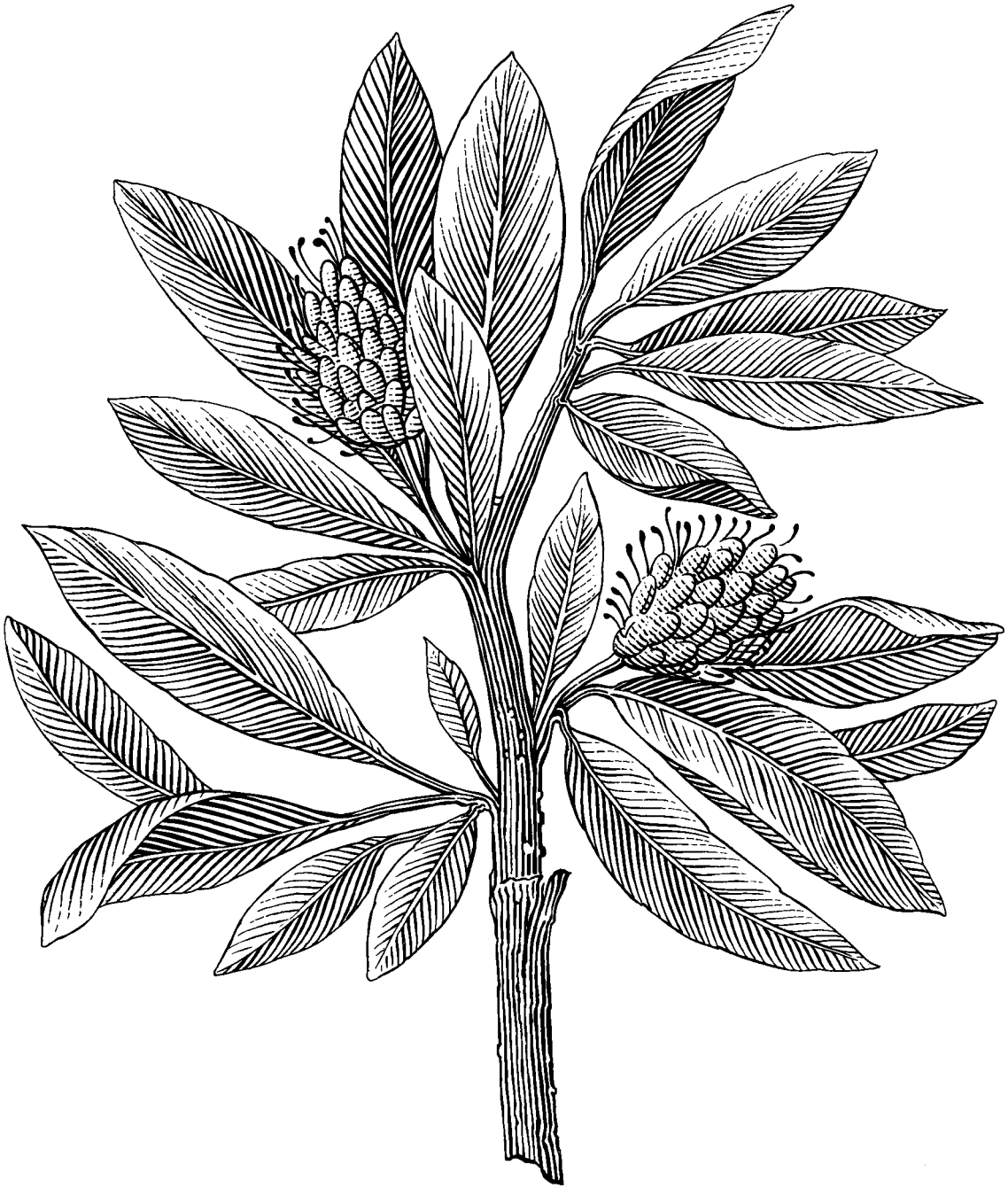
Linnæus University
School of Computer Science, Physics and Mathematics

Degree project

Visual Analysis of Author Impacts and Bibliometric Data



Linnæus University
School of Computer Science, Physics and Mathematics

SE-391 82 Kalmar / SE-351 95 Växjö
Tel +46 (0)772-28 80 00
dfm@lnu.se
Lnu.se/dfm

Author: Björn Kostkevicius
Date: 2012-01-11

Subject: Computer Science
Level: Bachelor Thesis
Course code: 2DV00E

Abstract

This thesis is about the visual analysis of author impact and other bibliometric data such as an authors publication history. It utilizes Publish or Perish as a data source, which is a search tool to find this bibliometric data. Bibliometric data is a concept within Bibliometrics with which to find and define notable publications, to draw a number of different conclusions, such as how much impact an author has had in a given field. To do this we use information visualization techniques. Information Visualization is a field of science about increasing insight and understanding of raw data. It does this by researching on details of human cognition and perception and how data itself is modeled, and by categorizing and developing new ways to encode and interact with data visually.

Since Publish or Perish only gives its information as a raw text feed, and do not allow for any real comparisons between authors, this thesis tries to rectify this by introducing a web based visualization tool to analyze this data. The data itself consists of a number of scientifically defined indexes which measures an author's impact in his given field, an overview of his publication history, and some general bibliometric data.

Keywords: Information Visualization, Bibliometrics, Publish or Perish

Acknowledgements

I would like to thank my supervisors prof. Andreas Kerren and Illir Jusufi for their advice and guidance trough this thesis. I would especially like to thank them for useful meetings which gave a good direction and helped motivate me and for setting up a good subversion repository around which to base my project. I would also thank my friend Johan Nygård for continuous advice and support even before I had selected a thesis project and ongoing while I worked with it.

Table of Contents

List of Figures

1 Introduction

This chapter introduces the problem, gives its motivation and the goals for the project. and outlines the structure of the thesis.

There is a huge amount of literature produced every day [3] In fiction or nonfiction alike as modern technology has made it vastly easier to both write and publish literature. This has had several advantages, but it has also meant that competition to get noticed never have been fiercer. There is a huge interest in tools to allow for the analysis, selection, and retrieval of information. Search engines such as Google or Bing compete to see who can best dissever the information of the Internet. But there is also room for smaller, more specialized instruments, such as the Web of Science, or Google Scholar, which is a collection of databases and a search engine focusing on Academic Publications. These instruments are further refined by third party applications. Publish or Perish is an application devised to obtain and calculate the bibliometric data of various authors the user might wish to analyze. Bibliometric data consists of scientifically defined indexes to indicate an authors impact on a field [3] as well as more general data. The application attempts to use this to help the user make a judgment of how important the author is in the field of academics. The Publish or Perish application only provides the data in a raw text format, which makes it difficult to compare different authors, see similarities or differences, as well as for a researcher to show off her own work, arguing for why she has had an impact in her chosen field. The problem then is how to best implement an information visualization of the bibliometric data of Publish or Perish.

1.1 Motivation

The motivation for this thesis can be found by first verifying the importance of academic impact, then that the data provided by Publish or Perish is useful in measuring academic impact, and lastly by verifying that visualizing this data provides an actual enhancement of the basic function. For the first the need for measuring academic impact is continually demonstrated in *Bibliometrics and Citation Analysis* where De Bellis [3] goes trough the foundations of the modern field of Bibliometrics, noting the necessity of finding effective ways to search and retrieve notable works as well as the difficulty to properly measure what works should be considered notable.

For the second it should be noted that the core information retrieved by Publish or Perish, the Citation indexes, where developed by experts in the fields of information retrieval. The initial idea was first formed and developed by Eugene Garfield, who is one of the founders of Citation Analysis as it is known today [3]. His research where later further developed by such people as Hirsch and Zheng [4]. The basics of Citation indexes and their claim to be a powerful indication of an authors research impact will be further discussed in Section 2.1. The data source Publish or Perish utilizes to obtain this data is Google Scholar, which is a specialized search engine developed by Google for the express purpose of retrieving academic articles and data about them [6]. It will be touched on in Section 2.1. This should suffice for now in verifying that the data of Publish or Perish have some legitimacy.

Lastly, the capacity of visualization to increase insight and understanding, assisting in the analysis of data, is a well defended concept. Colin Ware [2] motivates the field itself, connecting it to the cognitive sciences and claiming it as a field that still have many problems to be solved, and Robert Spence [1] outlines quite a few examples of Information Visualization having a real measurable importance in everything from hospital reforms to airplane construction [1].

1.2 Goal

The aim of this thesis project has been to construct a web based visualization tool for academic publications, using the data derived from Publish or Perish. This tool should not only visualize the data from an author, it should also provide an overview of that authors publication history, and access to some of that authors work. This tool has been named Poppry.

1.3 Specifications and Restrictions

The application have a few basic restrictions as well as specifications. The basic restrictions come in obligatory and recommended, and the specifications define core functionality and extended functionality. Basic restrictions are those which are obligatory. Recommended are those that could, if an alternative were found, be replaced. The core functionality is the most important functionality. If possible, it should be implemented. The extended functionality are things that while they might be useful are not as important. The division is as follows.

Core functionality:

- Parse the data given from Publish or Perish into a JSON format.
- Visualize the bibliometric data given from Publish or Perish.
- Compare two authors bibliometric data.
- Provide an overview and references to authors publication history, as derived from Publish or Perish.
- Reference the abstracts and papers for an author given from Publish or Perish
- Ability to filter and highlight specific parts of the data.

Extended functionality

- Extend visualization to accommodate more views.
- Compare more than two authors
- Integrate references to papers and abstracts into the web based tool itself.
- Add in a function to export visualizations.
- Compare two or more authors publication histories.

The obligatory restrictions are that the application should visualize bibliometric data derived from academic publications, with a focus on author impact. Further the application should be accessible on-line. The recommended restrictions is to use Publish or Perish as the data source and Protovis as the graphic library to paint the visualizations. By extension this means the recommended language for the application is Javascript, as that is what Protovis uses.

1.4 Thesis Structure

We have defined the problem, the goals of the thesis, and its motivation. As this is a bachelor degree thesis report with an engineering approach, the main method in solving the problem is in the construction of some form of system, in this case a visualization tool. In Chapter 2, Background, we will discuss useful background information to give the reader a context for the rest of the thesis. In Chapter 3 we will describe the tool: how it was structured, what it does, and how it was constructed. In Chapter 4 we will summarize the document, compare the application to the original problem, show what criteria it fulfills and how it fulfills them, and discuss possible future work which might be undertaken by other thesis projects.

2 Background

This chapter is divided into two sections. The first goes trough useful background information which this thesis builds on. It takes up general information about the data used. The second section, Technical Background, gives a small summarization of important fields of knowledge such as Information Visualization, Javascript, and Protovis.

2.1 Domain Background

Publish or Perish

Publish or Perish is an application created by Anne Wil-Harzing. It allows for its user to search for bibliometric data on authors. It connects to Google Scholar and derives its data from there, which it then uses to calculate a variety of bibliometric data. It has the stated goal of assisting individual researchers in making a case for their research impact, to help them show how important their contribution to their field has been. It does this by providing bibliometric indexes and other related citation based data. In general, an author with high citation metrics have had an impact in his fields, while an author with low may or may not have had one [4].

The data derived from Publish or Perish consists of two sets of data: *statistics* and *result*. *Statistics* contain the bibliometric data which is the most important to Poppry. This includes indexes defined specifically to help measure the research impact of an author as well as more general data such as name and average cites/paper. *Result* contains a summary of the authors publication history. This set consists of a list, containing all articles from the queried author which could be found. Each item in this list, or each article, consists of eight attributes. Both *statistics* and *results* will be outlined in Subsection 3.1.1. One note of importance is that *results* do not contain the queried for authors name as a separate attribute. Rather it is contained within the “authors” attribute, which represents all authors of the article. This means that to compare the *result* from different authors it is necessary to either connect the *result* to a relevant *statistic* set, or parse out the different authors from each article, add up the occurrences, and select the one with most occurrences. The first puts more responsible on the user, the second is inaccurate. In Poppry the first solution is used. While *statistics* is fairly useful in comparing authors, and its data can be used directly, *results* is easier used when drawing conclusions about a single author. Both these sets of data can be derived into a CSV (Comma Separated Values) form, allowing them to easily be parsed into various applications. In fact this is the express intent of this function [4].

Google Scholar and the Web of Science

Google Scholar and the Web of Science are both sites on the Internet specializing in providing bibliometric data to its users, mainly for citation analysis. The Web of Science was based on Garfield's old work in citation analysis, combining his citation indexes with scientrometric evaluation tools [3]. The basic idea of Web of Science is to provide interesting parties, researchers, students, academics and companies with easy access to high quality citational databases. They provide a wide variety of products and services and generally lives on their reputation for quality [8]. The downside is their services requires a paid license and is thus limited to being used in ways that might pay these licenses. Google Scholar offers free access to academic publication and citational data, as well as access to its API allowing easy use by third party applications such as Publish or Perish which in turn can be supplied as free software. The disadvantage of Google Scholar is its relative youth. De Bellis [3] especially notes that Google Scholar simply have not been around for long enough to form a definite statement either recommending it or not. He still notes that it is one of the major competitors against Web of Knowledge. Publish or Perish defends its use of Google Scholar, citing its availability and comparing it against the Web of Knowledge. It asserts that the disadvantages of Google Scholar either

do not apply to the data gathered by Publish or Perish, does not give rise to a grave enough error, is more favorable than similar disadvantages of the Web of Knowledge, or do apply but is out-weighted by its advantages [4].

Bibliometrics

The field of bibliometrics is the field of information measuring and retrieval. It goes back a long time and have only become more important over the past century as the set of available information have exploded. Bibliometrics is the science of retrieving relevant information, such as notable works within various fields of literature, be they factual or fictional. The main part in bibliometrics this thesis will touch on is Citational Analysis and in specific the Citational Indexes. Citational Analysis is a technique in Bibliometrics where the research is based around the measurement of Citations. From these citations a number of conclusions can be drawn, including gaining an idea of an authors impact. This is the basic idea of citation indexes which uses the citations a research paper receives to gauge that papers importance [12]. The citation indexes caught on during the 1960's with the work of Eugene Garfield and have been very important in a variety of research fields. In this view the citation of a research paper form an intellectual and conceptual link to the paper it cites. It can also be seen as a form of currency, each paper paying heed to relevant articles, giving them more importance [3]. Trough the years a number of citation indexes have been developed. The following indexes are calculated by Publish or Perish and used in their *statistics* dataset:

Hirsch's h-index (h-index): Was first proposed by J.E Hirsch in 2005 and has become very popular since. It defines itself as h for an author if that author has written at least h papers who have received at least h citations, and no other papers the author has written have received no more than h citations. Its intent is to measure the cumulative total impact of an authors research output. It has a number of variations and alternatives included in Publish or Perish to complement it.

Contemporary h-index (hc-index): This alters the original h-index to let age play a role in how important a citation is. It does this by adding an age related weight to the number of times a citation is counted, which essentially defines its worth. By default the citation is multiplied by 4/(number of years since publication) meaning a citation published in the current year is counted four times, a citation published 4 years ago is counted once, one published 8 years ago is counted 0.5 times, and so on. This numbers depends on the current instance of the contemporary h-index, but they are the ones Publish or Perish uses. It was proposed by Antonis Sidiropoulos, Dimitrios Katsaros, and Yannis Manolopoulos in 2006.

Individual h-index (hI-index, Original): This version of the h-index attempt to lessen the result of coauthors impact by dividing the h-index by the average number of authors in the articles who's citations are checked, bringing forth a clearer view of individual research output. It was proposed by Pablo D. Batista, Monica G. Campiteli, Osame Kinouchi, and Alexandre S. Martinez in 2006.

The normalized individual h-index(hInorm-index, PoP variation): This variant of the individual h-index first divides the citation rations of each article by dividing them with the number of authors for that article, then calculates the normalized individual h-index. This is mentioned to be more fine grained than the original, and is an attempt to give a fairer view of the individual research impact.

Multi-authored h-index (hm-index): This is also a method to measure individual research impact, though rather than an variation of the individual h-index it was proposed in its own right. It uses fractional paper counts instead of reduced citation rates, and determines itself using unaltered citation counts in combination with these altered paper counts.

Age-weighted citation rate (AWCR): This index is a variation of the AR-index, which was developed to complement the h-index. The AR-index is defined as the square root of the sum of all age-weighted citation counts over all papers that contribute to the h-index. In the AWCR variation all papers are allowed to contribute however, allowing early and less cited papers to contribute to the AWCR even if it does not contribute to the h-index proper. The original AR-index was proposed by Bihui Jin in 2007.

AW-index: The AW-index is the square root of the AWCR index, designed to allow for easier comparison with the h-index.

Age-weighted citation rate per author (AWCRpA): This variation of the AWCR index normalizes the number of authors for each paper.

Egghe's g-index(g-index): This variation of the h-index try to improve it by giving more weight to highly cited articles. It is defined as follows: *" [Given a set of articles] ranked in decreasing order of the number of citations that they received, the g-index is the (unique) largest number such that the top g articles received (together) at least g^2 citations."*

Zhang's e-index (e-index): The e-index tries to check for different citational patterns in authors with similar h-indexes by "*calculating the square root of surplus of citations in the h-set beyond* $h^{2.}$"[13]

2.2 Technical Background

Information visualization

Information Visualization is a field of science, detailing the illustration of data, in an effort to ease insight and understanding, affect the viewers opinion on some issue, and allow for truly huge amounts of data to be easily understood and worked with [1]. The term visualization is defined by Colin Ware [2] as a "*graphical representation of data or concepts"*. He further defines the advantages of information visualization as an ability to comprehend large amounts of data, ability to notice properties not readily apparent in the data, ability to note errors in the data itself, the ability of visualizations to ease understanding of data whether the data itself is large or small scale, and finally the ability to help creating a hypothesis [2]. The term visualize is defined by Robert Spence as *"to form a mental model or mental image of something"* [1].

The act of visualizing data is often defined as a process. Colin Ware defines four basic stages, with a possibility to call back to earlier stages. The four stages consists of gathering and storing the data, then preprocessing it into something understandable, then displaying it, and finally the fourth step consists of the human perception and cognition of it. The human user in the fourth step can call back to the first step, collecting and sorting, by gathering more data or changing the data used. He can call back to the second step, the preprocessing and transformation of the data, by exploring it, perhaps altering what part of it is to be focused on and highlighted. Finally, he can call back to the third step, the displaying of the data, by data manipulation, changing how it is drawn [2].

Robert Spence [1] divides information visualization into three categories. The representation of the data, the presentation and the interaction. This can roughly be mapped to the steps above. The

representation of the data consists of how to encode the data, and can be mapped to the preprocessing and transformation step as well as the displaying step. The presentation of the data consists of how it is displayed and in what environment, and can be mapped to third and fourth step, the display and the human user. The last category, interaction, can be mapped to the human users callbacks. Lastly Ward, Grinstein and Keim [11] define a number of five step transformation pipelines, which takes data and transform it into a visualization. The important part here is that information visualization is defined as a process with an *active* user. The user is active because he interacts with a graphic visualization of some form of information. The interaction in itself serves to refine a mental model the user has of the information in order to give him more insight about it [1, 2, 3, 11, 18].

The representation step as defined by Robert Spence consists of how to encode the data. A number of visualizations which can readily encode various types of data have been developed over the years. Things such as scatter plots, Chernoff faces, hyperbolic trees, star plots, tree-maps and many others are all good example of classic visualizations. To give an example of a successful visualization which has had a large impact we here present the London Tube Map as developed by Harry Beck:

Figure 2.1 : The London Tube Map. Taken from [16]

The leftmost image is an old design. The middle is Harry Becks revolutionary version. Both these are only a few months apart. The main point of Harry Becks design is a realization that the exact geographical locations of a subway network is irrelevant to the user, and the real important information is how the stations connect to each other and what area they are located in. This design have come to see widespread use in any city with a large scale subway network. Other famous information visualizations it is recommended for the reader to look up includes Minard's Map (of the French invasion of Russia in 1812) and Florence Nightingale's Rose Diagram.
The main addition the computer age have given to the information visualization field is the current ease of interactive visualizations. From something as fundamental as scrolling, to something as intricate as database exploration tools, matters of interaction have never been more important [1].

JavaScript

Javascript is a client side scripting language for HTML. HTML is a way to define what is in a web page, a language for filling web pages with various content. Javascript is designed to be used in conjunction with HTML, to allow for dynamic content or content that changes. It is client side because it executes on the client side, or the users browser as opposed to on the sites server. It is a scripting language because it changes the behavior of the browser. JavaScript have no connections to Java whatsoever. Javascript is a loosely typed language. This means that Javascript allows quite a few things

that stricter programming languages would not. As an example, Java is very strict in regards to the type of any variable being defined. It is necessary to always define the type when naming a variable, when defining the type of of whatever arguments which are passed in a method, as well as the type of whatever variable is returned, including void. Javascript does not have this. A variable name is set by typing `"var"` and then the name itself. In fact, it is not even necessary to use var. Doing an operation such as `"exampleVariable=true;"` works. Javascript first assumes the code is attempting to use a global variable and if that is not the case creates a local one. Not using var is highly discouraged, as it inevitable lead to accidentally using a global variable. Another example would be the visibility of methods and variables, private, public and protected. Javascript does not have this.
The advantage of this is flexibility. Javascript can do quite a few things stricter languages normally are not able to, such as using functions as objects in themselves. The disadvantage is that things very easily get sloppy, hard to read trough, debug, and parse.

Protovis
Protovis is a visualization rendering library for the JavaScript. It allows for the easy construction of very sophisticated forms of visualization, by allowing for the manipulation of basic shapes such as lines, rectangles(bars), and dots. It calls these shapes marks, each mark having a number of properties which is used to define the mark's behavior and how it is rendered. It uses what the creators, the Stanford Visualization group refers to as a declarative style of programming. What this entails is that rather than write out instructions for how to render a piece of graphics, the programmer declares what to render in what specific ways. This means that Protovis automatically handles a lot of things which would normally have to be explicitly printed out such as iteration trough arrays. It does so by taking full advantage of JavaScript being a loosely typed language. Among other things it allows for function chaining, functions to be passed as properties to marks, marks to inherit the properties of parent objects, and attaching event handlers to marks as properties [7]. Protovis will be discussed closer in Subsection 3.2.1

Types of data
There are a number of concepts for how to consider data that are helpful when working with visualizations. First, data can be divided into either an entity or a relation. A data entity is a set of data which represent some form of entity. The *statistics* dataset of Publish or Perish can be seen as an entity representing an authors academic impact. A relation is a set of data which represent a connection between two entities, and is not that important for this thesis. An entity or relation consist of attributes, which are different types of data. As an example the *statistics* dataset have several attributes which represents everything from bibliometric indexes to the authors name. When talking about attributes it is these types of attributes that is being referred to. When talking about datasets it is either the Publish or Perish *statistics* or its *result* dataset that is being referred to. The number of attributes an entity has denotes the dimensionality of the data. It is possible to have univariate, bivariate, tri-variate or multivariate data depending on whether there is one, two, three or more attributes [1]. The *result* and *statistics* datasets both consist of multivariate data. They will be described closer in Subsection 3.1.1

3 Design and Construction

This chapter explains the developed system, how it is designed and gives an overview of its source code. The first Subsection, 3.1 discuss how the application was developed and how it can be used. It will describe how to input the data, the parsing, the visualizations, and what interactions are possible. Subsection 3.2 describes implementation, outlining its architecture and the structure of its classes.

3.1 Method and Design

This thesis takes an engineering approach. This means that the main goal of the thesis is to create some system, and then demonstrate this system and judge how well it solves the stated problem. Therefore the main part of this project has been in the creation of an application with which to visualize the bibliometric data of Publish or Perish. This document exists to demonstrate this application and show how it visualizes this data. This section will introduce the basic framework of the application and describe how it is designed.

At the start of the project, the working process was largely undefined. Requirements were updated in an ad-hoc, iterative manner, no specific system model were used, and time was not managed. A lot of time was spent on designing the look and feel of the site, learning script based web development, and trying to grasp the needed architecture of the site. Later the working process would be improved. A ToDo list would be kept with short term tickets, the architecture would be formalized using UML, and the requirements were updated after talks with the supervisor of the project. Eventually a summary and overview of system Use Cases were designed. A formal system model were never used, but informally the site was modeled using a non architectural approach. As such the system was modeled using a wide variety of views. Using these, the requirements, architecture, input/output, GUI, and necessary operations were modeled, as well as a conceptual system design. Requirements can be found in Subsection 1.3, and a discussion of how they were fulfilled can be found in Chapter 4. To obtain an overview of the sites architecture, Chapter 4. Implementation can be used. To obtain an overview of the GUI views, Subsections 3.1.3 and 3.1.4 can be used. Subsection 3.1.4 also provides an example of how the site can be used, as well as an overview of the systems Use Cases. To obtain a view of how the input and output was structured, see Subsection 3.1.1 and 3.3 respectively. Fig. 3.1 provides an overview of the project solution. The requirements formed the basis for all other views. These were processed into architecture and GUI design respectively. The conceptual design consisted of defining specific areas of responsibility, decide how to encode and format input data, and decide what visualizations to provide as output respectively.

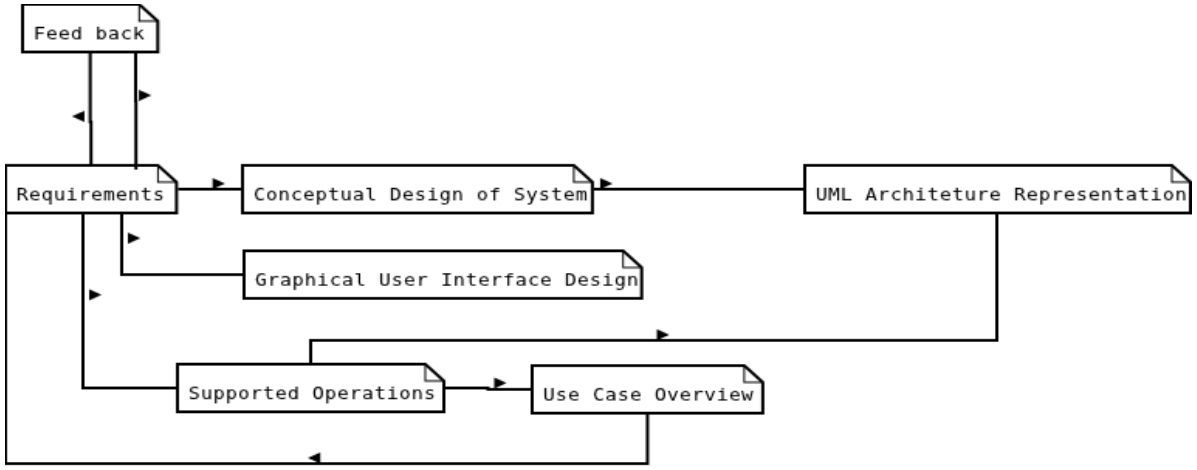


Figure 3.1 : System Solution

The requirements were continually updated during the project, and used to form the conceptual design of the system. A number of important design decisions happened at this step. The application was divided into three conceptual parts, the input, parsing of the input, and output. Later interaction was added. They will be discussed in greater detail later in this section, starting at Subsection 3.1.1. The first important decision was to select Publish or Perish as the data source. There were a number of

reasons for this. Publish or Perish already came with advanced algorithms for searching and deriving the data from Google scholar, something that would need to be implemented if Google scholar were to be used directly. In addition Publish or Perish allowed for easy formatting of the data into standard CSV based strings. Third it was completely free to use. Lastly because the dataset is based on Google scholar, the published articles used in obtaining the citational data are fairly comprehensive. What this means is that the data that can be searched for is not limited to a specific field of research or a specific geographical location. Though biases against English based articles naturally exists that would be true for most such data sources [4].

The disadvantage is that Google Scholar in itself is a comparatively young search engine, which was noted in Section 2.1. A more critical disadvantage is there is no easy way to integrate the raw data into the application as it is dependent on Publish or Perish. It is possible to derive the data, but only as CSV strings, and even that requires manually copying and pasting. A solution for this might have been found, but was not due to the second design decision. This consisted of putting the input on continually low priority. The reasoning behind this was that the vast majority of the requirements where not actually about obtaining the data. The problem as formulated was about the visualization of the data, and it was decided to focus on this part. As such the majority of the implementation was concerned with the parsing and the output.

After shifting the focus from the input to the parsing of the data and the output, the initial visualizations were designed. There where four initially. A parallel coordinate plot, a star plot, a line-chart, and a customized form of bar chart, each visualizing one specific type of data. On the recommendation of the supervisors the parallel coordinate plot was given highest priority, the star plot was dropped, and the line-chart and bar chart were both put on low priority in that order with the bar chart at the lowest. The reasoning for this was that the data visualized by the star plot, a subset of the authors *statistics* dataset, could equally well be visualized by the parallel coordinates plot. The reason to put the parallel coordinates plot at the highest priority was that the data it was supposed to visualize was the most critical one for the original intent of the application, visualizing academic impact. The data it was supposed to visualize included the citational based indexes of the Publish or Perish *statistics* dataset, which are the ones designed to measure academic impact [3, 4]. The reason the bar chart was put on the lowest priority was that the data it was supposed to visualize, a specific authors coauthors, fell close to another information visualization thesis and also did not actually measure academic impact. Lastly a need to connect the visualizations to each other was emphasized, and also to the need to allow for a file based input. In the end, these requirements were amended to the original.

Amended requirements

Core

- Create a parallel coordinate visualization to visualize the *statistics* dataset.
- Connect the visualizations to allow a change in one to be displayed in the other.
- Allow for file based input.

Extended

- Create a line chart visualization to visualize an authors publications per year, derived from *results*.
- Create a bar chart visualization to visualize what authors an author has cooperated with on what articles.

The visualization requirements could be seen as either more specified versions of the original requirements, or as added requirements in their own right. Finally the name of the application was decided on. Poppry, derived from Publish or Perish, Protovis, and the flower Poppy. The following subsections will go trough the design of the applications input, parsing, visualizations and interaction. It will describe how to use the application and show how it looks like.

3.1.1 Input

The input Poppry requires is that of either the *statistics* or *result* dataset from a Publish or Perish query on author impact. *Statistics* represents an authors bibliometric data while *result* represents an authors publication history. They were touched on in Section 2.1. There are three different ways for Poppry to use these raw CSV datasets. Test input, file input, and GUI input. Test input consists of a number of *statistics* and *result* that are already included in the site. They are read into two arrays using AJAX which is a Javascript technology to read files [17]. The files used are located in a folder supplied with the application. These datasets are used for tests and during development, and can also be used to demonstrate the application. File input accepts input trough accessing and reading files the user submits. It uses a FileReader element which will be discussed in Subsection 3.2.1. It is possible to input any number of files though at least two must be included for each author. One for *result* and one for *statistics*. In addition it is very important in which order the files are submitted. The *statistics* of an author must be submitted directly after his *result* is submitted. When creating a folder with files to submit it is therefore recommended to name them something like <name of Author1>, <name of Author1><S>, <name of Author2>, <name of Author2><S> and select them in alphabetical order. Due to limitations of the software elements used in the file input it is not possible to select a directory or a directory structure, only files. It is possible submit the entirety of a folder by selecting all files in that folder. GUI input allows the user to input data from Publish or Perish trough GUI text elements. This is done by first clicking the button for show/hide input fields which creates and appends the necessary elements to the site. The user can then to copy and paste the *statistics* and *result* from one author into the relevant text areas and submit them to the application. It is important to note that test and file input clears the data loaded into the application, while GUI input does not. With the GUI input it is however only possible to submit one author at a time. Lastly it is possible to completely clear the dataset. The datasets will now be outlined. The first dataset, *statistics*, contain the following attributes:

Statistics

Base

Author	The queried for author.

Cites

Papers	Total number of papers
Years	Years active.
Citations	Total number of citations
Cites/Year	Average number of citations per year
Cites/Paper	Average number of citations per paper
Cites/Author	Average number of citations per author
Papers/Author	Average number of papers per author
Author/Paper	Average number of author per paper

Indexes

h-index	Hirsch's h-index
hc-index	Contemporary h-index
hI-index	Individual h-index (PoP variation)
hI-norm	The normalized individual h-index
AWCR	Age-weighted citation rate (AWCR)
AW-index	AW-index
e-index	Zhang's e-index
hm-index	Multi-authored h-index
g-index	Egghe's g-index

It was quickly decided to divide *statistics* into three conceptual sets of data. The indexes data and a more general cites data, as well as the basic data consisting of the authors name. The reason for this was that it was initially planned to make one visualization for each divided dataset. This was later scrapped because it was unnecessary and counterintuitive to connecting the visualizations to each other. The division still helped while thinking of the data. The second dataset is called *results* and contains a summary of the queried authors publication history. This summary takes the form of a list of articles, each of which consists of eight attributes. These are the attributes:

Results

Numeric Data

Cites – The cites of the author.

Originating from

Authors – List of the authors who wrote this article.
Title – Name of the paper.
Year – Year of publication.
Source – Where the article was found.
Publisher – The articles publisher.

Sources

ArticleUrl – An url to the article itself.
CitesUrl – An url to articles which cites it.

3.1.2 Parsing

After we have obtained the raw CSV strings trough the input, we need to format them into an appropriate data structure to feed to the visualizations. The data structure of choice is JSON. JSON stands for JavaScript Object Notation and is the syntax for how JavaScript defines arrays and objects [17]. It is also an acceptable format for Protovis [7]. After the CSV strings are parsed to JSON arrays they can be given as an array to the Protovis visualizations to use to draw and render the visualizations.

3.1.3 Visualizations

Two main visualizations were developed to show the data. A parallel coordinate plot and a line chart. These are both classic visualizations. The Parallel coordinate plot encode the *statistics* dataset. The line chart partially encodes the *result* dataset by using it to calculate an authors publications per year. In addition to this a table presentation gives an overview of a single authors *result* dataset.

Parallel Coordinate Visualization

The parallel coordinate visualization is a classic visualization. It is an extension of the basic coordinate plot visualization, which are used as the base of everything from graphs to scatter plots. Fig 3.2 is an example of the basic coordinate plot visualization and some derivatives:

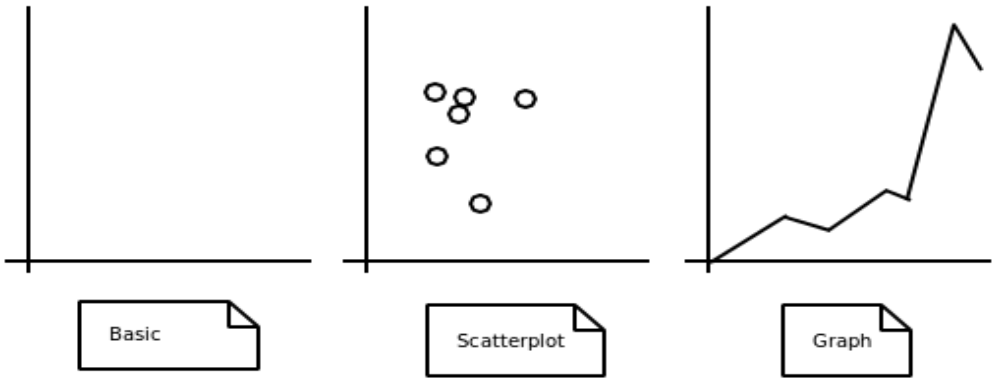


Figure 3.2 : Coordinate Plot Based Visualizations

It is very common while using bivariate or even univariate data to use this structure. It can be used to display a graph, function, form the basis of a scatter plot or any number of things. It runs into difficulty when trying to visualize multivariate data. Multivariate data is data with a large amount of attributes. *Results* is multivariate as each entry in a *result* entity consists of eight attributes. *Statistics* is also a multivariate set of data as each *statistics* entity consists of eighteen attributes. The parallel coordinate plot was designed to allow for the encoding of multivariate data [1]. It takes the axes of the common coordinate visualization and places them in parallel. It is then possible to have any number of axes, each axis representing one attribute. This also means any entry cannot be placed in between two axes, as any position only have different values in the vertical direction. The solution is to put an entry on the axes themselves. This means each entity put into the visualization will be multiplied by the number of axes, or attributes. Also if more than one entity is added there needs to be a way to see what values on different axes are connected. This is usually done by a line.

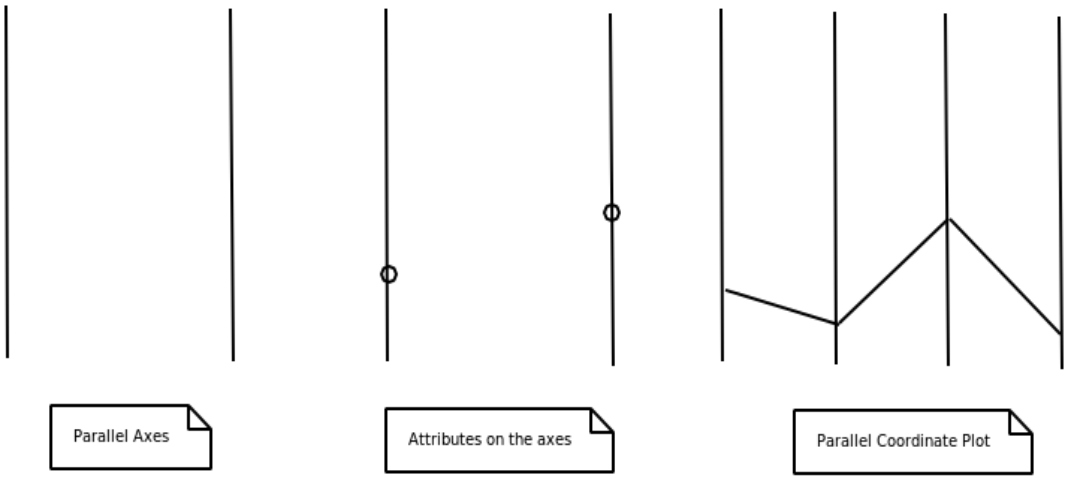


Figure 3.3 : Basic design of Parallel Coordinates

The Parallel Coordinate Visualization was firstly developed by Alfred Inselberg [11]. The one used in the application was derived from one of one of Protovis many examples [14] and is used to visualize the *statistics* dataset. It is the core visualization of the application as it visualizes author impact. It looks as follows:

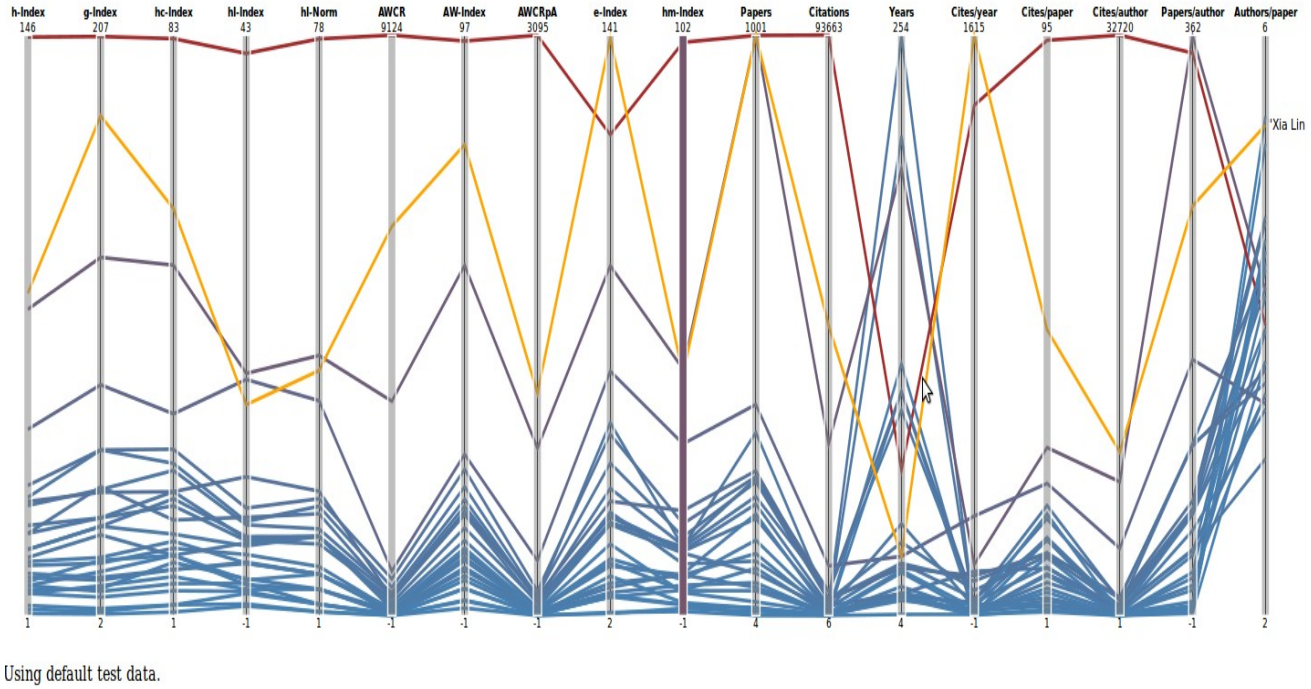


Figure 3.4 : The Parallel Coordinates Display

The Parallel Coordinate Plot shows all attributes of the *statistics* dataset, starting with the indexes. Each line represents the *statistics* of one author. Each axis has a bar scale that might be selected and scaled in the vertical direction, filtering out certain attribute ranges. The lines not within the range will be greyed out. The coloration depends on what axis bar is currently selected. In addition the visualization allow for high-lighting by moving the mouse over a specific line. This will cause it to change color and display the specific author on the right side of the visualization. A selection operator is supported by clicking on a line. This marks it both in this visualization, and in a list related to the visualizations. All such marked lines are high-lighted both in the parallel coordinate plot and in the line chart. Clicking on the line again causes it to be deselected from the visualizations and the list. The axes ranges scale depending on what values are given to it. In the cause of a single author, the axes do not scale, and the range given only have a difference of 0-1 units. This allows a quick overview over a single authors specific values.

Line Chart Visualization

The Line chart Visualization is another classic visualization. It is far older than the parallel coordinates plot. It is also based on two coordinate axes, utilizing a line to encode either uni- or bivariate data. Because it does not scale well when the data have higher dimensions we need to transform the multivariate data we are using into bivariate in order to properly display it [11]. In Poppry, the Line chart Visualization is derived from an authors *results* by calculating the number of publications an author has published each year. This is done by summing up the number of identical years in the *result* dataset. It was derived from the Protovis Line chart example [15] and is shown in Fig. 3.5.

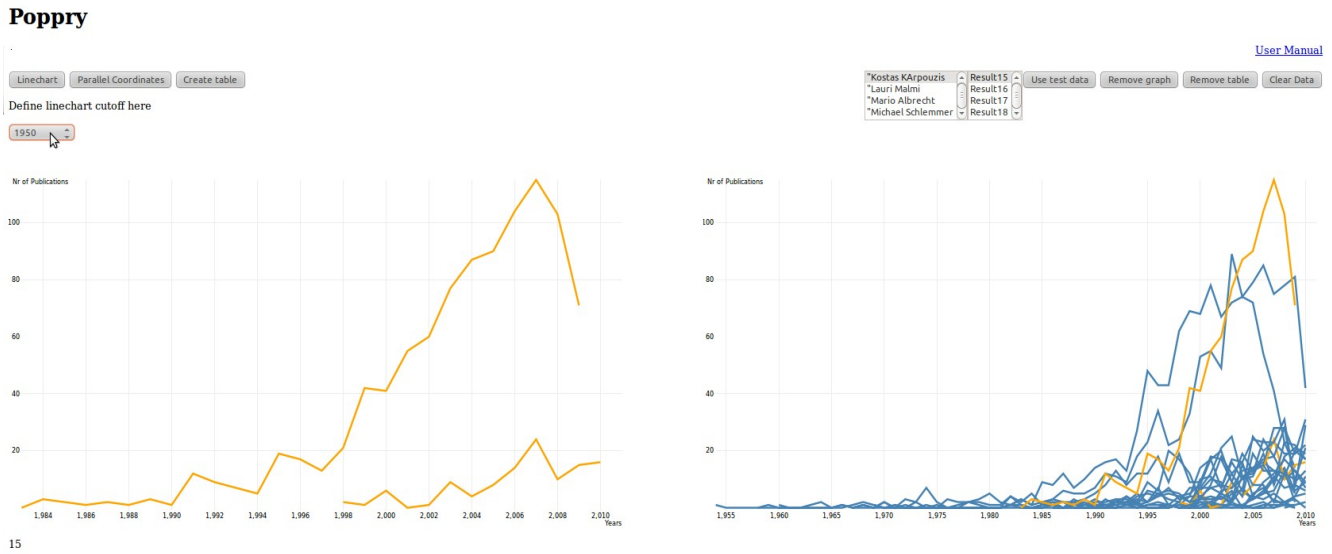


Figure 3.5 : The Line Chart Display

It is divided into two displays: the right one always displays every author in the currently loaded *result* dataset. The vertical axis encodes the number of publications and the horizontal encodes the year of the publication. The axes scales with the values. The line chart supports a selection operation either by clicking on a line in the visualization itself or by selecting it in a list in the application GUI. The left display only shows selected authors. Any selected authors will be high-lighted both in the right display and in the parallel coordinate plot. The coloration was chosen to match the Parallel Coordinate Visualization. This visualization allows for easy comparison between how many articles different authors have published. It does have some significant disadvantages though. The largest one is that because *result* does not include the authors name save indirectly, there is no easy way to know what line belongs to what author. Obtaining this data either requires *statistics* to get connected to *results*, putting the responsibility on the input and therefore the user, or trying to sum up the authors occurrences within *result* and picking out the author with the largest matches, which has the potential for errors such as an author who have co written each article with the same person. The current solution is to assume each *statistics* and *result* have been input into the application correctly. The second disadvantage of the line chart visualization is that usually the authors do not publish their first article at the same time, which makes it a bit more difficult to compare authors. Usually the variance is small enough not to matter, but if there are extreme values such as a difference between 1986 and 1961, or 1920, the way Protovis scales the axes tend to produce rather diminutive graphs. The third disadvantage is that the articles an author published in an unknown year is not shown anywhere in the visualization. The advantage of the line chart visualization is that it provides a rough overview of an author's research output, which in combination with the Table Presentation gives an overview of an author's publication history. In addition since the line chart visualization connects to the parallel coordinates visualization it is easy to compare an authors *statistics* to his *result*.

Table

The table visualization is only a graphic visualization in the most rudimentary sense. It exists to provide an overview of the unaltered *result* dataset, allowing the user to read trough an authors publication history in an easier format than the raw Publish or Perish feed. The design motivation for it was that while there was very little need for a graphic visualization of a single authors entire *result* dataset, there was a need for a presentation of an authors publication history. In addition since six of the eight attributes of *result* entries where string based, most visualization would need to derive the data from the dataset rather than use it directly. A table presentation gives a good overview of the *result* dataset and was therefore used. The example is given in Fig. 3.6.

Cites	Authors	Title	Year	Source	Publisher	ArticleUrl	CitesUrl
0	..., PS Olech, A Ebert, A Kerren	EEG-Based Measurement of Subjective Parameters in Evaluations	2011	HCI International 2011-Posters' ...	Springer	Article	Cites Url
0	A Kerren	Visualization of Workaday Data Clarified by Means of Wine Fingerprints	2011	Human Aspects of Visualization	Springer	Article	Cites Url
0	B Zimmer, D Ackermann, M Schröder, A Kerren...	Comparative visualization of user flows in voice portals	2011	Graph Drawing	Springer	Article	Cites Url
4	M Albrecht, A Kerren, K Klein, O Kohlbacher...	On open problems in biological network visualization	2010	Graph Drawing	Springer	Article	Cites Url
2	M Rohrschneider, A Ullrich, A Kerren...	Visual network analysis of dynamic metabolic pathways	2010	Advances in Visual ...	Springer	Article	Cites Url
2	..., C Heine, A Reichenbach, A Kerren...	A Novel Grid-Based Visualization Approach for Metabolic Networks with Advanced Focus&Context View	2010	Graph Drawing	Springer	Article	Cites Url
1	..., Y Dingjie, A Kerren	The Network Lens: Interactive Exploration of Multivariate Networks Using Visual Filtering	2010	2010 14th International Conference ...	computer.org	Article	Cites Url

Figure 3.6 : The Table Visualization

It lists all the articles of a *result* for a single author, sorted by year with the most recent article at the top, and also provides links to the articles the information was derived from. The article url links to the article itself. Therefore the article url for *"Visualization of Workaday Data Clarified by Means of Wine Fingerprints"* links to that article. The cites url for *"Visualization of Workaday Data Clarified by Means of Wine Fingerprints"* links to a Google Scholar based list of articles which contains cites to that article. In the case where either link is not provided in the *result* dataset, a placeholder label named *"NoUrl"* is added instead. The table uses a normal CSS file *poppry_layout.css* to define its style.

3.1.4 Interaction

This section will discuss the possible operations of Poppry and describe the basic GUI.

Poppry supports the following basic operations:

Load Test Data - Clears any currently used data in the application and replaces it with data loaded from the integrated dataset mentioned in Subsection 3.1.1

Load File Data - Clears any currently used data and replaces it with data loaded from files.

Load GUI Data - Adds a single author's *statistics* and *result* dataset to the current loaded data.

Clear all data - Clears any currently loaded data.

View Line chart visualization – Displays the Line chart visualization.

View Parallel Coordinate Plot visualization – Displays the Parallel Coordinate Plot. It is currently displayed in Fig. 3.7.

Create Table – Creates a table of a single *result* that is selected in the list element and appends it to the bottom of the page. This operation also clears any previously created table.

Remove visualization – Clears the screen from a visualization.

Remove table – Removes any currently displayed table.

Mark Author(s) from lists – Marks an author by clicking on an entry in either list. Multiple authors can be selected.

Mark Author(s) from lines – Marks an author in the visualizations by clicking on a line in that visualization.

Filter authors in the Parallel Coordinate Plot – By scaling one of the bars imposed on each axes.

Follow links from any currently selected table – Brings the user to a specific article of an author or to a list of articles which cites one of that authors articles by clicking on any of the links displayed in the table presentation.

The operations can be roughly divided into *Loading the Data*, *Viewing the Data* and *Manipulating the Data*. Together they allow the user to construct and manipulate visualizations of the bibliometric data of Publish or Perish. Poppry's user interface provides elements to conduct these operations. Fig. 3.7 shows the application while in use.

The GUI is divided into five different areas. There are the visualization views at the top left corner, the menu for data manipulations at the top right, the visualization area itself in the middle, the file input menu in the lower left, and the area for the input buttons which are currently hidden in the bottom right. The table is generated at the bottom of the page.

Loading the data can be done either by the file input menu, the GUI input menu, or the *"Use Test Data"* button. Loading the data from files consists of first preparing a folder of files for the application to use, as outlined in Subsection 3.1.1, then pressing the *"Browse"* button in the file menu.

This will display a dialog window were it is possible can select the files. It is impossible to select a directory, but it is possible to select multiple files. The files have to be ordered in the correct way, *result*1, *statistics*1, *result*2, *statistics*2, … *result*N, *statistics*N.

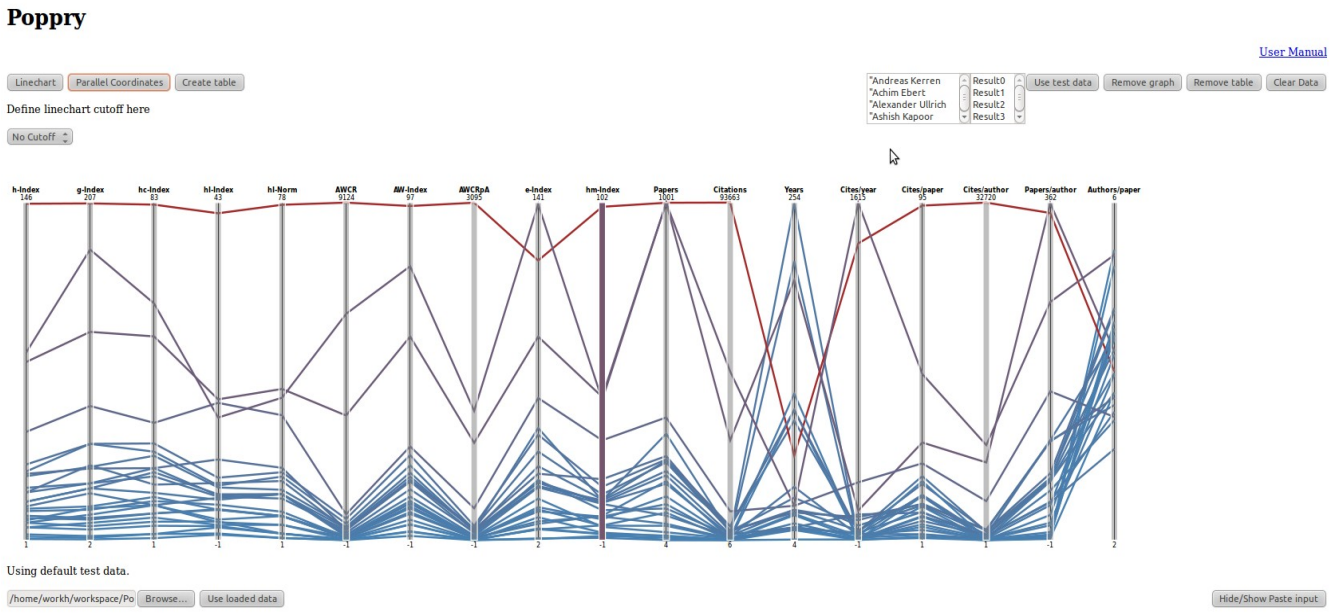


Figure 3.7 : The Poppry GUI

The titles are unimportant. After selecting the files they are read when pressing the *"Load"* button. This might take a few seconds. After the files are read they must be initiated by pressing the *"Use Loaded Data"* button which replaces the current dataset with the one derived from the files. To utilize the GUI input fields the user simply clicks on the *"Hide/Show Paste input"* button and the necessary input elements are appended to the page. Selecting any of the three visualization views displays a visualization. This visualization will be empty unless a dataset is loaded into the application. This is done either by using the test data, loading files into the application, or by copy and pasting an author directly into the application from Publish or Perish.

The lists are keyed to the visualizations. In the list menu the currently loaded data is displayed in two list elements, the left which contains all the names of the loaded authors as derived from *statistics*. Selecting any entry in either list selects the corresponding entry in the other list and also marks the corresponding line in both visualization. It is also possible to select multiple entries in the list either by clicking on multiple lines in the visualizations or by clicking on entries in the list itself in conjunction with ctrl or shift dependent on the browser used.

Everything on this page save the loading of test data is done client-side, including the file uploads. There is therefore no need to refresh the page. Fig. 3.8 and Fig. 3.9 outline the basics of the file and GUI input.

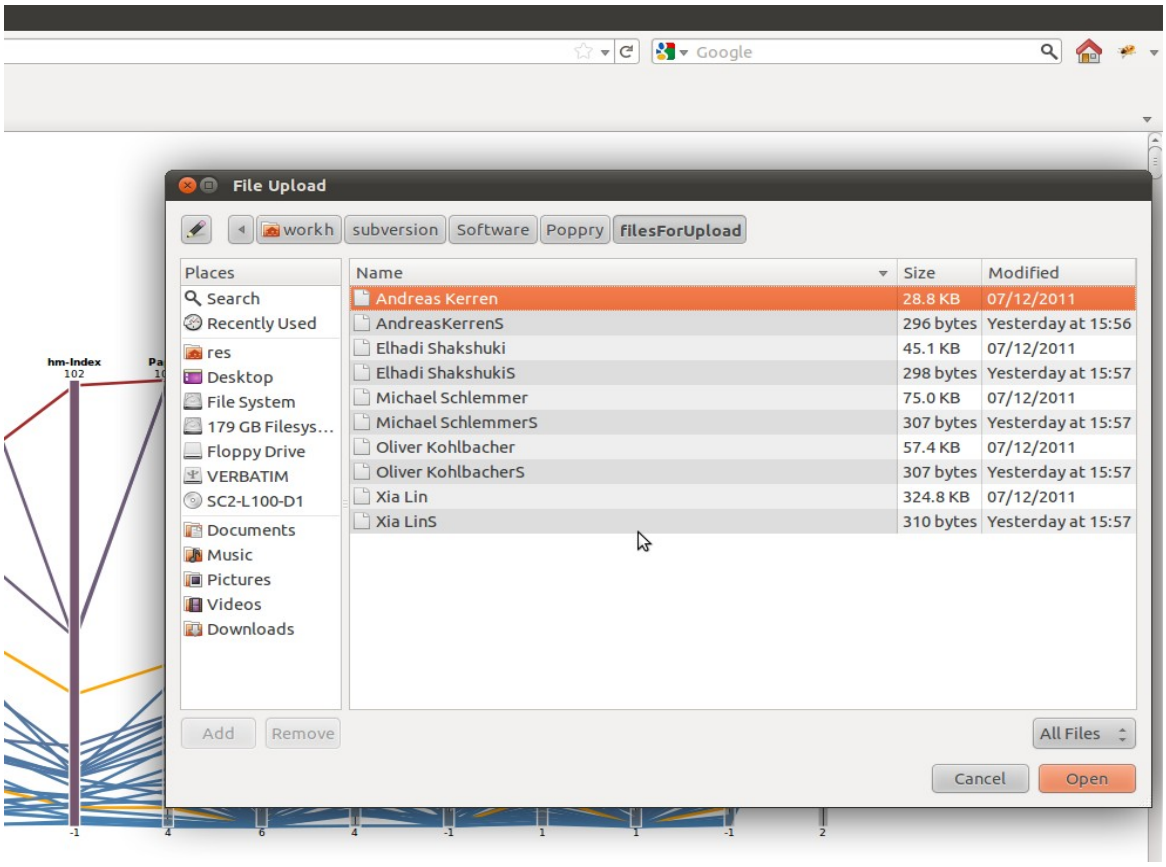


Figure 3.8 : File Input

Fig. 3.8 shows an example of how to organize the files when conducting file input. As can be seen they are ordered alphabetically. The user is able to select all files within this folder for upload at the same time.

Figure 3.9 : GUI Input

Fig. 3.9 shows the input fields for the GUI input. This allows the user to append the *statistics* and *results* for a single author to the currently loaded dataset.

To further clarify the interaction of the application a small example of how to use it as well as a use case diagram will be provided. The example will show a way a user can use Poppry to analyze a set of researchers and find out some information on their publication history. The use case diagram will show how the main operations of the application are connected.

Assume a user is interviewing a number of researchers for a job position. It is a fairly popular position and there are a lot of applicants with similar references. The user wishes to gain a clearer idea of the applicants research habits. He opens up Publish or Perish and Poppry. He conducts a search on each applicant in Publish or Perish and creates two files for each, which he places in a folder labeled "Applicants". He copies the results of his searches into these files. He then selects Poppry, presses the *"Browse"* button, navigate to the folder and selects all the files to open. He then clicks the *"Load"* button, followed by the *"Use Loaded Data"* button.

The information is now loaded into the site. He selects a few authors and creates tables for them just to check that they have been loaded correctly. He then selects the Parallel Coordinate Plot to gain an overview of the data. He is especially interested in the h-index as it is the basic index for measuring academic impact. He selects the h-index axis in the visualization and clicks next to the top of the axis, then drags down. This grabs the bar imposed on the axis and scales it vertically after the mouse. Only lines within this bar is rendered with full lines. The rest are grayed out. He releases the bar around the half way point of the axis.

As it now encloses the top half of the axis he has filtered out any applicant with an h-index lower than that. Of the remaining authors he selects those applicants who have high values in the rest of the indexes. This is done by first doing a mouse over on a line which identifies the person that line belongs to and high-lights it to make it easier to pick out. He then clicks on the line, which marks it in the list as a selected author and makes it permanently highlighted. When he has finished, he presses on the *"Linechart"* button to gain an idea how many articles the authors have produced in the last five years. The line chart visualization shows all applicants in its right display with the selected applicants highlighted. In its left display it only shows the selected applicants.

The user makes note of any particular interesting authors. Finally he creates a table for each of these interesting authors, both to see if they have done any work or cooperated with any particular author he might be familiar with, and also to check out a few of their articles in order to gain a better view of their writing style. The user makes a note to pay extra attention to these applicants. He has now gained an idea of the impact these people have had on their field, how much work they have produced recently, and how that work looks like. The user has done this by going trough the earlier mentioned categories of operations: *Loading the Data, Viewing the Data,* and *Manipulating the data.* In Fig. 3.10, we show a small use case diagram exemplifying some of these operations.

3.2 Implementation

This chapter discusses the architecture of the site, what files are included and how they function. We will also discuss relevant technologies and how they were used.

3.2.1 Technologies Used

This section will discuss some of the technologies Poppry is based on, including Publish or Perish, Protovis, and some miscellaneous techniques based on JavaScript .

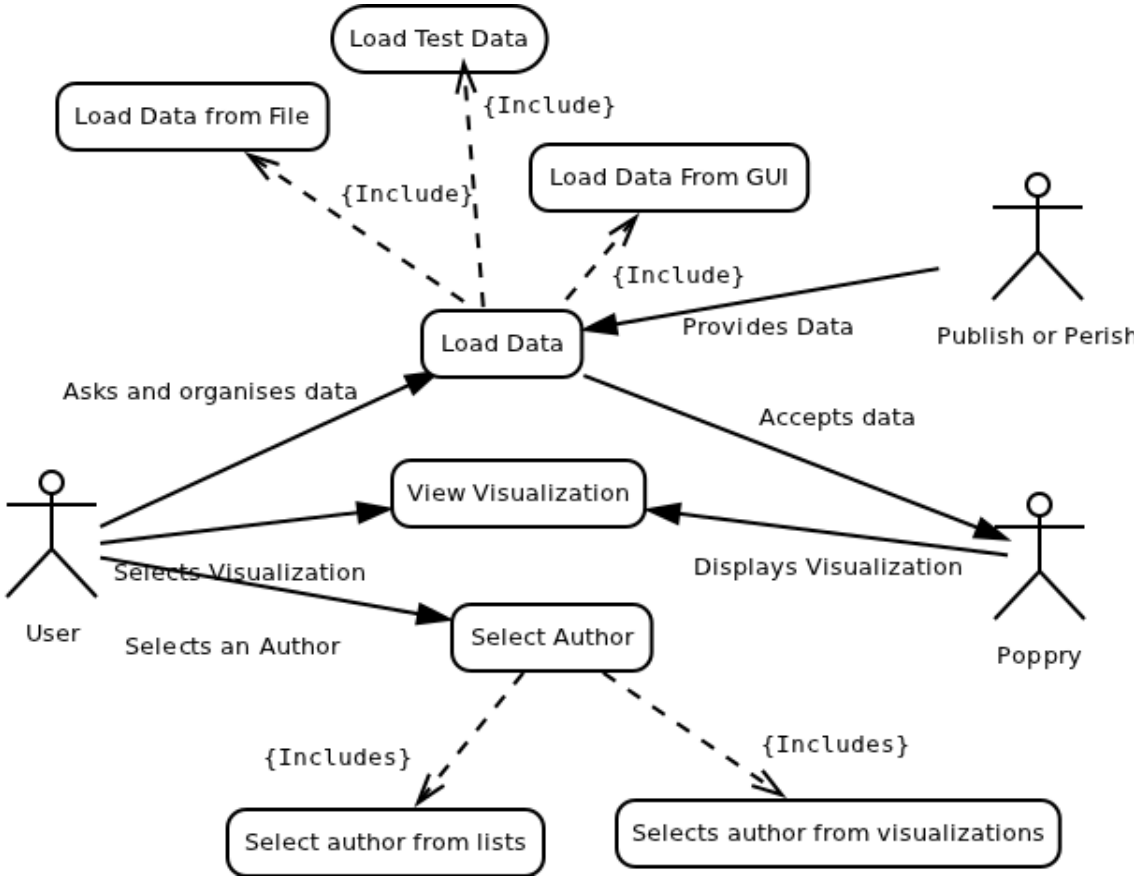


Figure 3.10 Use Case Diagram

Publish or Perish: The data source

As have been noted the application uses Publish or Perish as its data source. Fig. 3.11 gives an example of how Publish or Perish looks like while in use. It is possible to look up a specific author by inputting his name in the top field and press lookup. There are a few ways to modify the query by selecting fields or excluding names. The result of the lookup is displayed in two fields. The topmost field, the one starting with Papers:104 and somewhat confusingly labeled "Results" is the authors *statistics*. The second which contains the list of articles is the actual *result*. Publish or Perish stores any previously queried author in itself for a limited amount of time. This allows the user to build up a collection of often queried authors which the user can access at any time trough the multi-query center. Lookup Direct is used when it is suspected there might be newly released data. This circumvents the stored data and looks it up from Google Scholar directly. Publish or Perish do have a number of limitations. Even though it is possible to save *result* to a CSV file, it is not possible to save *statistics* in the same manner. Neither can multiple queries be performed at the same time. To derive the *statistics* or *result* of a specific author as a CSV string it is necessary to use the "*Copy >*" button. This is shown in Fig. 3.12.

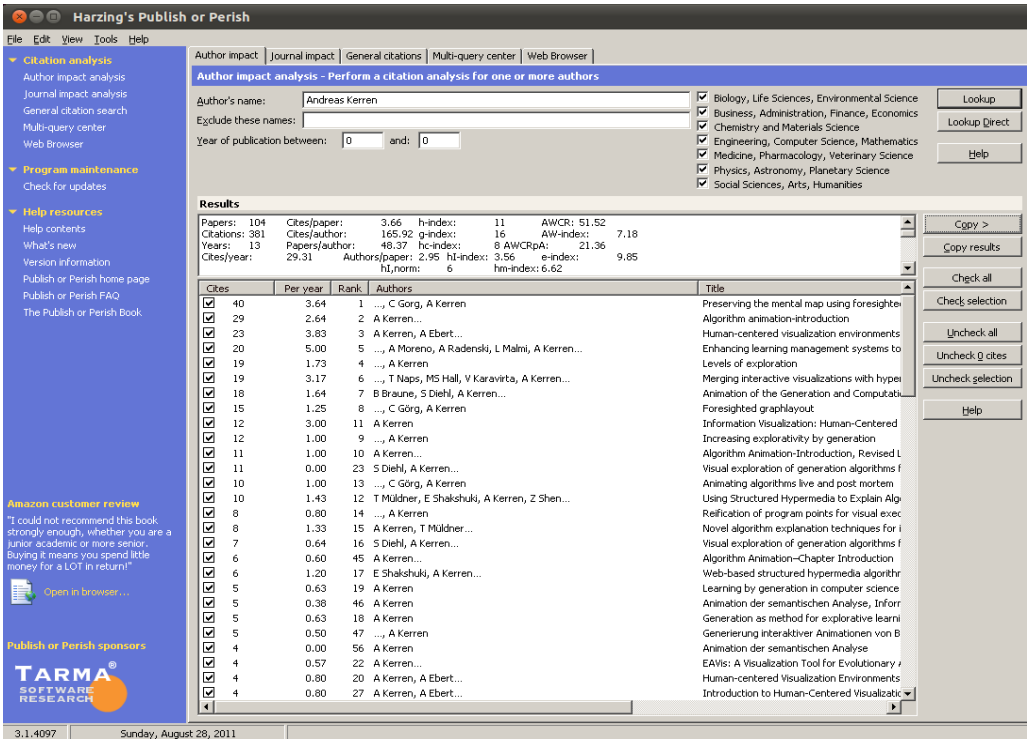


Figure 3.11 : Publish or Perish

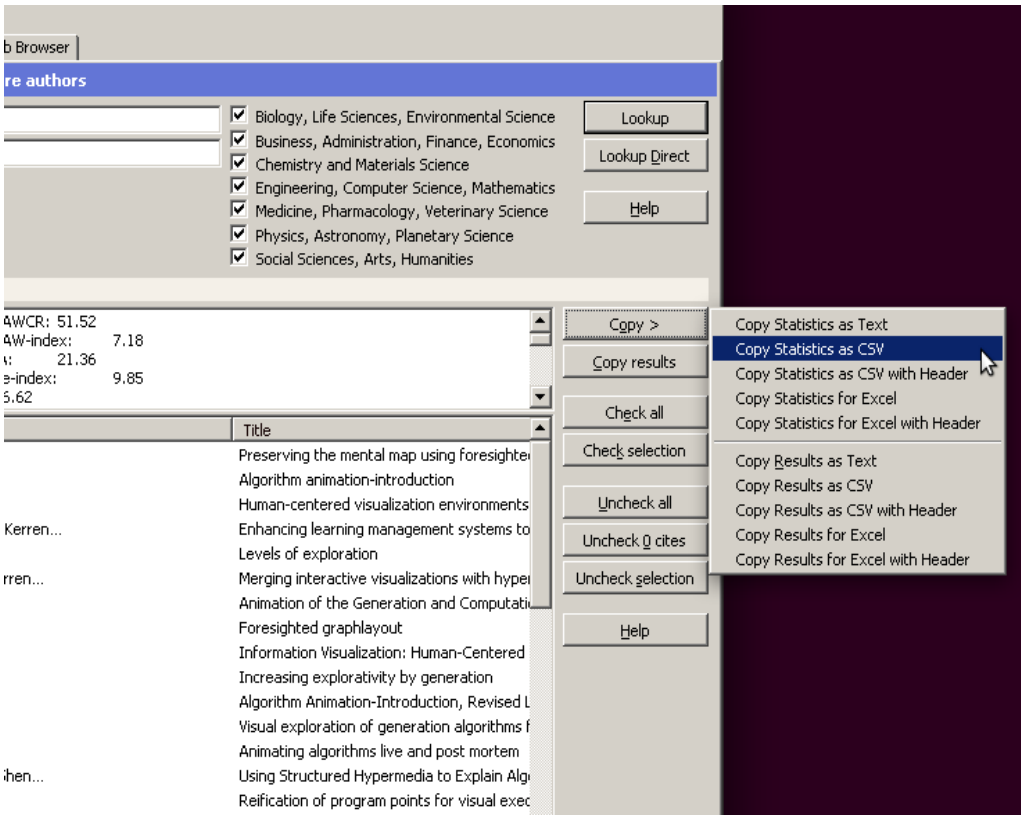


Figure 3.12 : Publish or Perish "export"

Protovis: The graphics engine

The library used for creating the graphics of the visualizations was Protovis. As noted in Section 2.2 it allows for sophisticated manipulations of basic shapes such as lines and rectangles to create a wide variety of possible visualizations. Protovis takes care of most of the work in the background, allowing for such things as automatic iteration trough arrays, easy to implement interaction, and a minimization of necessary code. When reading Protovis code, an important concept to know is function chaining. Function chaining is a design technique where any method that is called on a object returns the same object it was called on, or some closely linked object such as a child just added. For example, if there is an object: `car`, and a few methods, `openDoor()`, `getIn()`, `closeDoor()`, `startCar()`, with function chaining, instead of writing:

```
car.openDoor();
car.getIn();
car.closeDoor();
car.startCar();
```

It would be possible to write:

```
car.openDoor().getIn().closeDoor().startCar();
```

Each method returns the object `car`. This eases up on the programming, but does make it a little harder to read the code. It is important to look at the last method call to see what type of object is returned. Here is an example of the concept using Protovis code.

```
line.add(pv.Label)
  .text(”Hello World!”)
  .visible(function(d)this.index > dims.length-2 && this.parent.marked())
  .font("12px sans-serif");
```

What is important here is that while the original call is made on a line mark, the `add(pv.Label)` method adds a label to line, then returns that label. This means all the methods called afterwards return that label. Also note the location of the semi-colon. As there is only one, everything between `line.add(pv.Label)` and the end of the last line is one line. It could just as well be written with no line breaks. The reason it is not is a writing convention. Spelling out what each method does incrementally makes for an easier read. This is what lies at the heart of the earlier mentioned declarative style of programming [7].

Lastly, the example above shows another concept important to Protovis and intrinsic to JavaScript. While the `.text` property is set using a simple to understand string, the `.visible` property is set using a function. This is because in JavaScript functions can be treated as objects. They can thus be passed as arguments, stored in variables, and replaced. As an example, the text property might instead read:

```
  .text(function(a,b) b.Author)
```

In this function, the text would be set to the Author property of b, b being the parent of the label object `.text` was called on. Proper use of this technique allows for sophisticated manipulation of how the marks and the related data is displayed.

Miscellaneous JavaScript Technologies

A number of basic JavaScript technologies are used in Poppry. As have been mentioned AJAX is used for the integrated input, and JSON is used to represent the data. The file input uses a relatively new FileReader API which was developed in relation to the newest HTML 5 release. The FileReader provides easy to implement file reading capabilities which are entirely client-side. With it we can refrain from using a file uploader plug-in which would necessitate reading the files to the server, and include a number of other technologies and languages such as Flash and php. The drawback is that older browser might have trouble supporting it. In addition a few third party methods were used to conduct utility operations. An example would be the `parseCSV(csvString)` function in the *parser.js* file, which uses regular expressions to parse a CSV string which includes quoted tokens which themselves represents CSV strings. Where third party methods have been used this have been clearly marked in the source code with a link and a date referencing to where they were obtained.

3.2.2 Architecture and Classes

The application initially consisted of four parts. Later, the visualization.js class was combined with the index.html class, due to compability issues with Internet explorer. Each class defines an area of repsonisbility. The input accepts data into the site, either using copy and paste input, file input, or ajax based input, the last of which reads its data from a default set of textfiles stored on the server. The parser accepts these inputs and turns them from CSV strings into JSON object arrays. The visualization class when it was still implemented accepted these arrays and used them to render visualizations which it returned to index.html. Lastly index html takes care of the interaction. It is responsible for keeping track of how to display the information and how to manipulate it. The application is summed up in the following diagram which shows the classes and how they interact:

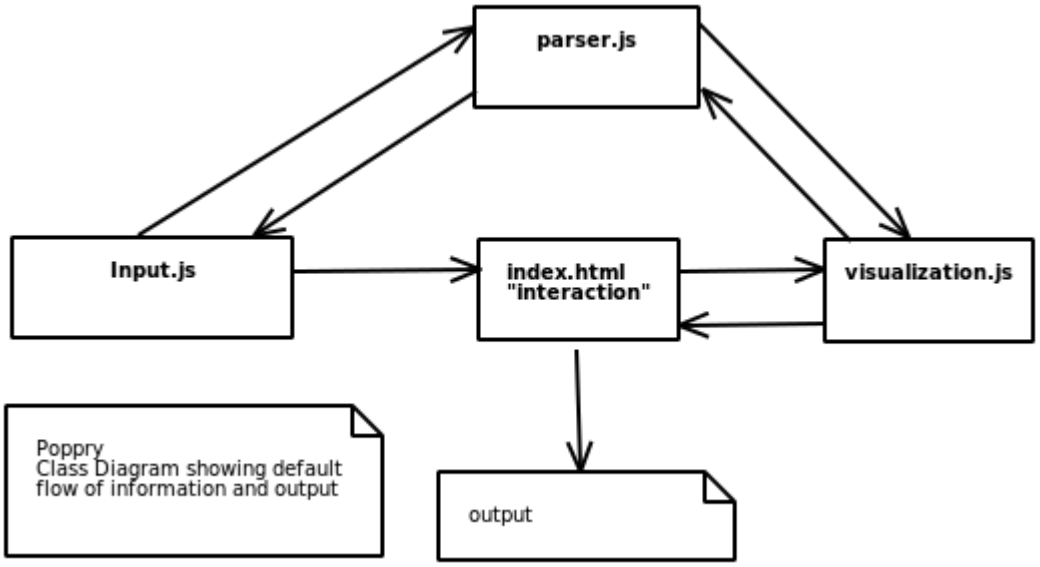


Figure 3.13 : Basic Architecture of Poppry

The classes each define a category of responsibility.

index.html - This is the main file of the site. It sets up the page, displays its GUI. This file is the core file, which all others connect to. It has the responsibility of interaction.

input.js -This file provides methods for all forms of input. All visualizations are initialized with data derived from this class. It has the responsibility of input.

parser.js - This class has the responsibility of formatting the CSV values into JSON objects for the visualizations. It has the responsibility of parsing.

visualizations.js -This class contained one method for each of the visualizations, creating and returning them. It originally had the responsibility of output, before this was overtaken by index.html.

The application also uses the Protovis script and a css file. They are contained in the following files:

protovis-d3.2.js – This file make up the protovis library used to render the visualizations.

poppry_layout.css – This file styles the table presentation.

manual.html – This file defines a user manual for the site.

As *index.html* is the class that is responsible for interaction and forms the core of the application it imports the other classes into it. In fact calling these files classes are a bit of a misnomer. Due to the loose nature of JavaScript, as all script files are imported in the *index.html* file, they can be seen one singular class. The reason they are divided is to clarify the flow of information and make it easier to find a specific method and also to make it easer to maintain and develop.

The input consists of loading the *statistics* and *result* datasets of Publish or Perish into the application. This is handled by *input.js* who provides default datasets used when developing, testing, and demonstrating visualizations, as well as methods for file and GUI input, and a method to clear the data. Because of this it references some HTML elements which exists in *index.html*, specifically those dealing with handling input.

Parser.js has responsibility for parsing the input. This consists of turning the *statistics* and *result* strings into JSON objects so that they can be used by *visualizations.js*. The responsibility for output lies with *visualizations.js*, which accepts JSON data and returns the visualizations. Lastly *index.html* takes care of interaction and how the different classes communicate with each other. In the following sections we will go trough each class, describing its data structures, methods, operations, responsibilities and connections to the other classes.

The Class *input.js*

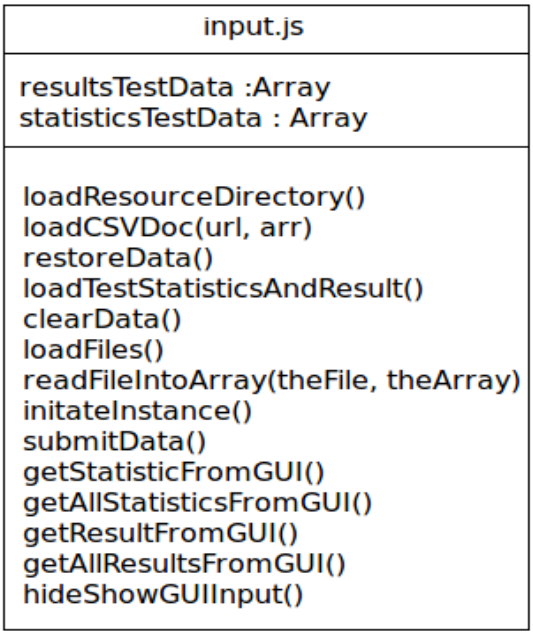


Figure 3.14 : The input class

input.js is responsible for the input of the application. It fulfills this by providing a variety of methods for different kinds of input. The types of input supported are the test datasets, the file input, and the GUI input. Each type of input have a number of methods related to it. The test datasets are loaded into the application using AJAX and were used when developing the visualizations. This basic dataset consists of the *statistics* and *result* for prof. Andreas Kerren, as well as the *statistics* and *results* of around thirty other authors who at one point or another where his coauthors in one of his articles. The second method uses the earlier mentioned FileReader to read directly from the files. A folder is included in the project containing files for testing the FileReader, for use during development. The third method simply reads the strings from HTML text area elements.

In all cases the loaded strings are first parsed, then added to two Arrays named `statisticsJSON` and `resultJSON`. These arrays are located in *index.html*, but can still be accessed by i*nput.js*. These arrays are then either used directly by the visualizations, or further formatted to derive some form of indirect data before given to the visualizations.

Due to the nature of AJAX there is a need to wait until all data have been read before trying to use it. Otherwise we might attempt to use data that is currently being read into the application. Because of this the file and test input both first uses an initialization method which reads the data, then a load method which replaces the current dataset with the newly read data.

The Class *index.html*

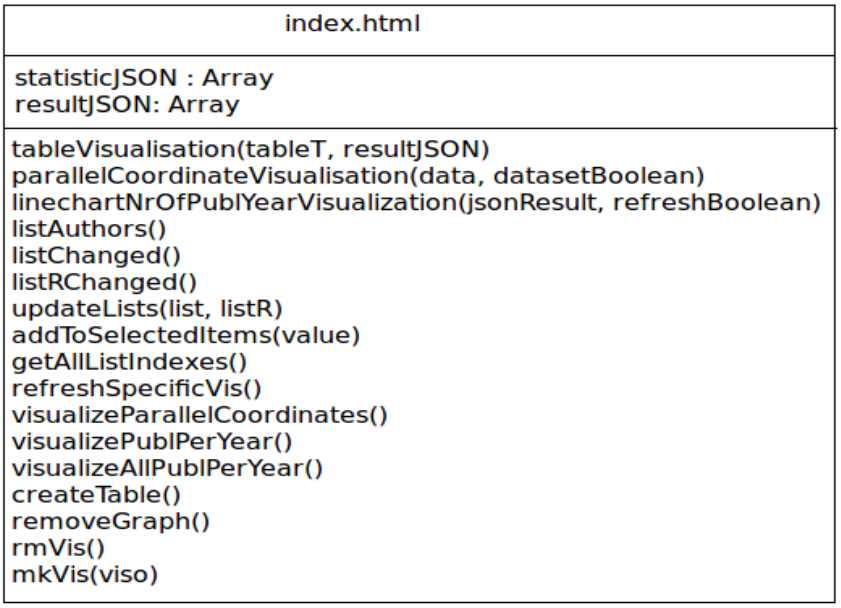


Figure 3.15 : The index class

index.html have the responsibility of interaction. It displays the application, including the visualizations. It also feeds the necessary data to the visualizations. Lastly it connects the entire application, forming the core class. As such it is a basic HTML document which in conjunction with the CSS layout poppry_layout.css defines the GUI.

The *index.html* file defines two JSON arrays which stores the parsed datasets. All relevant data is loaded into these arrays and later used by the visualizations. The GUI is constructed using basic HTML elements, and is designed to put the visualizations in the center. It defines a menu for the visualizations, menu for the file input, a menu for the GUI input, and a menu to manipulate the data. It fulfills its role as the core file of the application by importing all external scripts into itself. No other imports are necessary as this allows any script to access any other.

The Class *parser.js*

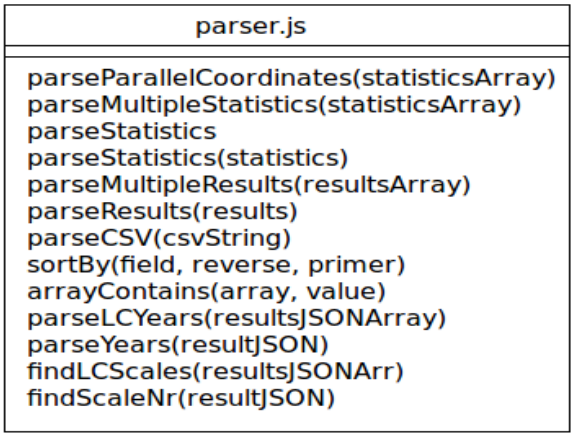


Figure 3.16 : The parser class

The parser has the responsibility to format the data. It provides no data structures in itself, relying on being given the necessary strings of data. It accepts arrays of *statistics* and *result* strings, and turn them into arrays of JSON objects.

The methods in *parser.js* are divided into three categories. The first is a set of visualization specific methods. These methods parse their input for one specific visualization. These are used when the visualization requires data which is not directly present but must be calculated from the basic datasets. The second set of methods consists of more general parser methods, which simply turns the given strings into arrays of JSON objects. The last set is a few utility methods, doing things such as parsing a CSV string with commas inside strings that should not be used to split the document up, sorting an array of JSON objects using some object property, or finding the minimum value in an array of JSON objects.

In the beginning, the *parser.js* external script was used by *index.html* to format the string based Publish or Perish data. In addition some parsing still took place in the *visualizations.js* external script. This was later changed so that the parser only connected to the visualizations, and so that all formatting took place in *parser.js*. This defined the responsibilities of each class better, as well as made the flow of data easier to follow. Later this was further restructured in that *parser.js* also connects to *input.js* to allow for a basic JSON dataset to be stored in the application rather than calculated each time a visualization needed to be redrawn. This was done to allow for easier manipulation of already given data. In addition, extra care was taken in the comments in the *parser.js* methods. This was done due to Javascript being a loosely typed language. It does not have strict types to its data. This means finding out what data structure a method accepts and what data type it returns is not automatic. Having a clearly outlined commentary for *parser.js* with its vastly different return formats greatly eases up the clarity of the code.

Here follows an example to demonstrate the basic functionality of this class:

```
	/**
	 * Accepts a string representing the results of one(1) author.
	 * Returns an array of JSON object, each JSON object representing one of his
	 * publications.
	 * @param {Object} results
	 */
function parseResults(results){
	var data = parseCSV(results);

	var dataArr = new Array();
	for(var x = 0; x < data.length; x++){
		var jsonData = {
			"Cites":parseFloat(data [x] [0]),
			"Authors":data [x] [1],
			"Title":data [x] [2],
			"Year":parseFloat(data [x] [3]),
			"Source":data [x] [4],
			"Publisher":data [x] [5],
			"ArticleURL":data [x] [6],
			"CitesURL":data [x] [7]
		};
		dataArr [x] = jsonData;
	}
	return dataArr;
}
```

This method should give an idea of the parser code design. A standardized comment, noting what information the method takes, and what it returns, The code itself starts with the string being tokenized and used to create an array of JSON objects. The `parseCSV(dataArray)` method is a third party method utilizing regular expressions to parse a CSV string which contains other strings. To clarify, given the CSV example string of '15, 1994, “A Kerren”, “A Ullrich, Xia Lin, Beatrix Braune”' `parseCSV(dataArray)` would output the last attribute as one single token: “A Ullrich, Xia Lin, Beatrix Braune“, rather than as the three tokens of '“A Ullrich' then 'Xia Lin' and finally 'Beatrix Braune”'.
This is a more complicated method, showing one of the more complex formats of the data.

```
/**
 * Accepts an array of JSON objects, each string represening one(1) authors
 * results.
 * Returns an array of JSON arrays, each JSON array representing that
 * authors nr of publications per year.
 * [[nrOfPublJobj, nrOfPublJobj], [nrOfPublJobj,nrOfPublJobj]]
 * @param {Object} resultArray
 */
function parseLineChartNrOfPublPerYearsResults(resultArray){

	//var dataArr = parseMultipleResults(resultArray);
	var dataArr = resultArray;
	var totalArray = new Array();
	var tempInnerArray = new Array();
	var innerArray;
	var sum = 0;
	var currentYear;

	for(var i = 0; i<dataArr.length; i++){


		if(dataArr [i].length != 0){
			innerArray = new Array();
			tempInnerArray = dataArr [i];
			tempInnerArray.sort(sortBy('Year', false));

			currentYear = tempInnerArray [0].Year;
			for(var x = 0; x < tempInnerArray.length ; x++){
				if (currentYear == tempInnerArray [x].Year) {
					sum++;
				}
				else {
					innerArray.push({
						'Year': currentYear,
						'Nr': sum
					});
					sum = 0;
					currentYear = tempInnerArray [x].Year;
				}
			}
			totalArray.push(innerArray);
		}
	}
	return totalArray;
}
```

Once again the comment is standardized. In addition, the name is standardized:

parse<visualizationame><datasetype> (<datasetype><Datatype>).

This is done to ease the clarity of the code, which is extra important for the parser.js class due to the often confusing JSON arrays it handles.

The Class *visualizations.js*

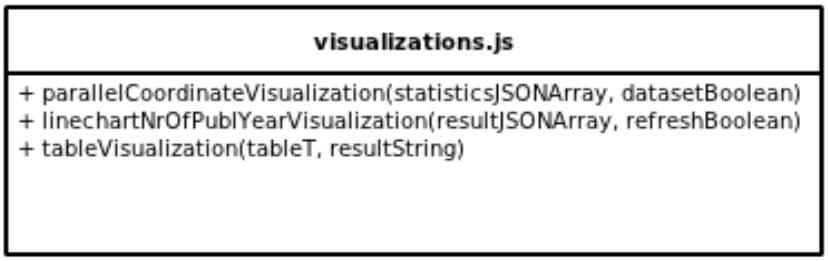


Figure 3.17 : The visualization class

During development, this class was moved in its entirety into the index.html file. This was in order to make the site compatible with Internet Explorer. In order for Internet explorer to read the Protovis code correctly, it needs to use the internal parser belonging to Protovis. The only way to define this is in the script tag in index.html. Since .js files cannot contain script tags, any method that uses Protovis code must be places within the html file in which it is used. In all other ways the methods are still used in the same fashion. As such, the visualization class remains here as a conceptual idea to give the reader a clearer view of the applications design.

The visualization class had the responsibility of creating and returning the visualizations. It provided one method for each visualization in the application. Each method is given the information it needs as an array of JSON objects. These objects are used to create and draw the visualization said object represents, then returns it to the method caller as a Protovis based object. The parallel coordinate visualization uses an array of JSON objects, each JSON object representing one specific authors *statistics* data. The line chart visualization uses an array of arrays, each secondary array representing an authors *results*, filled with a list of JSON objects representing that authors articles. The table presentation uses one array of JSON objects, that array representing one authors *result*, each entry representing one article. The visualizations themselves consists of setting any necessary scales including width and height, then defining the root panel and adding a variety of marks, declaring their properties which defines how the visualization is rendered, defining any form of interaction, and finally returning the root panel to *index.html.*

The methods are used by i*ndex.html* who accepts the returned visualization, removes whatever visualization is currently displayed, and adds the returned visualization to the GUI.

Example – Line chart visualization.

This is the line chart visualization, and it will be used to demonstrate some of Protovis capabilities. Pay attention to the relative small amount of code. A lot of the work is automatically done by Protovis.

We start with defining the scales. w and h is dependent on the width and height of the browser window. These stand for the width and the height of the visualization. x and y defines the scales for the axes and is necessary to properly map the given data to the axes. They are also used to set up the ticks which are the small lines running along the axes which define the axes values.

```
var w = window.innerWidth*0.45,
    h = window.innerHeight*0.5,
    x = pv.Scale.linear(findMinYear(nrYear),  new
      Date().getFullYear()).range(0, w),
    y = pv.Scale.linear(0, maxNr).range(0, h);

// The root panel.
var vis = new pv.Panel()
    .width(w)
    .height(h)
    .bottom(20)
    .left(20)
      .right(10)
      .top(5);
```

The two following code segments adds the straight lines, sets up the axes, the light gray grid pattern, and the ticks along the axes, as well as the text along those ticks.

```
// X-axis ticks.
vis.add(pv.Rule)
    .data(x.ticks())
.visible(function(d) d ≥ 0)
.left(x)
.strokeStyle("#eee")
.add(pv.Rule)
 .bottom(-5)
 .height(5)
 .strokeStyle("#000")
.anchor("bottom").add(pv.Label)
.text(x.tickFormat);

// Y-axis ticks.
vis.add(pv.Rule)
    .data(y.ticks(5))
    .bottom(y)
    .strokeStyle(function(d) d ? "#eee" : "#000")
  .anchor("left").add(pv.Label)
    .text(y.tickFormat);
```

The following code segment adds a line for each author in the given data array of `nrYear`. The reason we add a panel to the root panel `vis` is because we might have more than one author in the `nrYear` array. Adding another panel allows us to cycle trough the two dimensional array, adding a line for each author. As such we will add a line for each entry in the `nrYear` array, each of those lines will use the current `nrYear` entry for its data, that is the array for one author, to draw a polyline using that authors attributes. The reason we add the second panel has to do with how we handle the interaction, namely by defining a new property, "`marked`" and using this property as a flag to note how we should color the lines. The coloration is done in `.strokeStyle`. In addition the `.strokeStyle` and `.event` properties are both given functions to allow for further to interaction.

```
    vis.add(pv.Panel)
          .data(nrYear)
          .left(function() this.index * 10)
    .add(pv.Panel)
          .def("marked", false)
    .add(pv.Line)
          .data(function(array) array)
          .interpolate("linear")
          .left(function(d)x(d.Year))
          .bottom(function(d)y(d.Nr))
          .lineWidth(3)
          .strokeStyle(function() (arrayContains(selectedIndexes,
                      this.parent.parent.index) || this.parent.marked() || !
                      refreshBoolean) ? "orange" : "steelblue")
          .event("click", function(d)this.parent.marked(lineClicked(
                      this.parent.parent.index, this.parent.marked())));
```

This code segment consists of the function used by `.event` whenever a line in the line-chart is clicked. This function contains calls to functions in *index.html*, telling the page what parts it should update in response to the click. Lastly, he visualization is returned.

```
    function lineClicked(value, marked){

          if(refreshBoolean){
                addToSelectedItems(value);
                selectedIndexes = getAllListIndexes();
                visualizePublPerYear();
                return (!marked);
          }
    }

return vis;
```

4 Conclusion

At the start of this document the problem was defined, which was to create a visualization tool to enhance the data of Publish or Perish. In addition a number of criteria and restrictions was defined and divided into core, extended, recommended and obligatory. After defining the problem we discussed related work and gave a small introduction to necessary background knowledge. Then we described each part of the application and went trough the implementation details, outlining the architecture and code. We will now show what criteria has been fulfilled and what has not.

The initial core requirements were as follows:

- *Parse the data given from Publish or Perish into a JSON format.*
- *Visualize the bibliometric data given from Publish or Perish.*
- *Compare two authors bibliometric data.*
- *Provide an overview and references to authors publication history, as derived from Publish or Perish.*
- *Reference the abstracts and papers for an author given from Publish or Perish*
- *Ability to filter and highlight specific parts of the data.*

In the introduction, the concept of bibliometric data was identified and it was shown to be a part of the Publish or Perish *statistics* dataset. The authors publication history was identified as Publish or Perish *result* dataset. In Subsection 3.1.1 trough to Subsection 3.1.4 the core requirements were shown to be fulfilled. The Parallel Coordinate visualization fulfills visualizing bibliometric data, and allows two or more authors to be compared. It also allows for filtering data thanks to its sliders. In addition the combined Line chart and Table visualizations provide an overview of an authors publication history, and gives a reference to the abstracts and papers of that author.

The extended requirements were as follows:

- *Extend visualization to accommodate more views.*
- *Compare more than two authors*
- *Integrate references to papers and abstracts into the web based tool itself.*
- *Add in a function to export visualizations.*
- *Compare two or more authors publication histories.*

In addition there was a few amended requirements, both to the core and to the extended, functioning as more specific versions of the initial requirements. These were as follows:

Core

- *Create a parallel coordinate visualization to visualize the statistics dataset.*
- *Connect the visualizations to allow a change in one to be displayed in the other.*
- *Allow for file based input.*
-

Extended

- *Create a line chart visualization to visualize an authors publications per year, derived from results.*
- *Create a bar chart visualization to visualize what authors an author has cooperated with on what articles.*

The comparison of more than two authors was fulfilled by the parallel coordinate visualization. The comparison of two or more authors publication histories were partly fulfilled using the line chart visualization which allows for a comparison of the number of articles they have published each year. The line chart visualization do have three major disadvantages in that there is no easy way to find out what line belongs to what author, there is no easy way to compare the publication history of two authors who where active in different time periods and there is no real way of finding out the number of publications an author has had in unknown years.

Neither the integration of papers and abstract, nor the exportation of visualizations were implemented in the end. The first due to technical difficulties and the second due to time limitations. Of the amended requirements from Subsection 3.1 the parallel coordinate visualization was fully implemented, and the line chart visualization was implemented. The bar chart was not implemented due to time constraints. Both the file based input and the connected visualizations were implemented.

Comparable the number of restrictions where small. The application should be accessible trough the Internet, and it was recommended to utilize Publish or Perish for the dataset, Protovis for the graphic library, and by extension Javascript as the main language. All of these restrictions have been followed.

As for the amended requirements a parallel coordinate visualization was added which functions well, and a line chart visualization was added which could be improved by dealing with any of its three disadvantages. To summarize, all core requirements were fulfilled, a majority of the extended requirements where fulfilled, and all of the restrictions where kept to.

4.1 Disadvantages and Advantages of Poppry

We have shown that the application fulfills the core requirements laid out at the start of the document. In this section we will discuss what advantages and disadvantages the application has which was not included in the initial requirements, as well as neutral qualities that are neither negative nor positive. Starting with the disadvantages the response time when selecting a different author either in the list or in a visualization itself is non-instantaneous. During tests it usually lies below one second. Certain operations such as loading data generally takes around a second. The response time is fast enough not to be frustrating, but it is still worth noting. A second issue with the visualization is that due to the nature of how files are read, on occasion they are read out of order and as such *statistics* and *results* are incorrectly connected. This error occurs seldom enough for the application to still be functional. Due to the inclusion of the FileReader the site also requires browsers compatible with the FileReader element.

The last disadvantage is that the code itself could be more optimal as it was constructed with little experience in Javascript and as such many of the classes were a process in learning as much as in designing. As for the advantages the entire site save for the loading of the test data set executes client-side, the visualizations are connected to each other allowing for easy selection and filtering while looking for specific authors and both file and GUI input is possible. Of the other qualities the intended size of the dataset to be used with the site ranges from circa 1 to 40 authors though technically there are no upper limit, save for when the visualizations become incomprehensible. In addition due to the object oriented approach to the site the application should be relatively easy to understand and improve should it be used as a base for a future thesis.

4.2 Future Work

There are a number of areas where the application can be improved. Not only based on the requirements left out, but also on natural enhancements, the improvements can be divided into input, specifically how the data source is accessed and how the data is read, core improvement, adding more visualizations or improving those that are already there, persistence, allowing for a slightly easier manipulation of data, conversion, changing the libraries used to be up to date, or code improvement, focusing on making the application less prone to error.

Input

The input is the area where the application requires the most improvement. Currently the only real ways to add in new information is trough pasting the CSV derived values of Publish or Perish into the GUI, or uploading files. The first is a fairly uncomfortable operation for the user, especially if wanting to input more than one author, and the second requires extra care to input everything in order. There is also no form validation of the input, making it prone to error and a potential security risk. As such it would be interesting to somehow integrate obtaining the input from Publish or Perish into the application itself, or otherwise improve on the file input, easing up on the user and ensuring we can always get the correct data. An additional function this might lead to is allowing a greater ease manipulating the data while it is being queried. For example if a user does a search for an author within the application and is given the *result* of that author in a table presentation, he might for each article be given a field which allows him to do searches on that authors coauthors. A further addition could be

improving the FileReader to become more responsive and show more information such as a progress bar, or allow the user to keep more than one upload in the site. In addition improvements might be made in the parser, allowing for the application to correct for minor errors in the given data.

Alternatively form validation of the GUI based input might be added for much the same effect in addition to increasing application security. Security might also become more important if the persistence suggestion is implemented.

Core Improvement

Another area of improvement is in the applications complexity. Additional visualizations would be an useful addition to the application. A suggestion for one such visualization is a combination with the visual analysis of co-authorship thesis, integrating it into the application or the application into it. This might be combined with the conversion improvement discussed further down. It might also include a visualization of the *"authors"* attribute in *result*. Another improvement would be the visualizations measuring other things than author impact such as journal impact or improvements of the visualizations already present. Lastly, the manipulation of information already loaded into the application could be improved. For example it could be possibility to delete specific authors or separate them into a different list. Currently, there are very few real ways to manipulate already input data. These suggestions could be combined with the persistence suggestion.

Persistence

Another improvement of the application could be adding a database to it, allowing users to set up and maintain sets of authors which interest them. This would save effort in the need to constantly re-input authors which are used often. Another form of persistence would be in allowing the export of a visualization as an image, essentially allowing it to be saved for future use such as being included in some form of paper. This was one of the extended requirements not fulfilled due to time constraints.

Conversion

Sadly, Protovis ended active development June 28 2011. The last version was 3.3.1. The decision to continue using Protovis was made in part not to throw away previous work, and in part because by its last version it was already fairly well developed, and also independent requiring no third party dependencies save for the core library. However, a conversion to its successor toolkit D3.js might be of interest. It is developed by the same team who created Protovis and takes an interesting new approach to graphic visualization online.

Code improvement

The code could use more debugging as well as other improvements such as removing superficial methods, improving comments, optimizing certain parts to increase response times, or just make the system more robust in general, ensure better cross browser compatibility, comprehensive testing and so on. This could easy be combined with the Input and Core Improvement suggestions.

References

1. Information Visualization: Design for Interaction - Robert Spence (2nd ed 2007 Pearson Education Unlimited)
2. Information Visualization: Perception for Design - Colin Ware (2nd ed 2004 Elsevier Inc)
3. Bibliometrics and Citation Analysis – Nicola De Bellis (2009 The Scarecrow Press, Inc)
4. http://www.harzing.com/pop.htm 2011-03-06 The Publish or Perish home page.
5. http://dblp.uni-trier.de/ 2011-03-06 The Digital Bibliography Library Project
6. http://scholar.google.se/ 2011-03-06 The Google Scholar search engine
7. http://vis.stanford.edu/protovis/ 2011-03-06 The ProtoVis Javascript based visualization libraries.
8. http://www.isiwebofknowledge.com/ 2011-03-06 The Web of Knowledge Homepage
9. http://www.w3.org/TR/FileAPI/ 2011-08-28 The FileReader API
10. http://www.math.tau.ac.il/~aiisreal/ 2011-08-28 Al Inselberg homepage
11. Interactive Data Visualization - Ward, Grinstein, Keim (2010 A K Peters, Ltd)
12. Introduction till Bibliometri – Ritta Kärki & Terttu Kortelainen (1998 NORDINFO)
13. http://www.harzing.com/pophelp/metrics.htm 2011-08-12 Publish or Perish homepage.
14. http://mbostock.github.com/protovis/ex/ 2011-09-04 Protovis homepage.
15. http://mbostock.github.com/protovis/ex/line.html 2011-09-04 Protovis homepage.
16. http://www.visualcomplexity.com/vc/blog/?p=277 2011-08-09 Visualization Blog.
17. http://www.w3schools.com/ajax/default.asp 2011-09-04 W3 Schools on Ajax.
18. Course on Information Visualization – Andreas Kerren (Linné University 2010 Växjö Sweden)
19. Human-Centered Visualization Environments. A. Kerren, A. Ebert, and J. Meyer (Eds.). Volume 4417 of LNCS Tutorial, Springer, 2007.

Appendix

A. The original Problem Proposal.

Bachelors Thesis

Visual Analysis of Author Impacts and Bibliometric Data

Author: Björn Kostkevicius

Supervisor / Reference person(s) : Andreas Kerren, Ilir Jusufi

Index

1. Problem Description including Background, Aims and Restrictions

1.1 Background

There are a lot of academic publications every year, and it is sometimes difficult for authors to get noticed, just as it is sometimes difficult finding specific academic literature you are looking for. There are attempts to make this easier, with search engines made for looking up books and journals, for example Google Scholar, or applications built for this purpose such as Publish or Perish. Publish or Perish is an application that functions as a search engine and organizer of academic publications and bibliometric data. It has a variety of search functions, and then presents basic lists of the results. It does not provide any visualizations of the results however, which can make it difficult to get a proper overview of the data as well as compare it with others. The stated goal of Publish or Perish is to help individual academics to present the impact of their work. As such a visualization of its data could be of interest. The bibliometric data gathered by Publish or Perish takes many forms, such as the following:

- Total number of papers.
- Total number of citations
- Average number of citations per paper
- Average number of citations per author
- Average number of papers per author
- Average number of citations per year
- Hirsch's h-index and related parameters
- Egghe's g-index
- The contemporary h-index
- The age-weighted citation rate
- Two variations of individual h-indices
- An analysis of the number of authors per paper.

(From Publish or Perish homepage, see 4. References)

Of particular notes are the indices who summarice many different ways of looking at citations. Things such as Hirsch's h-index, which *"strives to quantify an individuals scientific research output"*, or Egghe's g-index, who tries to improve the h-index by giving more weight to highly cited articles.

1.2 Aim

The aim for this thesis is to develop a web based interactive visualization tool.

It should visualize the bibliometric data of academic authors and provide ways to explore a set of authors. There should be ways to compare or highlight specific authors, to highlight specific data, as well as show paper abstracts or articles. A later combination with the co-authorship visualization thesis should be possible.

The basic functionality is as follows

Core functionality:

Parse the data given from Publish or Perish into a JSON format.

Visualize the bibliometric data given from Publish or Perish.

Compare two authors bibliometric data.

Provide an overview and references to authors publication history, as derived from Publish or Perish.

Reference the abstracts and papers for an author given from Publish or Perish

Ability to filter and highlight specific parts of the data.

Extended functionality

Extend visualization to accommodate more views.

Compare more than two authors

Integrate references to papers and abstracts into the web based tool itself.

Add in a function to export visualizations.

Compare two or more authors publication histories.

Additionally, the application should be combined with a report explaining its construction, usage, motivation, the theories it is based on, and an evaluation of it, including possible extensions for other thesis projects.

1.3 Restriction

The restrictions are that it must be web based, possible combined with the co-authorship visualization at a later date, and based largely on JavaScript.

2. Main idea for problem solution.

Create a web based visualization tool which allows searching for authors, titles and subjects, and displays this information in an easily understandable, comprehensive way. The problem is divided into the following parts.

Obtain data – This can be done in any way possibly. It would be preferable if it could be integrated into the finished application, but a half-automatic or even manual way of obtaining it is acceptable for this thesis.

Format data – The data should be formatted to JSON, so as to allow the use for visualization libraries, as well as a possible key in with the co-authorship visualization project.

Visualize data – There are many specific visualization library out there such as ProtoVis. Specific ones should be selected and used in the application. Their use should be explained and motivated, and the overall design of the application should be justified.

3. Time Schedule

Week - Tasks

16 Design and implement visualisations.
17 Design and implement visualizations. Design gui. Work on report.
18 Evaluate design of visualizations and gui. Implement gui. Work on report.
19 Improve parser, visualizations, GUI. Work on report.
20 Improve parser, visualizations, GUI. Evaluate. Work on report.
21 Testing or further improvements.
22 Testing or further improvements.

4. References

- Information Visualization: Design for Interaction - Robert Spence (2nd ed 2007)
- Information Visualization: Perception for Design - Colin Ware (2nd ed 2004)
- Bibliometrics and Citation Analysis – Nicola De Bellis (2009)
- Interactive Web-based Visualization Tool to Support Inquiry-based Science Learning - Emil Johansson (2010-06-18)
- http://www.harzing.com/pop.htm 2011-03-06 The Publish or Perish home page.
- http://dblp.uni-trier.de/ 2011-03-06 The Digital Bibliography Library Project
- ttp://scholar.google.se/ 2011-03-06 The Google Scholar search engine
- http://vis.stanford.edu/protovis/ 2011-03-06 The ProtoVis Javascript based visualization libraries.
- *http://www.isiwebofknowledge.com/ 2011-03-06 The Web of Knowledge Homepage*