Proceedings of the 8th annual ACM symposium on User interface and software technology*, New York, 1995, pp. 61-70.

[42] Steven Feiner and Clifford Beshers, "Worlds within Worlds Metaphors for Exploring n-Dimensional Virtual Worlds ," in *Proceedings of the 3rd annual ACM SIGGRAPH symposium on User interface software and technology*, New York, 1990, pp. 76-83.

[43] Jarry H.T Claessen and Jark J van Wijik, "Flexible Linked Axes for Multivariate Data Visualization," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 17, no. 12, pp. 2310-2316, Dec. 2011.

[44] Pat Hanrahan, "VizQL: a language for query, analysis and visualization," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, New York, 2006, pp. 721-721.

[45] Jacques Bertin, *Semiology of graphics: diagrams, networks, maps*.: University of Wisconsin press, 1983.

[46] Jonathan C Roberts, "The Five Design-Sheet (FdS) approach for Sketching Information Visualization Designs," in *Eurographics 2011-Education Papers*, Llandudno, 2011, pp. 27-41.

[47] John Brooke, "SUS - A quick and dirty usability scale," in *Usability evaluation in industry*, 1996, pp. 189-194.