Asynchronous Collaborative Coordinated Visualisation

Charlotte Cuddy
B.A.I. Engineering
Final Year Project April 2014
Supervisor: Professor Owen Conlan

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland
Declaration

I, the undersigned, declare that this work has not been previously submitted as an exercise for a degree at this or any other University, and that, unless otherwise stated, it is entirely my own work.

____________________________________
Charlotte Cuddy
Permission to lend or copy

I, the undersigned, agree that the Trinity College Library may lend or copy this thesis upon request.

____________________________________
Charlotte Cuddy
Acknowledgements

I would like to thank my supervisor Owen Conlan for his continued help and guidance throughout the year.

I would also like to thank my family for their continued support and encouragement.
Table of Contents

Declaration.................................................................................................................. i
Permission to lend or copy......................................................................................... ii
Acknowledgements................................................................................................. iii
Table of Contents ...................................................................................................... iv
Table of Figures ......................................................................................................... vi
1 Introduction ........................................................................................................... 1
   1.1 Motivation ......................................................................................................... 1
   1.2 Research Question ......................................................................................... 5
   1.3 Objectives and Goals ..................................................................................... 6
   1.4 Overview ......................................................................................................... 8
3 Related Technologies ............................................................................................ 14
   3.1 Client-side ...................................................................................................... 14
      3.1.1 Baseline Technology ............................................................................. 14
      3.1.2 Visualisation Libraries ........................................................................ 15
      3.1.3 Communication .................................................................................. 18
   3.2 Server-side ..................................................................................................... 19
4 Design .................................................................................................................. 22
   4.1 Goals and Objectives .................................................................................... 22
   4.2 User Flow ...................................................................................................... 24
   4.3 The Coordinated Visualisation System ....................................................... 26
   4.4 User Tracking Methods .............................................................................. 27
   4.5 Storing Interactions .................................................................................... 29
   4.6 Linking Technologies .................................................................................... 31
5 Implementation .................................................................................................... 33
   5.1 Coordinated Visualisation System Modifications ....................................... 33
      5.1.1 HTML & CSS ...................................................................................... 33
      5.1.2 DC.js .................................................................................................. 35
   5.2 The Database ................................................................................................. 36
   5.3 Implementing the User Flow ........................................................................ 37
      5.3.1 Triggers by Class ................................................................................. 38
      5.3.2 Triggers by Type .................................................................................. 39
      5.3.3 Triggers by ID ...................................................................................... 40
   5.4 The “Next” Button ...................................................................................... 41
      5.4.1 Stepping through Replay .................................................................... 41
      5.4.1 Resuming Replay ................................................................................. 43
   5.5 Client-Server Interaction .............................................................................. 44
      5.5.1 Table of Replay Tables ....................................................................... 45
      5.5.2 Table of Current Filters ....................................................................... 47
      5.5.3 Replay Tables ...................................................................................... 48
   5.6 User Example ................................................................................................. 49
      5.6.1 Creating the Exploration Path ............................................................... 49
      5.6.2 Replaying the Exploration Path ........................................................... 51
      5.6.3 Pausing the Exploration Path ............................................................... 53
5.6.4 Cloning the Exploration Path................................................................. 53
6 Evaluation ............................................................................................................. 55
   6.1 Evaluation method ......................................................................................... 55
   6.2 Observations ................................................................................................. 56
7 Conclusion ........................................................................................................... 59
   7.1 Future Works .................................................................................................. 60
Table of Figures

Figure 4.1, User flow for the framework.................................................................24

Figure 4.2, Sample Dimensional Charting coordinated visualisation.......................27

Figure 5.1, Additional elements in the user interface.............................................34

Figure 5.2, checkIfExists function with AJAX request using GET method.............45

Figure 5.3, Anonymous function identified through the HTTP method and resource identifier using Slim Application.................................................................46
1 Introduction

1.1 Motivation

Coordinated visualisation systems are a powerful tool for data analytics. A coordinated visualisation is a collection of data representations, graphs and other visualisations, which uses one set of data for all visualisations. Where data elements are made up of a number of variables, the data set can be represented in many ways, depending on which variables the visualisation is based on. Visualisations can also be based on data derived from variables in the data elements.

For example, consider a data set consisting of data for all the property sales in Ireland over a certain period. This data set would include variables like the date, the revenue from the sale, the location. Many visualisations can be formatted from these three variables and data derived from these variable. Either the number of sales or the revenue from sales could be illustrated by day, by month, by year, by day of the week or by location. They could also be illustrated per day by location, per month by location and many more.

Where the power of coordinated visualisation systems comes in is that users can interact with the visualisations so as to only view a subset of the total data set. Any change in the subset of data is propagated through all of the visualisations. In the property example, if the user reduced the data set to only contain data from 2013 in a number of sales per year visualisation, because all of the visualisations are linked, the change would propagate through all of the visualisations so that they would also only represent data for 2013. If there was a sales revenue per month bar chart, the October bar would only refer to sales revenue in October 2013. This eliminates the need for producing visualisations for every scenario. Instead, to identify any trends in the property sector within a certain period in a given location, national estate agents can
use a coordinated visualisation with all of their data and restrict it for the year and location as needs be.

Large sets of raw data can be extremely complex to analyse. Through linking different visualisations of a data set, analysts can explore the behaviours of various subsets of that data. This can enable them to gain a better understanding of the data in order to draw conclusions about that data. Researchers can use coordinated visualisation to illustrate known relationships between variables so as to corroborate their research. During the course of the exploration of data through coordinated visualisations, unforeseen connections between variables may be stumbled upon.

While coordinated visualisations facilitate the exploration of data sets, for large complex data sets the exploration paths can be long and meandering. On identifying a relationship of interest, possibly requiring further investigation, it may be difficult to retrace the steps that led to the identification of this relationship. And should the exploration path be continued, it may be impossible to find the way back to an earlier point of interest. For this reason, the ability to track exploration paths through coordinated visualisations and revisit a tracked path would greatly improve the efficacy of coordinated visualisation systems as a tool for data analytics.

As well as assisting in the exploration and analysis of data by individuals, tracking explorations would enable the sharing of expertise. This may be the sharing of expertise between a number of experts in a field, or it may be an expert sharing their expertise so as to guide a novice through a complex topic, helping them gain an understanding of otherwise overwhelmingly complex data.

With the property example, an expert in property analysing the property bubble and ensuing crash in Ireland could be looking at the trends in the property market and in other sectors of the national market within a visualisation. They might come across an
unforeseen common movement between the property market and another sector which they may find worthy of further investigation. Annotating their exploration path that led to the discovery and sharing it with an expert in the other sector would facilitate the explanation of how the discovery was made and why it might be of interest. The other expert could then contribute their expertise on the happenings in their sector along the exploration path. Or an expert could provide a guide on how to determine whether it is a good time to buy property in Ireland through an exploration path for people who know nothing about real estate.

Currently this is feasible through screen capturing, either by taking screen shots at each step or by producing screencasts. Much software has been developed to produce screencasts with editing options to add audio narration and/or visual annotations. While useful and informative, following a video recording of an exploration path has major limitations. In order to get back to a certain point in the exploration path, the user needs to manually mirror the actions taken on the coordinated visualisation while watching the screencast. Should the user wish to deviate momentarily from the recorded path, deviating too far may result in the user needing to restart the path from the beginning in order to continue to the end of the exploration path. At an intermediary point of a recorded path, if the user wishes to produce a screencast to that point with an alternative exploration path appended to it, they may need to restart the path from the beginning, recording the path as they proceed through again.

In the property example, an analyst may be following and recreating an exploration path that looks at the change in property sales through the year by county as per a screencast. At some point they may decide to pause the screencast to compare a county to an adjoining county or a county of similar size or economic growth. In their reverie they may lose track of how to return to the point where they left off. Where there are only a few visualisations, this may not cause a problem; it may be easy to recall the scenario they paused the screencast at, or to reapply the necessary filters from looking
at the screen. However when there are many visualisations, it can be quite difficult to keep track of everything, and all of the visualisations may not be clearly visible in the screencast at one time. It may take some backtracking to get back to the point where they paused the screencast. And if they discovered a point of interest during their side exploration, they will need to restart the exploration path to record it, and may not be able to recreate the exploration that led to that point of interest with ease. With complex data and extensive exploration paths, these scenarios occur commonly and can be quite tedious to implement.

Another shortcoming of screencasts is that the exploration path is recorded in real time. Different viewers of a screencast may wish to spend varying periods of time at each step of the exploration path, most likely unequal to that spent during the recording of the path. This can be overcome by pausing while recording, editing before sharing with others or pausing and fast-forwarding during replay; however these solutions can be quite labour intensive.

In order to develop data visualisation into a truly powerful tool, a method of tracking user interactions with a coordinated visualisation system is needed which allows the author of the exploration path to

- comment on the exploration path
- replay the exploration path while making additional commentary
- share the exploration path with others, be it experts or novices

The experts or novices with which the path is shared should then be able to

- step through the path at their own pace
- add their own commentary to enable collaboration
- deviate from the existing path, whether it be momentarily or in order to create a new path.

This method should be simple and should require minimal effort beyond interactions with the coordinated visualisation system and the inclusion of annotations.
1.2 Research Question

This project proposes to develop a framework which allows the sharing of expertise through asynchronous guidance/collaboration over coordinated visualisation. It outlines the design and implementation of a system in which user interaction with coordinated visualisation systems can be recorded and annotated for replay at any future time for use in guidance or collaborative work.

Coordinated visualisation systems consist of various linked visualisations of one set of data. Users can interact with the individual visualisations to burrow down into a subset of the data. Since the different visualisations are linked through the data set, changes to one visualisation are propagated through to the others. This allows users to examine subsets of the data, removing some parameters which veil relationships between other parameters in the data set.

During collaborative research, experts will undertake research individually and share their findings with their collaborators. When the topic involves complex data sets, coordinated visualisation systems can allow the experts to visually illustrate their arguments. Exploring different paths in coordinated visualisations may lead to the discovery of unanticipated relationships. On discovering such a relationship, the expert may wish to share with their collaborators the discovery together with the chain of interactions and associated thoughts that led to the discovery. By tracking user interactions and facilitating annotations on the user interaction records, this would allow collaborators to follow the logic that led to the discovery, providing them with more information on which to base their expert opinion of the discovery.

While coordinated visualisation systems greatly simplify the analysis of data, it may still be difficult to understand, especially for users new to the topic. Experts may be willing to provide some guidance in order to assist novices in gaining a basic understanding. This could be achieved through coordinated visualisation where experts would create an introductory exploration path with annotations providing
detailed explanations and pointing out noteworthy relationships at each step. The novice could then step through the path at their own pace.

By developing a framework which recreates user interactions and annotated exploration paths, collaboration and guidance over coordinate visualisation could be done **asynchronously** since experts and novices could replay the desired path at their own convenience.

### 1.3 Objectives and Goals

The sharing of expertise through asynchronous guidance or collaboration over coordinated visualisation is achievable through screen capture; however there are limitations to using screen capture, the most notable of which are:

- The user must reapply the interactions with the coordinated visualisation as they follow the recording.
- The time spent at each step will vary for different users; this will require the viewer to pause and/or fast-forward at different steps and may require the creator of the exploration path to edit the exploration path, beyond adding commentary, before sharing it with others.
- If the user wishes to deviate from the exploration path momentarily then return to follow the remainder of the path, they must keep track of their deviation in order to retrace their steps or risk having to restart the exploration from the start.
- If the user wishes to record an exploration path extending from the initial part of another exploration path, rather than appending from that point, they may need to restart the exploration while recording. Alternatively, the existing recording could be used and clipped at the required point; however it may be difficult to remove all of the commentary.
- The editing of small sections of the commentary may prove difficult.
• There is no method available for a collaborator to respond to commentary in an exploration path.

In light of this, the project set out to overcome these challenges and produce a system that reapplies the user interactions to the coordinated visualisation while displaying the corresponding annotations. This would allow the user to step through the exploration path at their own pace. For added functionality, the project looked at permitting collaborators to make their own annotations at each step and at facilitating the deviations from a path.

In order to achieve this, the following goals were set:

• Store sufficient information about every user interaction with the coordinated visualisation to be able to recreate the user interaction.
• Replay the exploration path on demand by reapplying the user interactions.
• Facilitate the addition of annotation by the initial expert during the exploration and during the replay.
• Allow others to replay the exploration path, adding their own commentary if required.
• Enable the pausing of the replaying of the exploration path for an alternative exploration with the possibility of resuming where the deviation began.
• Allow viewers to clone the exploration path up to a certain point and append alternative exploration with minimal effort.

For these goals to be met the following objectives had to be realised:

• Analyse and understand the coordinated visualisation system so as to identify the minimum information required to recreate the user interaction and how, using this information, the user interaction could be recreated.
• Design the shell of the system, identifying what technologies are needed and how they should communicate.
• Design the inner working of the system, what should happen during the creation of the exploration path, during the annotation of the exploration path and during the replaying of the exploration path.
• Implement the technology to meet the required goals in line with the chosen designs.

1.4 Overview

This chapter looked at the motivation which led to the proposal of the research question. The proposed research question was stated and the terms used in the question were explained in the context of this project. Based on how the current technology is lacking when it comes to facilitating the sharing of expertise asynchronously over coordinated visualisations, goals were set with which to measure the success of the implementation of the framework. Objectives were then set with which to meet the goals. The remainder of this chapter looks at what is covered each chapter of this report.

Chapter 2 looks at how visualisations have evolved due to the need for tools to analyse the enormous volumes of data available due to new technologies. It looks at what features coordinated visualisations should possess in order to facilitate the data analysis. It finds that the ability to track, annotate, share and guide through coordinated visualisation has been identified as a key feature to facilitate data analysis, but that little progress has been made in this area.

Chapter 3 looks at the various technologies used in the implementation of the system, first on the client side, then on the server side. It looks at the advantages of using these technologies, and where the technologies are lacking.

Chapter 4 outlines the process undertaken to design the framework. A user flow is described to illustrate all the possible user scenarios which the design should account for. The choice of coordinated visualisation, methods and technologies used
in the design of the framework are explained and overall design of the system is described.

Chapter 5 looks at how the design was implemented by outlining the required modifications made to the existing coordinated visualisation system, how exploration paths are stored, how the system deals with user interactions and finally how the client and server communicate.

Chapter 6 evaluates the final system using the coordinated visualisation used to test and develop the system, and a coordinated visualisation on which the framework had not yet been trialled. Using the user flow for guidance, the system is tested on a path that follows all of the possible user process and finds that the system can cope with every scenario. It then outlines some flaws with the system which would need to be addressed if future work were undertaken on the project.

Chapter 7 concludes that the framework was developed as per the goals set out but highlights some features that would need to be implemented for the framework to be marketable.
Background

“...thought is impossible without an image”  (Aristotle, 350 BC)

This chapter looks at how visualisations have evolved with the development of new technologies and the need for visualisations to deal with the explosion in the volume of data available for analysis. It looks at what experts in the field believe is required to create a powerful data analysis tool using coordinated visualisation.

The use of visualisations to portray quantitative information dates back to the 10th century. Over time, techniques and instruments for gathering data evolved and with them visualisation techniques. The 17th century saw the real beginnings of visual thinking while the 18th century saw the development of new graphs and their use over a wider range of areas. The 19th century saw the development of modern graphing techniques and the importance of the analysis of data come to light. It was during this time that statistic offices were established and the first 3-dimensional graphs were developed. While little progress was made in the first half of the 20th century, research and development in the areas of computers and data analysis, both separately and together, led to a boom in the development of data visualisation in the second half of the century [Friendly, 06].

With the introduction of the web and the emerging of new technologies which enable the collection of colossal amounts of data, software has been developed to transform the data into visual forms and visualisation methods and techniques have been evolving continuously at a rapid pace. The study of the human eye has revealed that certain features of visual objects are perceived before the eye has time to react. These include size, shape, colour and direction of motion. In visualisations, these can be used to provide information about the node, or its relationship to other nodes. A change in the size of a node will represent a corresponding change in value and relative sizes will give users an initial indication of the relationship between nodes. Colour scales are used to depict a scale from one extreme to another which can be
understood by users very intuitively. Peaks and troughs in a timeline graph will attract the user’s attention before uniformly shaped regions. Motion, even in the periphery, will attract the user’s attention and is intuitively indicative of the change in the data. These features have therefore been built into computerised visualisations [Diehl, 07].

As the number of dimensions in data elements collected grew, two approaches were taken to assist with the analysis of large multi-dimensional data sets – multi-dimensional visualisations and multi-view visualisations. Multi-dimensional visualisations incorporate all of the dimensions in one visualisation. These were found to be difficult to generalise for all data sets and for users to comprehend and navigate intuitively [Schneiderman, 96]. Instead there has been a shift to multi-view visualisations where different dimensions of the data set are displayed side by side in a number of views.

Ben Schneiderman examined various visualisation designs and summarized the basic principle into what he named the Visual Information Seeking Mantra [Schneiderman, 96]:

“Overview first, zoom and filter, then details-on-demand”

That is, users should be given an overview of the entire collection of visualisations. They should then be able to zoom into a particular visualisation and select subsets of interest. Finally they should be able to get details on remaining data as required, for example by clicking on the node. Schneiderman proposed 3 further steps to make data exploration a more “joyous experience” – relate, history, extract. Relate refers to the relationship between attributes. Schneiderman proposed that once the user has selected a subset of a particular attribute, they should be able to see how other attributes of the dataset are affected by the reduction to that subset. Today, this is achieved by linking visualisations so that as the user filters down into a visualisation, all other visualisations are updated to only include that subset of data. History refers to the ability to track user interactions so as to replay or refine them. Extract refers to the ability to save a particular state of interest to share with others.
Schneiderman suggested that these features would greatly assist in the exploration and analysis of large data sets.

With data sets becoming too big for an individual to explore thoroughly, and differences in the way data is interpreted by individuals resulting in the need for discussions and debates, Heer et al. proposed that information visualisation was more of a social, than an individual, activity. [Heer et al., 07]. They developed a website sense.us allowing users to annotate and share views and discuss findings. Rather than keeping a full history of exploration paths, users could bookmark views of interest and begin a discussion on the view to which others could contribute. Bookmarked views could also be used as starting points for further explorations. The usage of the site was studied and they found that users benefited from asynchronous collaboration, exploring the data set more widely and in depth than if they had been exploring it individually. However, while research supports the fact that data visualisation is a social process requiring the sharing and discussion of findings, little progress has been made in developing systems that allow the retracing of steps through an exploration path, the annotation of that exploration path and the sharing of the annotated exploration path with the possibility of making further annotations for collaboration.

Heer and Schneiderman combined their research to provide a detailed description of what visualisations should consist of [Heer and Schneiderman, 12]. The development of visualisations was split into three sections – Data & View Specification, View Manipulation and Process & Provenance. Data & View Specification looks at how visualisations should be created, different filtering methods such as selection or the use of sliders, sorting nodes to identify patterns and the use of derived values as well as data values. View Manipulation looks at the manipulation of visualisations, Schneiderman's Visual Information Seeking Mantra and linking the visualisations so that filters are propagated through. While many techniques have been developed for these two sections, research continues to develop new techniques to provide further assistance in the exploration of vast
amounts of data. The final section is Process & Provenance. This is the area that the least progress has been made in, and the section that this project focuses on. Process & Provenance is split into four parts – Record, Annotate, Share and Guide. Under “Record” Heer and Schneiderman look at the need for a method of recording exploratory paths so that they can be revisited, cultivated and shared. They outline the importance of this feature as being due to the fact that exploration will lead analysts down different paths to a number of different findings and ensuing questions and hypotheses which they may want to revisit and explore further along a different path. Under “Annotate” they look at the benefits from having a method for documenting the findings and ensuing questions and hypotheses. As well as for an analyst’s own use, the use of annotations can facilitate the sharing of findings with collaborators. Under “Share” they emphasise the fact that data analysis is rarely a task undertaken by a single individual, but rather one which requires collaboration and discussions with colleagues, comparing individuals’ findings and hypotheses. They highlight the importance of incorporating a means of sharing findings, be it simply in the form of extracting the view in some format such as jpeg or png, or the subset of data in some format such as csv or json. Under “Guide”, they outline the potential for using recorded exploration paths as a guide for novices, either as instructions on how to analyse data or to tell a story through the analysis of data.

It is clear that due to the volume of data available for analysis, without powerful tools to assist with data analysis, it is not possible for individuals to explore the data fully. Despite much research and development in coordinated visualisation systems, there is a lack of tools available to facilitate the sharing of expertise over these systems. However, the importance of being able to track, annotate, share and guide using coordinated visualisation has been identified as a key feature for developing powerful data analysis tools.
3 Related Technologies

In order to develop a framework which allows the sharing of expertise through asynchronous guidance and/or collaboration over coordinated visualisation, a number of technologies are required to communicate with each other. The framework was developed using a Client-Stateless-server architecture so as to separate the user interface and the data storage components of the system while keeping all of the session state on the client side, not on the server-side. This chapter first looks at the technologies employed on the client side, starting with baseline technologies common to most web applications, then looking at technologies that make up the coordinated visualisation system used and finally looking at the technologies on the client side used to communicate with the server. The server side is then examined, looking at what is meant by a “stateless-server” and how this is achieved, then looking at the technologies used to develop and test the server-side and store the data.

3.1 Client-side

3.1.1 Baseline Technology
The web page in which the framework is set up is created using HyperText Markup Language (HTML). HTML is the most commonly used language for creating web pages. The structure and content of the page is built up using HTML elements. These elements can be have various attributes which can dictate their type, function or style. They can be assigned an identification tag or a class to which certain methods can be assigned or with which interaction with the element can be monitored. The framework takes an existing coordinated visualisation system and adds some elements to the HTML document with which to add the annotation and replay functionalities [HTML].
The formatting and layout of a webpage can be done within the HTML document, or the HTML document can be linked to one or more Cascading Style Sheets (CSS). In this framework, a number of CSS files are used, one specifically for the additional features of the framework, the rest from the existing coordinated visualisation system. In the CSS, the size of elements, their location relative to each other and their location within the page can be set out [CSS].

HTML provides the building blocks of the web page and the CSS styles it, but it is the JavaScript files that do the “interesting stuff”. Any JavaScript file which the HTML document links to is only compiled once the web page has been loaded; that is once all of the elements in the HTML document have been created. JavaScript is an object-oriented language used to handle the background activity of elements defined in the HTML file, such as the customisation and manipulation of the elements or computations related to the elements. Values and functions can be added to, and actions can be triggered on interaction with, HTML elements using JavaScript [ECMA]. In this project, a JavaScript file is used to tie together all of the client-side operations. It links the HTML and CSS files to the JavaScript APIs, jQuery event triggers and AJAX requests.

### 3.1.2 Visualisation Libraries

Three JavaScript APIs are employed in the coordinated visualisation system used in this project:

- D3.js
- Crossfilter.js
- DC.js

The Data Driven Documents JavaScript library (d3.js) combines HTML, CSS and Scaled Vector Graphics (SVG) to create almost any visualisation imaginable [D3]. Within a HTML document, SVGs objects are created and styled initially through CSS. Using the d3 library, the charts can then be set up so that users can interact with the visualisations. These visualisations can be set up in such a way that, on the selection
of nodes, the formatting will change or if the node is a parent in a hierarchy, users can delve deeper into an element. Transitions can be used to apply any changes gradually for added animation. The data can be bound to the SVG so that for example the radius of a node in a bubble chart can be determined by the value of some data for that node. D3.js is a powerful library in the sense that it can produce an extensive variety of creative visualisation. However, a weakness with the d3.js library is that it requires that the data be saved in specific formats in order to filter down into a particular dimension. This means that when using multiple-views, the data would need to be manually formatted as required for each graph in the coordinated visualisation system. Filtering down into one visualisation would only change the data set displayed for that visualisation; the change would not be propagated through to other visualisations in the coordinated system. This issue can be overcome using the Crossfilter API.

The Crossfilter JavaScript Library can be used to apply filters to large multi-dimensional data sets [Crossfilter]. When a user interacts with a coordinated visualisation system, this api can perform the required operations at high speeds, even for datasets with record numbers in excess of 1 million. Within the Crossfilter API, the crossfilter refers to the full set of data; that is a multi-dimensional dataset, usually made up of an array of JavaScript objects, in our case passed in form a csv file. The Dimension specifies what aspect of the crossfilter data the user is interested in. It can be data stored in the JavaScript objects such as counties or years, or data derived from the data stored in the JavaScript objects such as quarters or gain and loss. The Filters which can be applied to the dimensions are what that dimension can be sliced into. For example for months of the year, the filters would be the 12 months of the year; for annual quarters, the filters would be quarters 1 to 4, and analysts could look at the whole year, at Q1 alone, at Q1 and Q2 simultaneously, or any other combination of the four quarters. In order to determine the number of records in each dimension when filters are applied, the dimensions are grouped and the number of elements present is returned to be used in the rendering of the visualisation. The issue of formatting the data files which arises when using the D3
library is overcome using the Crossfilter library in the way the dimensions are defined.

The Dimensional Charting JavaScript Library (dc.js) combines the D3 and Crossfilter libraries to build coordinated visualisations [DC]. Chart objects are created linking the HTML elements representing the charts to the DC chart functions. These functions assign various properties and functions to the charts depending on the type of DC chart. Different types of charts are handled differently when filters are applied depending on whether an array of filters or a range is applied, whether the radius of nodes or the height and width of bars needs to be recalculated. The charts are created within a chart group so that any interaction with one of the charts in that group will be propagated onto the other charts within that group. A crossfilter is constructed with the full dataset and the required dimensions for the various graphs are defined. The D3 chart attributes are defined as required, using the relevant dimensions as data input. When a filter is applied to a chart, the DC API calls on the Crossfilter API to recalculate the number of elements in each remaining filter of each dimension. It then calls on the relevant D3 methods for each chart to recalculate the data bound parameters of the charts and redraw all of the charts. Therefore, as a user interacts with the visualisations, any filter applied to one visualisation is propagated through to all of the other visualisations within that group.

This makes the DC API a powerful for data exploration in terms of creating coordinated visualisation systems. However it lacks features that support collaboration. Many analysts can access the coordinated visualisation at any time; however they will only have access to their interactions and will not get any insight in others’ interaction with the visualisations. It lacks the ability to track user interactions, to annotate and replay interactions and to share an exploration with another user so that they may replay the exploration and add annotations for collaborative work.
3.1.3 Communication

Additional features are added to the coordinated visualisation system in order to create a framework suitable for asynchronous guidance and collaboration. Data is collected from the client side during user interactions and stored in the server side. It is then retrieved when it is needed. To achieve this, two more technologies are required on the client-side – jQuery and AJAX. As the user interacts with the visualisation, jQuery event handlers are triggered, executing a number of sequential functions, one or more of which contain AJAX requests which send data to and retrieve data from the server.

jQuery is a JavaScript library that allows event handling [jQuery]. As the user interacts with different HTML elements on the web page, different event handlers are triggered. This can be done by element id or by class to which a group of elements belong. For example, a button to save an annotation would require its individual event handler, while a group of charts belonging to the same coordinated visualisation system would require an event handler for elements in that group, or class. Where the event handler is defined for a class of elements, the id of the element with which the interaction occurred can be extracted using jQuery functions before calling further functions containing AJAX requests.

Asynchronous JavaScript and XML (AJAX) requests are a part of the jQuery API [AJAX]. They allow HTTP requests to be performed asynchronously. This means that rather than the page being reloaded each time a HTTP request is performed, the relevant features are updated as data is retrieved from the server. The HTTP method and URI of the relevant resource are provided in the AJAX request to the server. The most commonly used HTTP methods [HTTP] are:

- GET to retrieve data from the server
- PUT to update data on the server
- POST to send data to the server
- DELETE to delete data from the server
While the name suggests the sending and retrieving of Extensible Markup Language (XML) data, JavaScript Object Notation (JSON) is more commonly used.

JSON are made up of key/value pairs [JSON]. Before sending data to the server, the data object made up of a collection of key and value pairs must be converted to JSON using the “stringify” function. On retrieving data from the server, the JSON must be converted to the data object made up of a collection of key and value pairs using the “parse” function.

### 3.2 Server-side

PHP: Hypertext Preprocessor (PHP) is a scripting language used for code which is executed on the server [PHP]. Test servers are commonly used to execute PHP code so as to develop and test web applications before uploading them to the actual server. AMP servers facilitate the creation of database driven applications using an Apache HTTP server, a MySQL database and PHP [WAMP].

The client communicates with the server via a Representational State Transfer (REST) API. REST APIs are stateless; that is no session state is saved on the server [Fielding, 00]. Any data required to perform a request must be received from the client with the request. Any data which is to be sent to the client must be retrieved from some storage space, in this case a database. This allows each request to be dealt with independently, partial failures to be dealt with without terminating the session and multiple clients to operate with the same server.

The Slim Framework is the REST API used in this project. The Slim Application is defined in a PHP file. In this file, various routes made up of a combination of a resource plus a method lead to the appropriate anonymous function to be executed. In order to get to the required Slim application route, the URI from the AJAX request must be rerouted to this PHP file. This is done using a .htaccess file. Each time there is a HTTP request for a non-existent file, the request is rerouted to the PHP file.
where the Slim application route can be found. The URI from the AJAX request is made up of the location of the PHP file with the resource name appended to it. The Slim Application takes the part of the URI identifying the resource and the HTTP method which describes what is to be done with the resource from the AJAX request, then uses the route that matches the URI resource and method to determine which anonymous function should be executed.

The anonymous functions within the various Slim application routes make up the server side of the framework. Taking any data received from the client side, server actions are performed and data is returned to the client’s callback function which instantiated the AJAX request as required. Generally, data can be saved on the server; however when using a REST API, no data is saved on the server due to the stateless property of the REST API. Data received from the client must decoded before the server actions can be performed. Server actions may include querying a database to store, update, retrieve or delete information. Before sending any required data to the client it must be encoded into JSON format.

PHP Data Objects (PDO) are used to access databases for added security. Prepared statements and data objects replace standard SQL queries. Prepared statements are precompiled SQL statements. Once a statement has been prepared, each time data is sent to the server, the prepared statement is executed with the relevant data in the required placeholders. This protect from SQL injections attacks. With PDO it is possible to specify how the data from the query should be returned. For example, FETCH_OBJ will return records as objects with the database attribute names as the properties of the object, while FETCH_ASSOC will return records as arrays where the array index represents the column number.

Some of the benefits of using a database to store information are that sets of data can be stored in different tables and retrieved in later sessions. The database can be queried to retrieve any subset of the data required, to update records and to append to sets of records interchangeably. The queries can be performed quickly so that
actions appear instantaneous to the user. MySQL is currently the most popular open source database which is used for web applications. One reason for this is that it was designed and developed specifically for web applications. It is used by companies such as Google and Facebook for storage and fast retrieval of the large volumes of data [Oracle, 13].

This chapter gave a brief overview of the various technologies used in this project to develop a framework that allows the sharing of expertise through asynchronous guidance and/or collaboration over coordinated visualisation. Further insight into how these technologies interact and what features in particular are exploited to realise the framework will be provided in the Design and Implementation chapters where required.
4 Design

Coordinated Visualisation Systems are unarguably a powerful tool for data analytics, but their power could be magnified by incorporating a method to track, replay and share exploration paths for collaborative purposes or to act as a guide for novices. This project looks at developing such a framework using the technologies introduced in Chapter 3. In this chapter, the process undertaken to design this framework is outlined. The goals and objectives which the design must meet are first examined. A flow of the different possible user scenarios when using the system is described. The choice of coordinated visualisation system and methods and technologies used in the design of this framework are explained. Finally the overall design of the system is outlined.

4.1 Goals and Objectives

In order to incorporate asynchronous collaboration and guidance into a coordinated visualisation system, the key features which the user would want to have available to them were identified.

One of the main features of this system is that users should be able to replay exploration paths. While the replay could be set to automatically step through the exploration path with the same time gaps between steps as during the exploration, it was decided that users may not want to spend the same amount of time at each step and should therefore be able to step through at their own pace.

Users will want to annotate their exploration paths, be it to document their train of thought, to propose a hypothesis to a colleague, or to guide a novice. Collaborators will wish to add their own annotations so as to respond to their colleague's comments.
Coordinated visualisations can unveil interesting relationships or highlight data anomalies which users will want to investigate further by exploring various paths. It can therefore be expected that users will want to deviate from the exploration path being replayed. They may wish to do so briefly with the intention of returning to the point at which they deviated, or they may wish to create a new exploration path, cloning the exploration path being replayed up to that point and then appending to it.

With these user requirements in mind, the following goals were set:

- Store sufficient information about every user interaction with the coordinated visualisation to be able to recreate the user interaction.
- Replay the exploration path on demand by reapplying the user interactions.
- Facilitate the addition of annotation by the initial expert during the exploration and during the replay.
- Allow others to replay the exploration path, adding their own commentary if required.
- Enable the pausing of the replaying of the exploration path for an alternative exploration with the possibility of resuming where the deviation began.
- Allow viewers to clone the exploration path up to a certain point and append alternative exploration with minimal effort.

For these goals to be met the following objectives had to be realised:

- Analyse and understand the coordinated visualisation system so as to identify the minimum information required to recreate the user interaction and how, using this information, the user interaction could be recreated.
- Design the shell of the system, identifying what technologies are needed and how they should communicate.
- Design the inner working of the system, what should happen during the creation of the exploration path, during the annotation of the exploration path and during the replaying of the exploration path.
• Implement the technology to meet the required goals in line with the chosen designs.

4.2 User Flow
Using the goals and objectives, a user flow was designed to ensure that all of the possible scenarios the user might encounter would be considered. From the user flow, additional considerations for the design of the system were highlighted. Figure 4.1 illustrates the user flow for a system that allows asynchronous collaboration and guidance using coordinated visualisations.

\[\text{Figure 4.1: User flow for the framework.}\]

Users may follow different exploration paths. For it to be possible to revisit different exploration paths, these should be saved under different names. Saved exploration paths are referred to as “Replays” in this project. Users should be able to choose which Replay they wish to revisit, or have the possibility of creating a new Replay.

When a user enters a Replay title when prompted by the system, the system should check whether the replay already exists. If the Replay does not exist, users should be
able to explore the data with the system logging each step of their exploration path. The blue nodes in the user flow are steps where the user may want to annotate the exploration path. The addition of annotations should be possible during the initial exploration as users may want to document their train of thought as they explore the data.

The user should be able to continue to explore the data, appending to the steps logged by the system. At any point, they should be able to Replay their exploration path from the start, adding or editing annotations along the way. When in Replay mode, users may wish to step through to the end of the Replay. Once they reach the end of the stored Replay, they may wish to

- Continue the exploration of the data, appending to the Replay
- Restart the Replay
- Start a new Replay or explore a different existing Replay

Alternatively, users may wish to deviate from the Replay rather than step through to the end. If so, they may wish to pause the Replay so as to return to the point at which they left off after they have explored some alternative path. It is therefore necessary to implement a method of resuming the replay. Once the user decides to resume the replay, they can step through the remaining steps of the Replay, either continuing to the end or deviating from the path again. Rather than pausing the Replay, users may wish to create a new Replay by cloning the replay up to that point and appending a different exploration path from that point on.

In the case where the user chooses to open a Replay which already exists, the Replay should be opened from the start so that the user can step through the Replay. As outlined above, they may choose to step through to the end of the Replay and append to the exploration path, restart the Replay, create a new Replay or explore a different existing Replay. Or they may wish to deviate from the exploration path,
either pausing the Replay and resuming it at that point later, or cloning the Replay to that point and appending to the new Replay.

While the goals clearly state that users should be able to replay the exploration path and share it with others, the user flow highlights the fact that the system should be able to store multiple exploration paths so that users can change the exploration path they are working with so as to revisit and append to different exploration paths. The system should also be able to verify whether an exploration path with that name exists.

4.3  The Coordinated Visualisation System
With the explosion in the volume of data which is being collected thanks to new technologies, much research has been undertaken to develop coordinated visualisation systems. As a result, there are many different systems available, some very specific to certain data sets, some adaptable for any data set.

Coordinated Visualisation Systems built using the Dimensional Charting (DC) library were chosen for this project. The DC library is open source; this means that it is a mature technology which has been tested and developed over time. It is relatively straightforward to create a Coordinated Visualisation System with this library and it is available for anyone to use.

The DC library uses Data Driven Documents (D3), a powerful open source library for creating visualisation, allowing the selection of elements, the binding of data to features of the visualisation and the transition of elements from one state to another, providing users with a deeper understanding of how the data changed.

The DC library also uses the Crossfilters library, again an open source library and a library commonly used in coordinated visualisations due to its ability to filter data
extremely quickly when users interact with coordinated visualisations, even for datasets in excess of 1m records¹.

![Sample Dimensional Charting coordinated visualisation](image)

**Figure 4.2: Sample Dimensional Charting coordinated visualisation**

An example of a DC coordinated visualisation is illustrated in Figure 4.2. Through interactions with the visualisations, the data has been reduced to containing only the subset of data pertaining to the years 1997, 1998, 1999 and 2000 in the first and last quarter of those years where daily fluctuation was limited to the range [-3%, 3%]. Due to the variety of visualisations used in this coordinated visualisation system, this is the system that was used during the development and testing of the framework which allows the sharing of expertise through asynchronous guidance and collaboration over coordinated visualisation.

### 4.4 User Tracking Methods

Once the Coordinated Visualisation System had been selected, a number of different means of recording exploration paths were considered.

---

¹ [http://square.github.io/crossfilter/](http://square.github.io/crossfilter/)
The tracking of mouse coordinates is a common method used for attempting to analyse what parts of the websites are of particular interest to users and for tracking where a user clicks on the screen. This was considered as an option for tracking user interactions with the coordinated visualisation system with the intension of reapplying clicks at those locations to recreate the exploration path. The problem with this method however is that the screen needs to be locked in place for this to work. Coordinated visualisations are commonly made up of many visualisations which may not all fit comfortably on the screen at one time. Users may want to zoom into one or more charts for a more detailed view. Some libraries do exist which allow users to scroll up and down on a web page. An example of this is Together.js [Together]. However, on incorporating Together.js with dc.js, it was found that the mouse tracking feature was not very accurate and, more importantly, that it did not work over visualisations. When the cursor was moved over a visualisation, the mouse coordinates remained stuck at the point at which the cursor entered the visualisation. An alternative method for tracking user interactions was therefore explored.

The option of saving the filtered data at each step was also considered. By extracting the data at each step and saving it, the stored data could be reapplied to the coordinated visualisation system so as to display the visualisation as it appeared during the exploration path. However, this could limit the volume of data which can be used, limit the number of steps which can be stored and require large storage space. The use of large datasets to implement the system could also lead to latency problems.

A better option is the use of filters. As a user interacts with a visualisation, the data is filtered by means of the Crossfilter library so that only data satisfying the criteria of the filter is considered. A filter on a visualisation is made up of the identifying features of the selected nodes for that visualisation. For example, consider a visualisation which groups a set of data ranging from the year 2000 to the year 2010 by year. When a user interacts with the visualisation to select a subset of those
years, 2000-2002 say, the filter on that visualisation will be [2000, 2001, 2002]. Other visualisations within the coordinated visualisation may also have filters applied to them which will affect what data is displayed in this visualisation but these will not appear in this visualisation’s filter.

On analysing how DC.js works, it was found that as a user interacts with a visualisation, the new filter on that visualisation, obtained from the Crossfilter library, is used to redraw that visualisation through calling on D3 methods with this filter. Since slices of the yearly dimension are being included or excluded, there is no need for the recalculation or the recounting of data in the various slices for that visualisation. Changes in the visualisation will simply be changes to which nodes are highlighted. However, the overall subset of data used is revised, necessitating the recalculation and recounting of data in the various slices of the other dimensions before they can be redrawn. The recalculation and recounting of data is performed by the Crossfilter library while the redrawing of the visualisations is handled using the D3 library. The DC library acts as intermediary between the two, calling the necessary function in each.

It was found that the filters could be extracted from the DC library when interactions with the visualisation triggered the redrawing of the coordinated visualisation. These filters could then be reapplied as long as the name of the chart and the filter applied to that chart were stored. Due to the ease with which this could be done, the increased accuracy over the tracking of mouse clicks and the significantly smaller amount of data which needed to be saved at each step, this was the chosen method for logging exploration path records.

4.5 Storing Interactions

Since the system needs to log information each time the user interacts with the coordinated visualisation system, the use of the jQuery library was found to be the
simplest method for triggering the events required to store the information at each interaction.

Initially storing the data on the client side through localStorage was considered [LocalStorage]. localStorage is a HTML feature which allows the storage of large amounts of data on the user’s computer with no expiration date, even after the browser session is ended. Each time the user re-opens the application, the data is still available. Data is stored in key-value format, as with JSON. However, a different key is required for each localStorage record and issues arise when attempts are made to dynamically generate the key. For example, if a variable is incremented at each interaction so as to be used as the key for the record, the initial value of the variable rather than the current value will be used so that the first record is repeatedly overwritten. The use of localStorage would also have led to complications when maintaining several tables. To share an exploration path with another user, all of the stored records for that exploration path would have needed to be extracted and sent to the other user if using local Storage.

It was found that using a database for storage would eliminate a number of issues and improve the functionality of the system. Splitting the architecture into a client side and a server side would facilitate the sharing of exploration paths with others for collaboration or guidance. Different tables could be created for different exploration paths; this would eliminate the need to extract records relevant to one exploration path from set of records pertaining to a number of different exploration paths by dealing with data from the required table only. Exploration paths could then easily be shared with others by simply providing them with the Replay name.

With the introduction of a server-side and database into the design, a method of communicating with the server was required. Due the asynchronous property of AJAX, it was chosen as a method of communicating with the server. This would allow the web application to be updated as required rather than reloading the application each time.
A key advantage of using REST APIs is that they simplify communication between the server and the client. The Slim Framework is a REST API which is open source, lightweight and easy to incorporate into web applications. By using the Slim Framework, AJAX requests could be rerouted to anonymous server-side functions enveloped in functions identified by the same HTTP method and by the same resource as that contained in the URI of the AJAX request. By definition of a REST API, it enforces the use of a stateless server-side which ensures that all session state is kept on the client side. Multiple clients could then operate with the same server and, since operating a stateless server allows each request to be dealt with independently, the potential for latency due to the presence of multiple clients could be reduced.

4.6 Linking Technologies
The final design of the system is as follows:

- The Coordinated Visualisation System is created using HTML, CSS and the DC.js library which incorporates the D3.js and Crossfilter.js libraries.
- Additional features are added to the HTML file to allow for the addition of annotations, the displaying of another user’s annotations, the move from the exploration mode to the replay mode, the stepping through of an exploration path while in replay mode and the option of creating a new Replay or opening a different existing Replay. These additional features are styled using an additional CSS file.
- jQuery is used to trigger event handlers when the user interacts with the system, be it in exploration mode or in replay mode.
- Where event handlers require the sending or retrieval of data from the database, AJAX requests are used on the client side.
- The Slim Framework reroutes the AJAX requests to the anonymous function required to deal with that request.
• The anonymous functions make up the server in the framework. They query the database to create tables, insert records, update records, retrieve partial or full records and delete records. Where required, they return data to the callback function from which the request originated on the client side.

• On receiving data from the server, the client performs the required actions with that data, for example moving from exploration mode to replay mode.

While this chapter concentrated on the design side of the framework, the next chapter will look at how this is implemented.
5 Implementation

Having identified the design requirements and the technologies with which to satisfy these requirements, this chapter looks at how the system was implemented. In order for this framework to be added to existing coordinated visualisations with minimal effort, where possible the code for the additional features which enable the tracking of user interactions and replaying of exploration paths was kept separate. However, some modifications to the coordinated visualisation system were inevitable for the system to be implemented. This chapter begins by outlining the modifications made to the existing coordinated visualisation system. The implementation of the additional features to convert the coordinated visualisation system into one that enables the sharing of expertise is then described. This is split into the implementation of the database, jQuery event handlers triggered by user interactions in line with the user flow and communication between the client and server.

5.1 Coordinated Visualisation System Modifications

5.1.1 HTML & CSS

Modifications must be made to the user interface to allow for the addition and displaying of comments and the replaying and stepping through of exploration paths. Users may also wish to change from one exploration path to another, or start a new exploration path. Figure 5.1 illustrates an example of how the additional features for these actions to be possible appear within a coordinated visualisation. In this example, the additional elements have been added between the coordinated visualisation title and the first visualisation.

As can be seen, the name of the current Replay and the date on which it was created are displayed below the title. This can act as a reminder of the name given to the Replay, or a method for the user to check that they are looking at the correct Replay. By clicking on “Change/New” below the details of the current Replay, the user can
choose to open a different existing exploration path, or create a new exploration path.

![Figure 5.1: Additional elements in the user interface.](image)

A text box is included in which the current user can add comments. Where another user has contributed commentary, this is displayed beside the text box. Below the text box is a “Save Comments” button on which the user must click to save their comments. It was decided that the user would have to click on a button to save a comment rather than saving the comment automatically when the user moved to the next step of the exploration path to avoid situations where the user may accidentally delete the comment. Comments will only be deleted if the user clears the text box and clicks on the “Save Comments” button.

The “Replay” button allows the user to start an exploration path from the beginning, whether in exploration mode or already in replay mode. The “Next” button is only displayed when the user is in replay mode. When the user pauses a replay, the “Next” button is replaced with a “Resume” button. The implementation of this will be described in Section 5.4.

The code which must be added to the HTML file to insert these elements is generic and can be copied and pasted directly into the HTML of any DC coordinated
visualisation system. The layout of the comments of the current user relative to those of another user is formatted in a separate CSS file. A link to that CSS file must therefore be included in the HTML document.

As mentioned in the previous chapter, jQuery is used to trigger event handlers when a user interacts with the system. The jQuery library must therefore be included in the HTML document as one of the source libraries. jQuery event handlers can listen for interactions with a particular type of element, a single element with a particular id or all elements belonging to a particular class. Therefore in order to be able to listen for interactions with all of the charts within the chart group, the charts are defined as belonging to a particular class. This is implemented by wrapping each chart in a div of a particular class, such as “dc-chart”.

5.1.2 DC.js
As the user interacts with a visualisation, filters are applied to that visualisation. This is done within the DC.js library in an event handler. The filters applied are only stored while the visualisations are being redrawn. In order to be able to extract the filters, a global object, dc.replay, is added into the DC.js library. This object's filters attribute, dc.replay.filters, is updated each time the event handler is called.

There are two types of filters which can be applied to DC visualisations – filters made up of an array of values and ranged filters. For example, a filter containing data for the years 2000 to 2002 only would be a filter array extracted in the form \([2000, 2001, 2002]\) while a filter containing data ranging between -3% and 3% would be a ranged filter extracted in the form \([-3.0, 3.0]\). The methods in which these two types of filters are applied to visualisations are not the same. So as to be able to implement the correct method during replay, it is necessary to extract additional information about whether the filter is ranged or not. A second attribute of the global object, dc.replay.rangedFilter, is updated within the “dc.filters.RangedFilter” function, a function which is only called when a ranged visualisation is filtered.
The value of the global object can then be pulled into the event handlers specific to the asynchronous collaboration framework.

5.2 The Database
The database sits at the opposite end of the system from the coordinated visualisation. While intuitively it may not make sense to jump to how the database was implemented, understanding what data is collected, how it is stored and why it is necessary to store it will simplify the explanation of how the various paths through the flow chart were realised.

One of the benefits of using a database to store exploration paths which was highlighted in the design chapter is the ability to create new tables for each exploration path. Each time a Replay is created, be it a new exploration or one cloned from another Replay, a new table is created. As the exploration path is continued, new records are inserted into the table. These include the name of the visualisation to which the filter was applied, the filter that was applied to the visualisation, and whether the filter is a ranged filter. As annotations are added, records are updated.

As was seen in the flow chart, when the system is initially started, the user must enter the name of the Replay they wish to use. A method is therefore required to check whether the Replay already exists, or whether it is a new Replay. A table listing all existing Replays is therefore maintained. This table contains the name of the Replay and the date on which it was created. This information is retrieved and displayed as illustrated in Figure 5.1 for the current replay being created or revisited. The table also contains the name of the user who created the table. This is used to differentiate between comments made by the author of the table and those made by a collaborator.
A Current Filters table maintains the filters applied to each visualisation in the coordinated visualisation at each exploration or replay stage, except where replay is paused as interactions are not logged in this mode. One reason for maintaining the Current Filters table is to avoid duplicate entries. If a user clicks on a graph without altering the filter, such as in the node free area of a bubble chart, the interaction with the chart will trigger an event handler resulting in the logging of a duplicate entry. By comparing the new filter to the current filter for that visualisation, the system ensures that only valid interactions are logged.

Another reason for maintaining the Current Filters table is to avoid latency issues where a replay is paused and later resumed. Two options were considered to resume the exploration path from paused replay mode. This could be done by stepping through the entire replay up until that point, applying the required filters to the appropriate visualisations. However, exploration paths can be extremely long and hence could result in latency issues. In order to ensure the speedy resumption of the exploration path, the maintenance of the Current Filters table was implemented. Since the table is not updated during paused replay mode, this allows the filters for each visualisation at the point where the Replay was paused to be retrieved from the database and applied to the required visualisations. At most, this will involve the application of filters to the each of the visualisations, while reapplying the entire exploration path could require applying the filters to the visualisations multiple times.

5.3 Implementing the User Flow

The flow in Chapter 4 illustrates the different user processes due to interactions between a user and the client-side of the system. Each time a user interacts with the system, an event handler must be triggered to perform the required actions using jQuery. The different user interactions fall under the following headings:

- Applying filters to a visualisation during exploration
- Removing all filters from a visualisation during exploration
• Removing all filters from all visualisations during exploration
• Saving annotations during exploration and replay
• Starting or restarting a Replay
• Stepping through a Replay
• Pausing and Cloning the Replay
• Resuming the Replay
• Changing or creating a new Replay

jQuery event handlers can be triggered by interactions with elements belonging to a particular class, of a particular type or with a particular ID.

5.3.1 Triggers by Class
Having enveloped each of the visualisations declared in the HTML document in divs assigning the visualisations to the same class, an event handler can be created which is triggered each time a user interacts with an element from that class. For example, if the charts belong to the “dc-charts” class, the event handler

```javascript
$(".dc-chart").click(function() {...})
```

will be triggered. In this function, the ID of the visualisation with which the user interacted can be extracted using the ‘this’ identifier.

There are three system modes in which the user may interact with the visualisations – exploration mode, replay mode and paused replay mode. The actions taken by the event handler depend on which mode the system is in.

Where the system is in exploration mode, the system
• Extracts the visualisation ID from the HTML document using the ‘this’ identifier within the jQuery
• Extracts the filter information – what has been applied to the visualisation and whether the filter is a ranged filter – from the DC library
• Compares the new filter to the current filter. This is done by checking whether a record exists in the current filter table for that visualisation. If it
does and the current filter is equal to the new filter, the interaction is
discarded and no further action is taken.

- Creates a new current filter record or updates the existing current filters
  record as required.
- Wraps the data to be saved in the exploration path table in a JSON object and
  sends the data to the database.

Where the data is in replay mode, the user is given the option to pause the replay. If
they choose not to pause the replay, a new Replay is created, cloning the exploration
path up to the current point, then adding the additional user interaction record. The
user is asked to input a Replay name. The system checks whether a Replay exists
with that name. If it does, the user is prompted to enter a different name. If the
Replay name does not exist, a new table is created. When the system enters replay
mode, the records for the replay are retrieved from the database and stored locally
in an object array. The system sends each record, up to and including the current
record, in sequence to the server to be logged into the new table. Finally the latest
filter applied to the visualisation which triggered the event is logged. The system
changes into exploration mode.

Where the user chooses to pause the replay rather than cloning the replay, the
system goes into paused replay mode. No interactions are logged in paused replay
mode.

5.3.2 Triggers by Type
Removing all filters from a visualisation, removing all filters from all visualisations
and changing or creating a new Replay event handlers are triggered by listening for
interactions with elements of a particular type, in this case HTML href elements.

When adding the additional features to the HTML document, the “Change/New” link
to change or create a new exploration path is created using a HTML href element. An
ID is given to the “Change/New” element so that it can be differentiated from the
other href elements. A jQuery event handler was created to be triggered each time a user interacts with a href element. If the ID matches that of the “Change/New” element, a prompt box appears asking the user to insert the Replay name they wish to use. The system checks whether the Replay exists. If it doesn’t, a new table is created and the table of existing Replays is updated, using AJAX requests. If the Replay does exist, the date on which it was created and the author are retrieved from the existing Replays table, all records for that exploration path are retrieved from the table for that Replay and the Current Filters table is cleared.

It is common to have the option to reset a single visualisation or all visualisations. In DC visualisations, this is commonly done using HTML href elements. Where the href ID does not match that of the “Change/New” element, it will indicate the resetting of one or all charts. In order to differentiate between resetting one visualisation and resetting all visualisations, the ID of the visualisation is extracted. In the case where all visualisations are to be reset, the ID will be blank. ‘reset’ will be stored instead of a visualisation name to indicate that all visualisations must be reset during Replay. Empty filters were found to sometimes cause problems when data retrieved from the database was parsed. It was therefore decided that the word ‘reset’ would also be stored instead of the empty filter when one or all visualisations were reset.

5.3.3 Triggers by ID
The remaining interactions are controlled by interactions with buttons, each of which is assigned a particular ID.

The “Save Comments” button is used to update the current record with the contents of the text box. The method varies slightly depending on whether the system is in exploration mode or in replay mode. Where the system is in exploration mode, the ID of the last record logged must be retrieved from the database. It is then included in a JSON which is sent to the server with the text box contents so that the server can update that record. In Replay mode, all of the data for that exploration path is retrieved from the database and stored locally. The client can therefore access the
database ID of the current record and include it in the JSON with the comment and sent to the server.

The “Replay” button is used to return to the start of an exploration path, whether in exploration mode or already in replay mode. When the user is in exploration mode, all of the data for the exploration path is retrieved from the database and stored locally in an array of objects. The length of the array is stored in a variable ‘replayNo’; this is the number of records in the exploration path. A variable ‘replayCount’ is used to step through the array of replay steps. If the system was already in replay mode, there is no need to retrieve the data from the database, the variable ‘replayCount’ is simply reset to zero. The first step of the exploration path is recreated. If the system was in exploration mode, it enters replay mode, causing the “Next” button to appear.

The “Next” button is used to step through the exploration path. Each time the user clicks on the “Next” button, the next step of exploration path is recreated by applying the recorded filters to the relevant visualisation. When the Replay is interrupted and the user chooses to pause the Replay for a side exploration, the “Next” button becomes the “Resume” button. Due to the importance of this button, it is described in depth in the next section.

5.4  The “Next” Button
Interaction with the “Next” button is what drives two key features of the framework – the recreation of the exploration path and the ability for users to deviate from the exploration path temporarily and resume the Replay where they left off.

5.4.1  Stepping through Replay
The “Next” button is used to step through the exploration path. As was mentioned earlier, when the saved exploration path records are retrieved from the database, a ‘replayCount’ variable is set to zero and the number of records in the exploration path is stored in the variable ‘replayNo’. Each time the user clicks on the “Next”
button, the ‘replayCount’ variable is incremented, provided it is less than the value of ‘replayNo’. The ‘replayCount’ variable always refers to the current record it is displaying for the following reasons:

- When a user wishes to save a comment during annotation, the database ID of the current record can easily be extracted and sent to the server with the comment.
- If a user chooses to pause the system, the variables continue pointing to the current record to which the user wishes to return.
- If a user chooses to clone an exploration path up to the current point, the array up to the current record can be copied into a new table.

Once the variable has been incremented, the system looks at the data for that record. As previously mentioned, when one or all visualisations are reset, the word ‘reset’ is saved instead of the filter. If the filter is ‘reset’, it checks the name of the visualisation. If the visualisation name is ‘reset’, the filters are removed from all of the charts before rendering all of the charts. Otherwise, the filters are removed from the required chart before rendering all of the charts.

It was noted earlier that ranged filter visualisations are handled differently to array filters. If the filter is not ‘reset’ the system must check whether the array is ranged or not. If it is ranged, the current filter is replaced with the new filter.

If the filter is not ranged, the current filters must first be removed before the new filters are applied. The reason for this is that applying a filter twice will remove the filter. For example, if the current filter is \([2000, 2001, 2002]\) and the new filter is \([2000, 2001]\), applying the new filter on top of the old filter will leave only data for 2002 displayed, that is the data which should have been removed. Removing all filters from the visualisation before applying the new filters overcomes this problem. For array filters, each filter in the array must be applied individually. The entire chart group must then be rendered.
As well as applying the stored filters to the appropriate visualisation, the comments for the record must be displayed. The user's comments for the current record are displayed in the text box. This is implemented in this way in order to allow the user to edit their comments. Comments made by another user are displayed in the area beside the text box, as was illustrated in Figure 5.1. These cannot be edited by the user. In order to separate comments made by different users, two comments fields are maintained in the exploration path table in the database. Where the current user's username matches that of the author of the table, comments are saved in the first comments field. Otherwise, comments are saved in the second comments field. Where the current user's username matches that of the author of the table, the contents of the first comments are displayed in the text box and those of the second comments field are displayed in the area beside the text box. Otherwise, the locations where the comments are displayed are inverted.

5.4.1 Resuming Replay
The HTML element with “Next” as ID also operates as the “Resume” button. When a Replay is paused, as outlined in the Trigger by Type section, the text displayed on the “Next” button is changed to “Resume” and the system enters paused replay mode. Therefore, each time the event handler for the “Next” button is triggered, it first checks whether replay has been paused. If it has not been paused, it proceeds as described in the Stepping through Replay section above. However, if the system has been paused, this is an indication that the user wishes to resume the Replay at the point where it was paused. This could be done by stepping through the entire replay up until that point, applying the required filters to the appropriate visualisations. However, exploration paths can be extremely long which could result in latency issues.

In order to ensure the speedy resumption of the exploration path, a Current Filters table is maintained in the database. Each time the coordinated visualisation changes, the Current Filters table is updated for the visualisation which caused the change in
the visualisation, regardless of whether the change occurred during in exploration mode or replay mode. However, while the system is in paused replay mode, the Current Filters table is not updated. This allows the filters for each visualisation at the point where the Replay was paused to be retrieved from the database. By clearing all filters on all visualisations and then applying the filters to the corresponding visualisations, the exploration path can swiftly be resumed at the point at which it was paused.

It is worth noting that as well as a “Next” button, the implementation of a “Previous” button was considered. This would add to the functionality of the framework as it would allow users to navigate backwards through the exploration path should they wish to revisit a particular step. Without the implementation of a “Previous” button, the user would need to restart the Replay. In order for this to be implemented, the system would need to reapply every step of the exploration path from the beginning which, for long exploration paths, could lead to latency issues. Unlike the “Resume” button where the current filters table is used to circumvent the need to reapply all of the filters, no alternative solution was identified using the selected design.

5.5 Client-Server Interaction
So far the implementation has focused on the user interaction with the client-side of the architecture. However, nearly every time the user interacts with the system, there is some communication between the client and the server. This takes place in the form of an AJAX request from the client which is intercepted and rerouted by the Slim Framework to execute the required server code which queries the database.

AJAX requests always contain a HTTP method, a URL identifying the resource of interest. It will also contain data where data is required to execute the database query as no information is saved on the server. The HTTP methods used in this framework are:

- GET to retrieve data from the server
- PUT to update data on the server
- POST to send data to the server and create tables
- DELETE to delete data from the server

The resources of interest are the existing Replay tables, the current filters and the exploration path records.

In the first description of the interaction between the client and server, the code is provided and described in detail. However, the steps are very similar for each of the methods used so any difference of note will simply be described as it arises.

5.5.1 Table of Replay Tables
Where the resource of interest is the existing Replay tables, this requires the querying of the table containing a list of existing Replays. Two HTTP methods are used with this resource.

The GET method is used to check whether a Replay exists. The system checks whether a Replay exists when the application is first opened, when a Replay is interrupted and cloned, and when the user clicks on the “Change/New” link. Each of the event handlers triggered by these user interactions calls the checkIfExists function illustrated in Figure 5.2.

```javascript
function checkIfExists(tableName) {
    var tablename = JSON.stringify(tableName);
    $.ajax({
        type: 'GET',
        url: rootURL + '/checkTable/' + tablename,
        success: function(data){
            if(!clone){
                changeTable(data);
            } else {
                replay = false;
                $('#next').hide();
                cloneTable(data);
            }
        }
    });
}
```

**Figure 5.2:** checkIfExists function with AJAX request using GET method
The GET method only allows the inclusion of data in the request if it is appended to the URL, therefore the table name is appended to the URL. The Slim Framework reroutes the request to the anonymous function matching the method and URI. As can be seen in the first line of Figure 5.3, the Slim application points to a ‘get’ function where the end of the path matches the end of the URL in Figure 5.2.

```php
$app->get('/logged/checkTable/:tablename', function ($tablename) use ($app) {
    $sql = "SELECT * FROM listoftables WHERE tablename='$tablename'";
    try {
        $db = getConnection();
        $stmt = $db->query($sql);
        $tableinfo = $stmt->fetchAll(PDO::FETCH_OBJ);
        $db = null;
        echo json_encode($tableinfo);
    } catch(PDOException $e) {
        echo '{"error":{"text":' . $e->getMessage() . '}}';
    }
});
```

**Figure 5.3**: Anonymous function identified through the HTTP method and resource identifier using Slim Application

In the anonymous function, the SQL query is prepared in a statement using the data from the client to set the criteria. A PDO is used to connect to the database, then query the database. All data satisfying the query is selected and returned as objects into the $tableinfo variable.

The database connection is terminated by setting it to null. The data to be returned to the client is encoded in JSON format and echoed back to the callback function. The data is then used by the client for to complete further actions. As the table name is the Primary Key for that table, at most one record can exist. If the JSON object is empty, this signifies that no Replay of that name exists.

If the Replay does not exist, a new table will be created, as described in the Replay Tables subsection below. The table of Replay tables must be updated to ensure a
Replay is only created once under that name. This is done using a POST method in the AJAX request. Where a new table is created, the current user will be the author. The username, the new table name and the date are stringified as a JSON with keys identifying the valued and included in the AJAX request. The HTTP request is rerouted by the Slim Framework to the required anonymous function as before. This function decodes the JSON to extract the required information for the database query, prepares the PDO as described in the related technologies section and executes the PDO, thus adding the new Replay to the table as required.

5.5.2 Table of Current Filters
Five AJAX requests use the current filters as the resource of interest. Three of these are used during the logging of exploration paths.

A GET method is used to check what the current filter on a particular visualisation is. The name of the visualisation must be included in the AJAX request. The Slim Framework reroutes the request to an anonymous function that selects and returns the current filter for that visualisation in JSON format. Where the JSON is empty, the exploration record is saved as will be described in the Replay Tables subsection below and a new record is created in the Current Filters table using an AJAX request with a POST method. This AJAX request contains the name of the visualisation, the filter and whether or not the filters are ranged filters wrapped in a JSON object. The corresponding anonymous function decodes the JSON and uses the data to create a PDO with which to query the database, creating a new record with this data. If however the JSON returned by the GET method is not empty, the current filter is returned by the GET method is compared to the new filter for the visualisation. Where these are equal, the exploration record is not logged. Where these are different, the exploration data is logged and the record for the visualisation in the current filters table is updated using an AJAX request with a PUT method. The data to be updated must be included in the AJAX request. The anonymous functions decodes the data to create a PDO with which it updates the database.
One of the main advantages of maintaining the current filters table outlined in section 5.2 is the use of the current filters to resume the exploration path at the point where it was paused. When a user clicks on the “Resume” button, an AJAX request containing the GET method with the current filters table as a resource is fired. The anonymous function to which it is rerouted executes a query selecting all of the data from the table, encodes the data in JSON format and returns it to the callback function. The JSON can then be parsed and the filters can be applied to the relevant visualisations.

In order for the current filters table to remain accurate, it needs to be cleared when a new Replay is loaded or created, when all visualisations are reset, when the Replay is restarted. The AJAX request contains the DELETE method. The Slim Framework reroutes the request to an anonymous function that deletes all records in the table.

5.5.3 Replay Tables
Due to the fact that the database stores multiple exploration paths in individual tables, all AJAX requests where the resource of interest is the exploration path records must contain a minimal amount of data – the name of the table which must be queried. Five AJAX requests require the querying of the current Replay table.

Once a user enters the name of a new Replay and it has been established that it does not already exist, the table must be created. This is done using a POST method. The name of the new Replay is included in the AJAX request. Since all Replay tables are made of the same attributes, the table name is the only data which needs to be added to the PDO in the anonymous function in order to query the database and create the new table.

Once the table has been created the first record must be inserted, that is the record where the full data set is used. This record must be logged in case the user wishes to add comments to the first exploration step. Exploration path records are logged using a POST method. The data is wrapped into a JSON object on the client side and
sent to the server using the AJAX request. The anonymous function decodes the data and uses it to create the PDO to query the database and insert the data required to recreate that step at a later date.

Comments for a specific record are updated using a PUT method. Where the system is in replay mode, the client has access to the ID of the record which needs to be updated. Where the system is in exploration mode, the AJAX request to update the comment must be preceded with an AJAX request using a GET method which returns the ID of the last inserted record. As before the AJAX requests are rerouted to the required anonymous functions and the corresponding queries are executed. The ID of the last inserted record is returned to the client to be used in the request to update the comment. Due to the fact that data cannot be stored in the stateless server, the ID cannot be used without returning it to the client first.

When an existing Replay is opened or when the user changes from exploration mode to replay mode, all of the data stored for the Replay must be retrieved from the database. This is done through an AJAX request using the GET method which, as previously mentioned, must contain the relevant Replay name. The data is retrieved from the database by the anonymous function, encoded in JSON and returned to the callback function. The JSON can then be parsed and stored in an array and the exploration path can be stepped through using the “Next” button.

5.6 User Example
Consider a user who wants to create an exploration path, replay the exploration path, pause the exploration path and then clone the exploration path, all the while annotating the paths.

5.6.1 Creating the Exploration Path
As the user interacts with the visualisations, the following events occur:
The jQuery event handler which listens for interactions with any element from the class to which all the visualisations have been assigned to is triggered. Within the event handler, the name of the visualisation is extracted from the HTML.

The system checks whether it is in the replay or paused replay mode. As this is the exploration path, a function is called to prepare the data to be logged in the exploration path record. The filter is extracted from the DC library. The ranged filter is also extracted from the DC library. If the visualisation with which the user has interacted uses ranged filters, the filter and the ranged filter will be equal; if it uses filter arrays, they will not be equal. The two are compared and the ranged variable to be logged is updated accordingly.

As previously mentioned, the current filters table is used to ensure duplicate entries are not saved. Using a GET method, the current filters as a resource and the chart ID as data, an AJAX request is fired to the server. This is intercepted by the Slim Framework which reroutes it to the anonymous function defined within the Slim Application with the same method and URL as the AJAX request. The database is queried and the current filter for that chart ID is returned to the client callback function.

A function is called to check whether or not the new filter is equal to the current filter for that chart. Three options are possible:

1. No current filter is logged for that chart
2. The current filter logged for that chart is different to the new filter
3. The current filter logged for that chart is the same as the new filter.

In the first scenario, a new record must be created for the current filter and the exploration path record must be logged. An AJAX request using a POST method to insert a new record into the current filters table is fired with the required data. The data is made up of the chart id, whether the chart is ranged or not and the current filter. It is wrapped into a JSON object before it is sent to the server as follows:
The Slim Framework reroutes the AJAX request the required anonymous function. The data is decoded and injected as required into the SQL query so that the new record is created. Similarly, the exploration data is wrapped into a JSON object containing the name of the Replay, the name of the chart, whether it is ranged and the filter. As was mentioned, due to the fact that different tables are used to store the different exploration paths, the name of the Replay must always be included in the data when querying the Replay table. Using the POST method again, the data is sent to the server, decoded and inserted into the Replay table.

In the second scenario, the current filters table must be updated for the given visualisation. Using the PUT method and the chart ID as criteria, the new filter is wrapped in a JSON object and sent to the server via an AJAX request. The data is extracted and the table is updated for the record with the corresponding chart ID.

In the third scenario, the fact that the filters are the same is an indication that this is a duplicate entry and it is therefore discarded.

### 5.6.2 Replaying the Exploration Path

When the user clicks on the “Replay” button the jQuery event handler which listens for interactions with the element with ‘replay’ as ID is triggered. The replayNo and replayCount variables are set to zero.

An AJAX request with the GET method and the Replay name as data is sent to the server. Through the rerouting of the Slim Framework, all elements in the table are selected, JSON encoded and returned to the client callback function. The data is parsed converting the JSON into an array of objects. Length of the array, or the number of exploration path records, is stored in ‘replayNo’.
Since the first record of the exploration path is always a completely unfiltered view of the visualisations, all filters are removed from all of the visualisations.

There are two comments fields in the exploration records table. The first is for the author’s comments, the second is for the other user’s comments. The system checks whether the author and the user have the same user name and assigns the comments to their locations accordingly.

The system moves into the replay mode and the “Next” button is displayed.

The current filters table must be cleared since all filters have been removed from all of the visualisations. An AJAX request with the DELETE method and the required URL is sent to the server. The anonymous function queries the current filters table deleting all records.

The user can now step through the exploration path. The method for stepping through the exploration path was outlined in detail in section 5.4. In brief, the replayCount variable is incremented. As long as it is lower than the replayNo variable, the system checks if the filter for that record is ‘reset’. If it is, it checks if the chart name is ‘reset’. If it is, this means that all charts must be reset. If on the filter is ‘reset’ this means that only the chart needs to be reset. If the filter is not ‘reset’, the system checks whether it is ranged or not. If it is not ranged, the current filters must first be removed, then the new filters can be applied. If it is ranged, the range is replaced with the new range. All charts must be redrawn and as before the comments are displayed in the required locations, depending on whether the user is the author or not.

The user can continue to step through the replay until it reaches the end, or until it interrupts the replay.
5.6.3 Pausing the Exploration Path
When a user interacts with a visualisation, the relevant event handler is triggered. It checks to see whether the system is in exploration mode. If it is not, it checks to see if it is in replay mode. Where the system is in replay mode and the exploration path has been interrupted, the user is given the option of pausing the system and returning to it later. If the user chooses to pause the system, the system enters paused replay mode and the text on the Next button is changed to Resume.

In paused replay mode, all interactions with the coordinated system are discarded, except for interaction with the Resume button. Once the user clicks on the Resume button, an AJAX request is sent to the server using the GET method with the relevant URL. All records from the current filters table are returned to the client. All filters are cleared from all of the visualisations; then the filters from the current filters table are applied to the corresponding visualisations. The user can then continue through the exploration path from the point where they left off.

5.6.4 Cloning the Exploration Path
Where the user decided not to pause the system on interrupting a Replay, the user is asked to enter the name of the new Replay. The system queries the table containing a list of existing Replays table using an AJAX request with a GET method and the table name as data.

If the table exists, the user is prompted to enter a different Replay name. If the table doesn’t exist, a table is created using an AJAX request with a POST method and the name of the table as data. A record of the new table must be inserted into the list of existing Replays table. The date the Replay was created, the author of the Replay’s username and the name of the new Replay are rolled into a JSON with their corresponding identifiers. Using a POST method, the data is sent to the server. The server decodes the data, inserts it into the database query as required and inserts a new record in to the database.
The records from the replay data array are inserted into the new table, starting with the oldest one, but only up to the current replay record, using the POST method, as described in the Creating the Exploration Path section. Once all records to be cloned have been added, the data is prepared for the new record. The same steps are taken as described in the Creating the Exploration Path section, that is the filter and ranged filter are compared to see if the visualisation is ranged, the new filter is compared to the current filter for that visualisation, and so on.

The system enters exploration mode so the Next button is hidden.

This chapter has described the implementation of the design. The next chapter will look at how well it satisfies the requirements.
6 Evaluation

6.1 Evaluation method

Once the system had been implemented it was tested using the user flow as a guide on two different coordinated visualisation – the Nasdaq Index visualisation used to test and develop the system and an Irish Properties Prices visualisation on which the additional components had not been tested. The main features from the user flow which would determine whether or not the system met the requirements were:

1. Users should be able to explore a visualisation while making annotations, then replay the visualisation, adding and editing the annotations as desired.
2. Users should be able to share an exploration path with others, who in turn should be able to replay the visualisation, adding and editing their own annotations as desired.
3. Users should be able to pause the reconstruction of an exploration path for a momentary side exploration, then resume where they left off, regardless of whether they are the author of the exploration path.
4. Users should be able to clone an exploration path up to a certain point and append any further exploration, regardless of whether they are the author of the exploration path.

To check whether the system was able to perform these actions, the evaluation described below was carried out.

A new replay was created using the username 'user1'. Filters from each of the charts were applied sequentially, making annotations in some but not all of the steps. The replay button was selected and the exploration path was stepped through until the end, adding additional annotations and editing some of the initial annotations. The exploration path was then appended to.

The replay button was selected again and a part of the exploration path was stepped through. While in replay mode, one of the visualisations was interacted with causing the system to provide the option of pausing. The system was paused and further
filters were applied to a number of visualisations. The resume button was selected and the Replay was resumed.

While still in replay mode, one of the visualisations was interacted with. This time the option to clone the visualisation was selected and additional exploration steps were appended to the new exploration path. The Replay button was selected to step through the exploration path from start to finish.

The Change/New link was selected and the first Replay was restarted. Using the Next button, the exploration path was stepped through from start to finish to check that the additional steps had been logged.

The browser was refreshed and ‘user2’ was provided as a username. The first Replay was opened and stepped through, checking that the ‘user1’ annotations were displayed in the correct location. Annotations were added along the way. The system was paused and resumed as before.

The browser was refreshed and ‘user1’ was provided as a username. The first Replay was opened and stepped through, checking that the annotations for each user were displayed in the correct locations.

6.2 Observations
The system was found to behave as required, not only for the Nasdaq Index visualisation but also for the Irish Property Price visualisation. The sending and retrieval of data to and from the database did not appear to affect the latency of the embellished system when compared to the original system.

When the exploration path was replayed for the first time, the filters applied during the initial exploration were reapplied in the same order and annotations were displayed. When the first few steps of the exploration path were replayed, the same
filters were reapplied and the additional and edited annotations were displayed. The first action in the set of requirements was therefore successfully completed.

When the system was paused and resumed, the visualisations were restored to the state they been in when the system was paused. When part the beginning of the exploration path was cloned, appended to and replayed, the required steps from the cloned exploration path were present and followed by the appended steps. The third and fourth actions in the set of requirements were therefore successfully completed.

When the Change/Next link was selected and the initial Replay was reopened, the exploration path, including the appended steps, was still as required with annotations displayed correctly.

When the browser was refreshed and used as ‘user2’, the comments made by ‘user1’ were located to the right of the text box so that they could not be edited. Pausing, then resuming the system after a deviation in the exploration path returned the system to the point where the exploration path had been logged. The second action in the set of requirements was therefore successfully completed.

When the browser was refreshed and the initial Replay was opened using the username ‘user1’, the ‘user1’ comments were displayed in the text box while the ‘user2’ comments were displayed to the right of the text box, as required. This demonstrates that the system can be used for collaborative work.

From these observations, it can be seen that the system satisfies all of the scenarios from the user flow. In doing so, the goals set out in this project were achieved. These were to:

- Store sufficient information about every user interaction with the coordinated visualisation to be able to recreate the user interaction.
• Replay the exploration path on demand by reapplying the user interactions.
• Facilitate the addition of annotation by the initial expert during the exploration and during the replay.
• Allow others to replay the exploration path, adding their own commentary if required.
• Enable the pausing of the replaying of the exploration path for an alternative exploration with the possibility of resuming where the deviation began.
• Allow viewers to clone the exploration path up to a certain point and append alternative exploration with minimal effort.

A point worth noting is that as filters are applied to visualisations, these are extracted from the DC library and recorded with the name of the visualisation. Where a method might be overwritten so as not to make certain changes to a visualisation, the overwriting of the method may not be recreated during the replay, depending on how it is implemented. This inconsistency between the actual exploration path and the recreated exploration path appeared when it was noticed that the Days by Fluctuation chart in the Nasdaq Index visualisation was not redrawn correctly when that chart alone was reset. The range for the previous filter was still highlighted when this should not have been the case. During the Replay, the highlighted section was removed. This was a small bug with the Nasdaq Index visualisation which has since been fixed; however it did identify the fact that if a DC coordinated system were purposely customised, this might render this framework incompatible with that DC coordinated system.
7 Conclusion
This project proposed to develop a framework which allows the sharing of expertise through asynchronous guidance/collaboration over coordinated visualisation. Goals were set where the achievement of these goals would signify the successful implementation of the framework.

On testing the finalised system, it was found that the developed framework was able to extract the required information during the exploration path to subsequently recreate that path by reapplying user interactions, allowing users to step through each stage at their own pace. Annotations could be added during the creation and the reconstruction of the exploration path. Other users could replay the exploration path, add commentary, pause the system to explore alternative paths and then resume where they left off, and clone the exploration path to a certain point, then append an alternative exploration path. This indicated that all of these goals set against which to measure the merit of the developed framework were in fact realised.

Where the framework was found to be somewhat limited was that only the initial expert and one other collaborator could add annotations while keeping different user annotations separate. If a third expert wished to contribute, they could only append their annotations to the second expert’s annotations.

Another limitation was the lack of security around the system. Any user could view any exploration path and, while they could not edit the comments made by the initial expert, they could overwrite comments made by other users. A method for user authentication and granting of permissions would be beneficial.

As a tool for providing guidance to a novice however, there would no requirement for the option to contributing annotations from many sources. An expert could create an exploration path guiding novices through the path in order to provide
some insight into data which they may have otherwise found overwhelmingly complex. The lack of security would also be less of an issue as this would simply be a guide for novices, not a means for collaboration.

Given that the main purpose of the framework is to facilitate collaboration and guidance, the limitations around the collaborative aspect of the framework are a clear indication that further development would be required for the tool to be marketable.

7.1 Future Works
It is clear that a system which facilitates the sharing of expertise through collaboration and guidance over coordinated visualisations would greatly facilitate data analytics. While the goals set out for this project were realised, there are still a number of features which would need to be implemented before it could truly be a powerful tool.

As mentioned above, a method for user authentication and granting of permissions would need to be implemented in this system. It would restrict which users have access to the explorations and preventing malicious interference with collaborative work, or even guidance systems. Currently, while only the author can edit their annotations, any other user can edit the other annotation field. Users would also feel more comfortable using a secure system; researchers may wish to keep their findings confidential.

There is no method implemented for sharing an exploration path with another user. A replay is revisited by remembering the name of the replay. While the name could be shared with a collaborator or novice, a more sophisticated method should be implemented, such as granting another user permissions to view and edit the exploration path and inviting them to visit it via email. This again strengthens the argument for implementing a user authentication method.
Research groups may be made up of more than 2 collaborators; therefore, the option to have more than two experts commenting on the exploration path should be available. One issue with this is that the more space is allocated for collaborators, the harder it is to view the comments with the visualisations. That said, the system could be developed to adjust its interface, depending on the number of users.

The implementation of a “Previous” button would greatly improve user experience. During exploration paths, it is highly possible that a user may want to rewind the exploration path by several steps without having to reapply the full exploration path. However, the latency caused by reapplying the full exploration path at each step while rewinding the Replay by a number of steps, may dampen the user experience more than having to replay the exploration path from the start.

Users may benefit from the option of changing their mind with regards to cloning a deviation from the exploration path to track a new exploration path. Users may choose to pause the system only to discover a relationship of interest during the paused replay mode exploration. This could be feasible by tracking the deviated exploration path in a table specifically for temporary records; however, some issues may arise with the maintenance of the current filters table should the user choose to clone the replay. It may require the maintenance of a temporary current filters table as well.

These are just a few of many possible improvements which could enhance user experience. ...............
Bibliography


