

University of Manchester
School of Computer Science
Project Report 2007

Google Scholar Analyser

Author: Md Mooktakim Ahmed

Supervisor: Professor Norman W. Paton

Abstract

Google Scholar Analyser

Author: Md Mooktakim Ahmed

Google Scholar provides a simple way to search for academic literature. The most prominent way of publishing research is through research papers. Such papers commonly reference other papers on related work. One could use the links between the research papers to determine the popularity of the paper, or how relevant it is to a subject area.

Stand on the shoulders of giants ¹

The main objective of this project is to take the results from Google Scholar and display it in alternate ways so that it can be used for bibliography analysis. This analysis should give the user a better understanding of the importance of a research paper, or its author, in a particular subject area.

Supervisor: Professor Norman W. Paton

¹Google Scholar has adopted Stand on the shoulders of giants as its motto.

Acknowledgements

I would like to thank my supervisor Professor Norman W. Paton for his help and guidance throughout the duration of the project. I would also like to thank my friends and family for their continuous support.

Contents

1	Introduction	6
1.1	Introduction	6
1.2	Citation Analysis	6
1.3	Existing Solutions	6
1.4	Aims and Objectives	7
1.5	Conclusion	7
1.6	Overview of Subsequent Chapters	7
1.6.1	Chapter 2 - Background Research	7
1.6.2	Chapter 3 - Requirements Analysis	7
1.6.3	Chapter 4 - Design	8
1.6.4	Chapter 5 - Implementation	8
1.6.5	Chapter 6 - Evaluation	8
1.6.6	Chapter 7 - Conclusion	8
2	Background Research	9
2.1	Introduction	9
2.2	Google Scholar	9
2.3	Web Mash-ups	9
2.4	Web Application	10
2.5	Model View Controller Architecture	10
2.6	User-centered Design Principle	10
2.7	Google Scholar API	11
2.8	Screen Scraping	11
2.9	Usability Engineering	12
3	Requirements Analysis	13
3.1	Introduction	13
3.2	Related Proposals	13
3.2.1	Citeseer	13
3.2.2	DBLP	14
3.2.3	The ACM Portal	14
3.2.4	Google Scholar	16
3.2.5	Conclusion	16
3.3	Prototyping	18
3.3.1	Phase 1	18
3.3.2	Phase 2	18
3.3.3	Phase 3	19

3.3.4	Phase 4	19
3.3.5	User Evaluation	25
3.4	Feature Cost Analysis	27
3.5	User Requirements	27
3.5.1	Overall System	27
3.5.2	Results Page	29
3.5.3	Articles Page	29
3.5.4	Authors Page	29
3.6	System Requirements	30
3.6.1	Functional Requirements	30
3.6.2	Non-Functional Requirements	30
3.7	Data Modelling	30
3.8	Conclusion	32
4	Design	33
4.1	Introduction	33
4.2	Software Architecture	33
4.2.1	Model	33
4.2.2	View	34
4.2.3	Controller	34
4.3	Overall Architecture Design	34
4.4	Class Diagram	34
4.5	Conclusion	35
5	Implementation	37
5.1	Introduction	37
5.2	Programming Languages	37
5.2.1	Java	38
5.2.2	PHP	38
5.2.3	Ruby	38
5.2.4	Java Servlets	39
5.3	Web Framework	39
5.3.1	Google Web Toolkit	39
5.3.2	Apache Struts	40
5.3.3	Ruby on Rails	40
5.4	Webserver	40
5.4.1	Apache with mod php	40
5.4.2	Apache Tomcat	41
5.4.3	Mongrel	41
5.5	Chosen Technology	41
5.6	Cache Technology	41
5.7	Screen Scraping Techniques	42
5.8	Further Detail on Ruby On rails	42
5.8.1	HAML Template Engine	44
5.8.2	Regular Expressions	44
5.8.3	ActiveRecord	45
5.8.4	Gruff - Graphs for Ruby	45

5.8.5	Testing Framework	46
5.9	Architecture Implementation	46
5.10	Implementation of Requirements	46
5.11	Conclusion	49
6	Evaluation	50
6.1	Introduction	50
6.2	System Evaluation	50
6.3	User Evaluation	51
6.4	Conclusion	51
7	Conclusion	52
7.1	Introduction	52
7.2	Project Achievements	52
7.3	Further Work	52
7.4	Conclusion	53
	Bibliography	54
A	Mock-up User Evaluation Questionnaire	55
B	Different Phases of Mock-up Design	60

List of Figures

2.1	Model View Controller architecture diagram	11
3.1	Citations per year graph generated by Citeseer for an article	14
3.2	Citeseer - Most cited authors in Computer Science - August 2006	15
3.3	DBLP - Category listings	15
3.4	DBLP - Coauthor Index	16
3.5	The ACM Portal - An article page	17
3.6	Google Scholar - Example search result	17
3.7	Prototyping Phase 1 - Results page	18
3.8	Prototyping Phase 2 - Results page	19
3.9	Prototyping Phase 2 - Citations per year	20
3.10	Prototyping Phase 2 - Cited By	20
3.11	Prototyping Phase 3 - Results page	21
3.12	Prototyping Phase 3 - Statistics pie chart	22
3.13	Prototyping Phase 3 - Citation Tree	23
3.14	Prototyping Phase 4 - Articles page	24
3.15	Prototyping Phase 4 - Front page	25
3.16	Prototyping Phase 4 - Front page with live search	26
3.17	Prototyping Phase 4 - Results page	27
3.18	Prototyping Phase 4 - Articles page	28
3.19	Entity Relationship Diagram	31
4.1	Model View Controller architecture diagram	34
4.2	Overall Architecture Design	35
4.3	Design class diagram	36
5.1	Ruby on Rails architecture diagram	43
5.2	Example graph using Gruff	45
5.3	Class Diagram	47
5.4	Complete Class Diagram	48

Chapter 1

Introduction

1.1 Introduction

Google Scholar provides a simple way to search for academic literature. The most prominent way of publishing research is through research papers. Such papers commonly reference other papers on related work. One could use the links between the research papers to determine the popularity of the paper, or how relevant it is to a subject area. This is called Citation Analysis.

1.2 Citation Analysis

Citation Analysis is the examination of the frequency and pattern of citations in research papers [Rub04]. When a paper references another paper, it conveys to the reader that the paper which it refers to is relevant to the current paper. You may then use the number of times a research paper is referenced as a guide to measure the relevance of the paper. The reader can, by traversing through the citation links, find alternate research papers which may be even more relevant than the current paper. You can even find authors who are regarded as experts in a certain subject area, just by looking at the citations of the papers which they have published. Using this technique a researcher can assimilate papers which are related to their subject area without wasting time on unrelated materials.

1.3 Existing Solutions

Search engines for academic material is not a new concept. There are many services available on the Internet which provide academic search. Some require membership and others provide the service for free. There are search engines which are available for only certain subject areas, such as medical or computer related subjects. However, most of these search engines require the author to submit their work manually. Since there are many available, the author sometimes has to submit to multiple search engines.

Search engines such as CiteSeer¹ are very popular and well known for their citation analysis features. However, their database, just like other search engines, is incomplete

¹<http://citeseer.ist.psu.edu>

or out of date. This is due to the fact that they do not share data with each other, therefore their citation information is inaccurate.

1.4 Aims and Objectives

The aim of this project is to design and implement a web based application which interfaces with Google Scholar and provides added value by augmenting analysis. The main objectives are to:

- Analyse existing solutions and determine features which are not provided by Google Scholar.
- Analyse user requirements.
- Combine data from other sources to produce integrated experience.
- Generate mock-ups of features and then evaluate them.
- Design and build a modular architecture which allows easy integration of new features.
- Design and build an API interface to Google Scholar which can withstand future changes.
- Design and build a user interface which follows web standards and has usability in mind.

1.5 Conclusion

This chapter has introduced citation analysis and how it be an advantage to researchers when looking for information on certain subject areas. Since this project is user oriented, the project will follow an iterative development process. Even though much of the work will go into analysing and creating new features, they will only be implemented after careful evaluation by its users.

1.6 Overview of Subsequent Chapters

1.6.1 Chapter 2 - Background Research

This chapter explains the background knowledge that has been gathered leading up to implementation.

1.6.2 Chapter 3 - Requirements Analysis

This is a very important section because it analyses the project as a whole. Existing solutions are analysed and features which should be incorporated into the project is identified. Here you will find work on requirements analysis and user requirements. Mock-ups and prototypes are created here to allow the users to evaluate its worth. Once

analysis is complete a thorough requirements is generated which is prioritised in the order of their importance.

1.6.3 Chapter 4 - Design

In this chapter the design details of the overall project is displayed. The API which will interface with Google Scholar is finalised. The modularity of the project is discussed and justified. In depth deliberation of the technology used and how they will be used.

1.6.4 Chapter 5 - Implementation

This chapter displays how the project was implemented. Viable technologies are discussed thoroughly. Technologies which are to be used are justified. There is much detail on the development process and the methodology used. Testing and debugging is also discussed here.

1.6.5 Chapter 6 - Evaluation

This chapter analyses the project after its completion. The project is analysed in accordance to its specification.

1.6.6 Chapter 7 - Conclusion

The conclusion aims to be a reflection on the whole project. There is discussion here about the process that was used, the pitfalls, the successes of the project, and any further work that can be carried out.

Chapter 2

Background Research

2.1 Introduction

Google Scholar is a web application, so therefore this project will be also a web application so that it can be used in the same way as Google Scholar. This chapter discusses the technologies that are involved in this project. The type of application being built is identified and described in this chapter.

2.2 Google Scholar

Google is a search engine. At the heart of its technology is the PageRankTM system, which was developed by Larry Page and Sergey Brin [goo07]. PageRankTM looks at the links coming into a web page and determines the importance of each link. Each link coming in acts like a vote, the importance of the link provides a certain weight in the vote. If a page is ranked high then it means lots of other important web pages have hyper links pointing to it. This principle is very similar to how Citation Analysis works. Instead of looking at the hyper links, we look at the citations.

Google Scholar populates its database by crawling through all the academic search engine. This means it has more data than any other academic search engines. By combining all the other search engines into one place they have created a more accurate citation database. This also means that authors do not have to submit their research papers or books to Google Scholar.

Google Scholar has no features which allow analysis of the data. It is very limited when compared to other academic search engines. Even though Google Scholar lacks in features, it is perfect source for this project because it has much more complete database than any other.

2.3 Web Mash-ups

Over the recent years, with the advent of Web 2.0¹ and AJAX², a lot of work has gone into combining web sources together to produce new integrated solutions. This type of solutions

¹Web 2.0 is a term applied to fully fledged computing platform serving web applications to end users.

²Asynchronous JavaScript and XML (AJAX) is a term describing a web development technique for creating interactive web applications.

are seamless to the user. New technologies, such as web services, allow programmable interface to web applications. Developers and web designers have taken advantage of these services to enrich their own websites by integrating data or features, which they could not have provided on their own. Companies like Google and Amazon have enabled easy access to the web services that they provide. In doing so emerged a new way of building rich web applications. Now anyone with an idea can build sophisticated applications. This new way of building applications has also created a new way of thinking of the web as a resource. When a web application combines data from multiples sources to produce a new web application it is called a Web Mash-up [mas07].

2.4 Web Application

Web application is an application that is accessed with a Web browser over a network such as the Internet or an Intranet [web07]. Web applications are popular because it is not dependent on the user having to run certain operating systems. All that is required is a capable Web browser, nowadays it is very hard to find a computer without a decent Web browser. You instantly support thousands of users on the Internet just by creating something for the Web. This is why many applications, such as Web-mail, have become very popular. There is no need to install software and you can access your email from anywhere around the world. Having multiple computers is also becoming very popular. Most people now carry a laptop and have Desktop PC at home. This means that regular applications are very cumbersome because it is installed only in one computer at a time, and synchronisation can be difficult.

2.5 Model View Controller Architecture

Model View Controller (MVC) is an architecture design pattern [mvc07]. The pattern contains three main components. MVC is a very common design pattern when building web applications. Most web frameworks are designed to follow this pattern. This is because web applications are already separated into 3 layers. The front end of a web application is typically HTML. This represents the View. Most web applications are built on top of a database. This represents the Model. The application is normally built using some server-side technology which takes requests from clients web browser and then it generates a response by creating the HTML web pages. This represents the Controller. Figure 2.1 shows a diagram of this architecture.

2.6 User-centered Design Principle

User-centered design is a design philosophy which considers the needs of the user above all else [ucd07]. If an application heavily depends on its users, then this principle needs to be considered. The main difference from other interface design philosophy is that User-centered design tries to bend the interface to the users behaviour, instead of forcing the user to learn how to use the interface. This design principle heavily relies of the analysis of how users use the system. It is best to create prototypes or mock-ups of the interface which the users can then evaluate the strengths and the weaknesses. This should be done

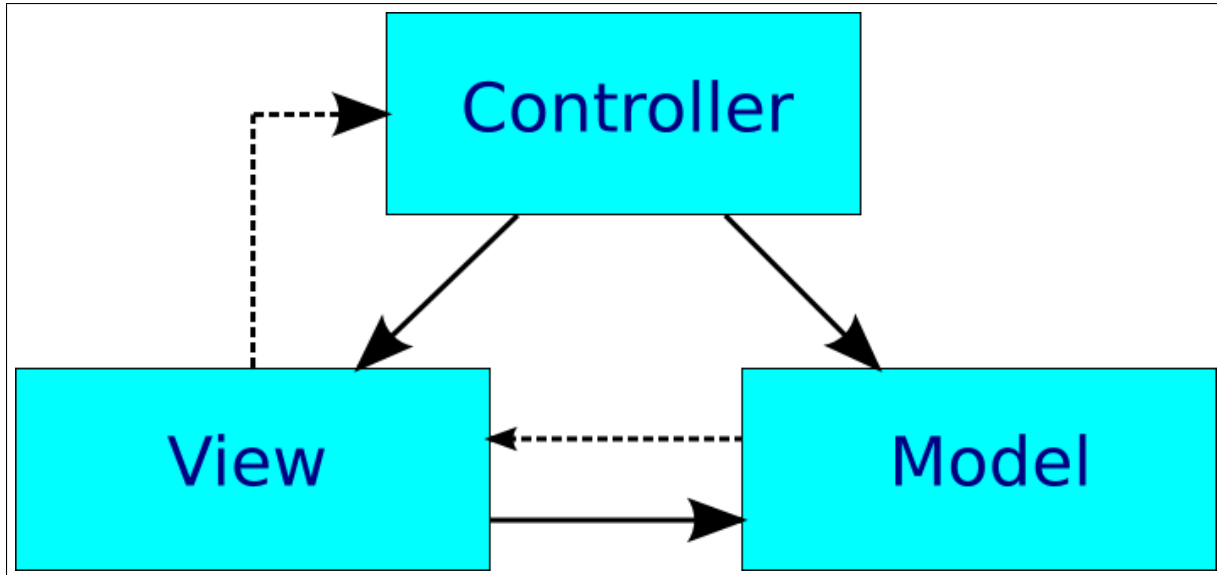


Figure 2.1: Model View Controller architecture diagram

at requirements analysis stage where the users can help decide what features should or should not be built.

2.7 Google Scholar API

Google provides API's³ for most of its web applications. API's allow programmable interface to the web application so that you can easily integrate them into your application. However, Google does not provide an API for its Google Scholar application. An API is used when an application, or a module, requires to interface with another application. Usually two the applications are very different. An API is created to provide an abstracted interface, so that all the complexity of how an application works is hidden away from the interfacing application. This abstraction also means that the underlying application can be changed without having to worry about all the other applications that interface to it, as long as the API is unchanged. Since this project requires an interface with Google Scholar, it is very important that the interface part of the project is separated out from the main application. Not only it is simpler to work this way, but it also means that you can easily change the API without having to worry about the application breaking.

2.8 Screen Scraping

Screen Scraping is a technique used to parse data from the output of programs which were designed for humans [scr07]. This technique is different from normal parsing because the output contains information which is not structured in any way for a program to read easily. The output contain mixture of content, style and formatting. Separating the

³An application programming interface

content from all other information can be very difficult for computer programs, specially if the output contain errors.

Normally when transferring data between two programs you use a structured protocol which both programs understand. The programs confirm to this protocol, and data which does not confirm is ignored or an exception is generated. These protocols are often not understood by humans.

With the advent of the Internet many people have begun publishing websites. These websites contain information designed for human readability. Search engines which index web pages on the Internet have to use Screen Scraping techniques to separate the content from the styling. The indexing technologies that are used by search engines can only be applied to the contents of the web pages. Standards have been adopted to make this process easier. Standards such as XHTML⁴ forces structure onto web pages. Style and content of the web page is separated into two files. Applications can be developed to read these web pages and can also be made to understand them.

2.9 Usability Engineering

Usability rules the Web. Simply stated, if the customer can't find a product, then he or she will not buy it. The Web is the ultimate customer-empowering environment. He or she who clicks the mouse gets to decide everything; all the competitors in the world are but a mouse click away.

Designing Web Usability - Jakob Nielsen

Usability is a way to describe how effectively a user can interact with a product. A good user interface should be intuitive. It is important to note that Google Scholar has been designed for simplicity in mind. Since this project intends to add features and functionality to this simplicity, studying how websites can be designed without harming its simplicity is vital. To achieve this goal, user analysis is required to determine what features are useful to the user and how they should be implemented on the web page. This can be done through the use of mock-ups and then collecting feedback from the users themselves. Conventions are very important when designing any user interface [N⁺94]. A user expects the title of a web page to be at the top of the page. They expect company icons to be at the top left corner of the page [Nie99]. If these conventions are not followed users tend to get quite frustrated and confused at how they should navigate between the web pages. Consistency is also very important. If every page on a website contains totally different layout then the users will get very disorientated and it will get very difficult for them to be able to utilise the websites functionality properly. Accessibility of a website needs to be considered carefully. A successful web user interface would allow every user of the website to be able to access all the features of the website without degradation. Since this project is a web application which is designed to be used by people it is important to take a more User-centered design philosophy when building the user interface.

⁴eXtensible HyperText Markup Language. An extention of HTML as an application of the XML language.

Chapter 3

Requirements Analysis

3.1 Introduction

This chapter analyses the requirements and develops a specification which this project will be developed on. The design principle which will be followed strictly through out this development is the User-centered design principle. This design principle was chosen for the sole reason that this application heavily depends on the user interface. If the user interface is not designed properly, it does not matter how many features are created, the user will not be able to utilise them. The user will be fully involved during this requirements analysis stage. Much of the work in this chapter analyses what functionality the user might want. The user is then given the chance to evaluate each feature and then a specification is generated. Throughout this section the user is fully involved with the process of gathering the feature list.

3.2 Related Proposals

Existing solutions available on the Internet has already been introduced in chapter 1. In this section we will analyse some of the most popular and free academic search engines. We will identify features and functionality which should prove useful in this project.

3.2.1 Citeseer

Citeseer¹ is a search engine which mostly focuses on computer and information science. Citeseer was the first search engine to provide automated citation indexing and citation linking. You first start by typing in your query. The results are shown as a list of articles which matches your query. Citeseer provides summary of each article in the list, which includes title, abstract, and number of citations. When you click on an article you are taken to its page which includes all the information Citeseer has gathered about this article. One very interesting feature of Citeseer is that it provides a graph which shows the number of citations per year for an article, an example of this is shown in figure 3.1. Other features of Citeseer includes list of similar articles and link to a location of the article. Citeseer provides a feature which no other search engine provide. They provide statistics about articles and the relationships between them. For example, an author can

¹<http://citeseer.ist.psu.edu/>

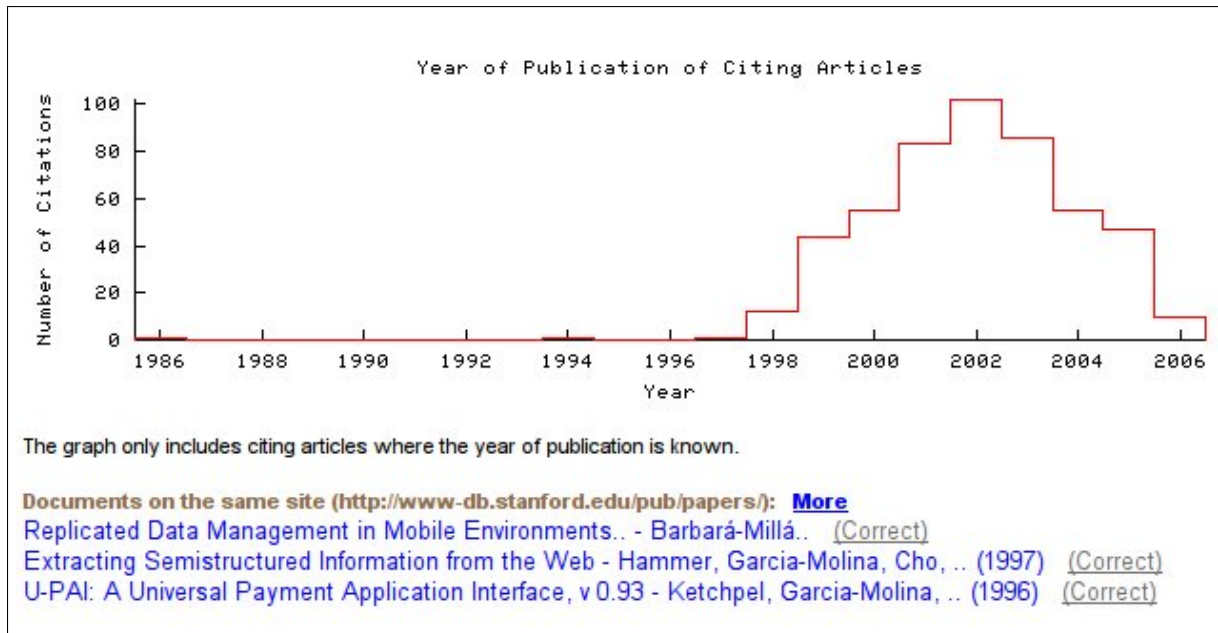


Figure 3.1: Citations per year graph generated by CiteSeer for an article

check the number of times they have been cited. They can also see how they are ranked globally, by comparing the number citations, an example of this is shown in figure 3.2. These features might not be useful for researchers, however it is interesting to see the results of these statistics.

3.2.2 DBLP

DBLP² is another search engine for computer science related articles. One of the main difference between this and other search engines is that DBLP is organised in a hierarchical manner, an example of this is shown in figure 3.3. The main page does not contain a search text field where you type in a query. You actually see a list of categories. You can browse the database by traversing through the categories. There are categories for conferences, journals, and subject areas. It does have a search feature. The results page for search is a table of articles containing summary. You can click on an author to see all the articles the author has published. One interesting feature about the authors page is that it contains a table with a list of co-authors, an example of this is shown in figure 3.3. Each author has a hyperlink to their own author page.

3.2.3 The ACM Portal

ACM Portal³ is one of the more feature packed academic search engine. It has the usual search feature, which displays the articles as a list. But, the amount of detail it contains about each article no other search engines can rival. Each article has complete details about the author and which institutions they are from. There is abstract and complete references of each article. There is a feature where a list of terms are displayed, so you

²<http://www.informatik.uni-trier.de/~ley/db/>

³<http://portal.acm.org/>

Most cited authors in Computer Science - August 2006

Generated from documents in the [CiteSeer.Continuity](#) database. This list does not include match, or citations where the relevant author is an editor. An entry may correspond to multiple authors. Citation counts may differ from search results because this list is generated from a total of 790329 authors were found. Homepages listed may not be for the most cited individual. Click on HPSearch to see and update the latest homepage data.

[Next 250](#)

1. D. Johnson ([HPSearch](#)): 16227
2. J. Ullman ([HPSearch](#)): 13245
3. A. Gupta ([HPSearch](#)): 10156
4. R. Rivest ([HPSearch](#)): 9967
5. R. Milner ([HPSearch](#)): 9878
6. S. Shenker ([HPSearch](#)): 9456
7. V. Jacobson ([HPSearch](#)): 8659
8. S. Floyd ([HPSearch](#)): 8487
9. M. Garey ([HPSearch](#)): 8485
10. R. Tarjan ([HPSearch](#)): 8269
11. E. Clarke ([HPSearch](#)): 7909
12. J. Smith ([HPSearch](#)): 7893
13. L. Lamport ([HPSearch](#)): 7759
14. J. Dongarra ([HPSearch](#)): 7722
15. L. Zhang ([HPSearch](#)): 7284
16. D. Knuth ([HPSearch](#)): 7269
17. R. Agrawal ([HPSearch](#)): 7073
18. R. Karp ([HPSearch](#)): 6833
19. C. Papadimitriou ([HPSearch](#)): 6816
20. H. Zhang ([HPSearch](#)): 6802

Figure 3.2: Citeseer - Most cited authors in Computer Science - August 2006

Search

- ◆ [Author](#) - [Title](#) - [Advanced](#)

Bibliographies

- ◆ **Conferences:** [SIGMOD](#), [VLDB](#), [PODS](#), [ER](#), [EDBT](#), [ICDE](#), [POPL](#), ...
- ◆ **Journals:** [CACM](#), [TODS](#), [TOIS](#), [TOPLAS](#), [DKE](#), [VLDB J.](#), [Inf. Systems](#), [TPLP](#), [TCS](#), ...
- ◆ **Series:** [LNCS/LNAI](#), [IFIP](#)
- ◆ **Books:** [Collections](#) - [DB Textbooks](#)
- ◆ **By Subject:** [Database Systems](#), [Logic Prog.](#), [IR](#), ...

Figure 3.3: DBLP - Category listings



Coauthor Index

1	Scotty Allen	[2]
2	Serdar Badem	[1]
3	Jake Engleman	[2]
4	Roger King	[1] [2]
5	Clayton Lewis	[2]
6	Emil Meng	[2]
7	Jon Raphaelson	[2]
8	Michael D. Williams II	[1]

Figure 3.4: DBLP - Coauthor Index

can click on these terms to reveal more articles in that subject area. There is also a list of keywords for each article. Figure 3.5 shows an example of an article from ACM Portal.

3.2.4 Google Scholar

Even though this project is based of Google Scholar, it is important that we analyse all the features provided by Google Scholar. As mentioned already Google Scholar works just like a normal search engine. You get a list of articles which relate to your query. Google Scholar does provide some extra features which other search engines do not. When each article is displayed as a list, they contain a fragment of the text where the search query occurs in the article, an example of this is shown in figure 3.6. For each article in the results page there are some actions you can perform on them. For example, if you want a list of all the articles which cited a particular article you can see it by clicking of the "Cited By" link. There is also a "Related Articles" feature where Google Scholar uses its proprietary algorithms to gather articles which relate to a particular article. As mentioned previously Google Scholar's database is actually a combination of all other academic search engines. Google Scholar provides a link to the location of each article. Google Scholar also allows you to click on an authors name and see a list of all the articles they have written. Other features include direct link to your local library and multiple source locations for each article.

3.2.5 Conclusion

Analysis of the existing solutions has provided many valuable features and functionality which could prove to be very useful when combined with Google Scholar's database. After analysing alternate solutions it is clear that Google Scholar lacks the functionality that other search engines provide.

Google Scholar coverage of a multidisciplinary field

Source **Information Processing and Management: an International Journal** [archive](#)
 Volume 43 , Issue 4 (July 2007) [table of contents](#)
 Pages: 1121-1132
 Year of Publication: 2007
 ISSN:0306-4573

Author [William H. Walters](#) Helen A. Ganser Library, Millersville University, Millersville, PA 17551-0302, USA

Publisher Pergamon Press, Inc. Tarrytown, NY, USA

Additional Information: [abstract](#) [references](#) [index terms](#) [collaborative colleagues](#)

Tools and Actions: [Find similar Articles](#) [Review this Article](#)
[Save this Article to a Binder](#) Display Formats: [BibTex](#) [EndNote](#) [ACM Ref](#)

DOI Bookmark: [10.1016/j.jpm.2006.08.006](#)

↑ **ABSTRACT**

This paper evaluates the content of Google Scholar and seven other databases (Academic Search Elite, Social Sciences Abstracts, and Social Sciences Citation Index) within the multidisciplinary subject area. The content of these databases is evaluated with reference to a set of 155 core articles selected in advance-the most important studies published in 2000. Of the eight databases, Google Scholar indexes the greatest number of core articles (93%) and has the most up-to-date coverage. It covers 27% more core articles than the second-ranked database (SSCI) and 2.4 times as many as GEOBASE. At the same time, a substantial proportion of the citations provided by Google Scholar are to abstracts (33%).

Figure 3.5: The ACM Portal - An article page

[The Google file system - group of 134 »](#)
 S Ghemawat, H Gobioff, ST Leung - Proceedings of the nineteenth ACM symposium on Operating Systems Principles
 Page 1. The **Google** File System Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung
Google * ABSTRACT We have designed and ...
[Cited by 199](#) - [Related Articles](#) - [Web Search](#) - [Import into BibTeX](#) - [Find It via JRUL](#) - [BL Direct](#)

[A Pattern Language - group of 2 »](#)
 C Alexander ... - New York, 1977 - ai.wu-wien.ac.at
 ... PUBLISHER = {Oxford University Press}, SERIES = {Center for Environmental Structure Series}, ADDRESS = {New York, NY}, QUERY = {http://www.google.com/search?q ...}
[Cited by 1071](#) - [Related Articles](#) - [Cached](#) - [Web Search](#) - [Import into BibTeX](#)

Figure 3.6: Google Scholar - Example search result

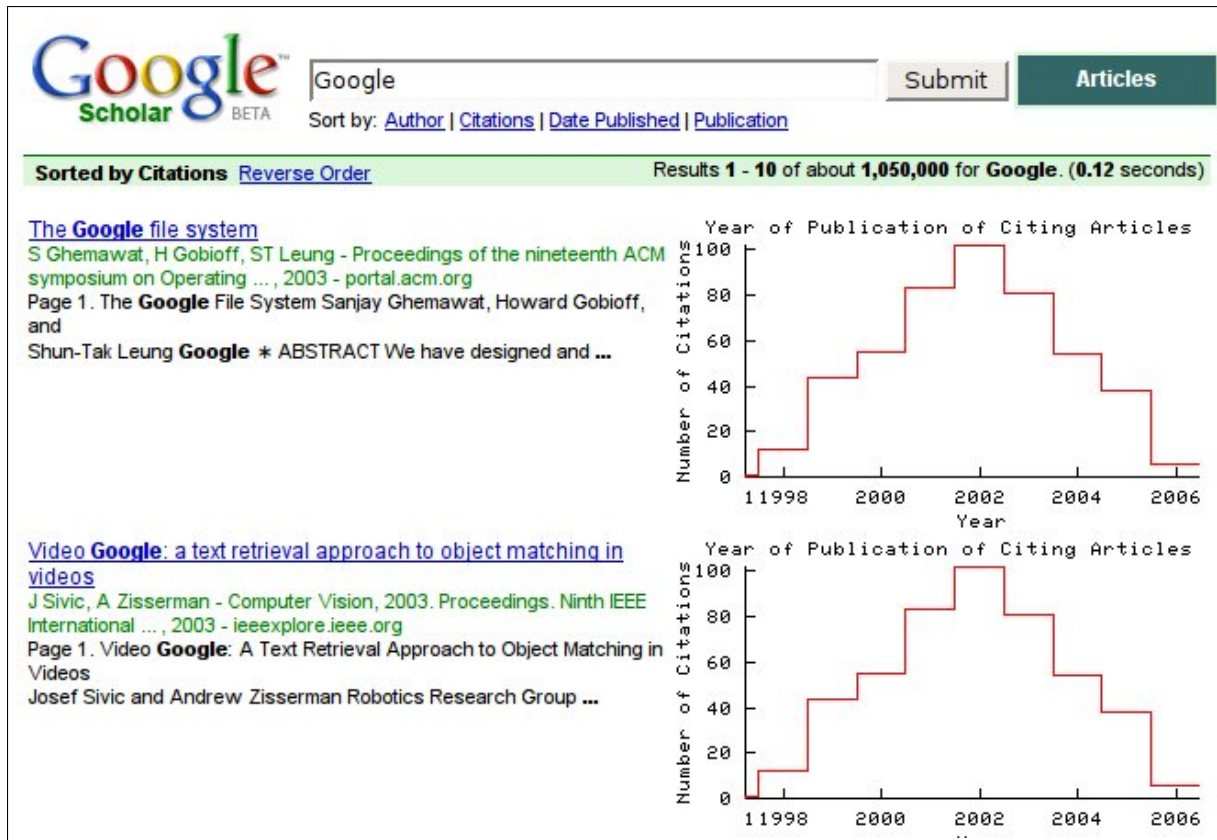


Figure 3.7: Prototyping Phase 1 - Results page

3.3 Prototyping

To effectively generate user requirements it was required to create prototypes which mimic a real system to allow users to properly evaluate its effectiveness. Mock-ups were created using static HTML and images. Prototyping underwent a number of phases until a final draft was created which the real system would be built on. The prototyping phases are outlined below:

3.3.1 Phase 1

First phase involved creating the core pages, such as the results page. Figure 3.7 shows a screenshot of the results page. The page shows a list of articles, with the same data as Google Scholar. It was decided at the time to include a graph of citations per year for each article. You would then be able to click on the image and go to a bigger version of the graph. The mock-up was created to mimic the style and formatting of Google Scholar.

3.3.2 Phase 2

In this phase it was decided that creating a graph for each article on the results page would be too costly, and also it would clutter the results page. Figure 3.8 shows that the graph has been removed. A new feature has been added to the list of actions that you can

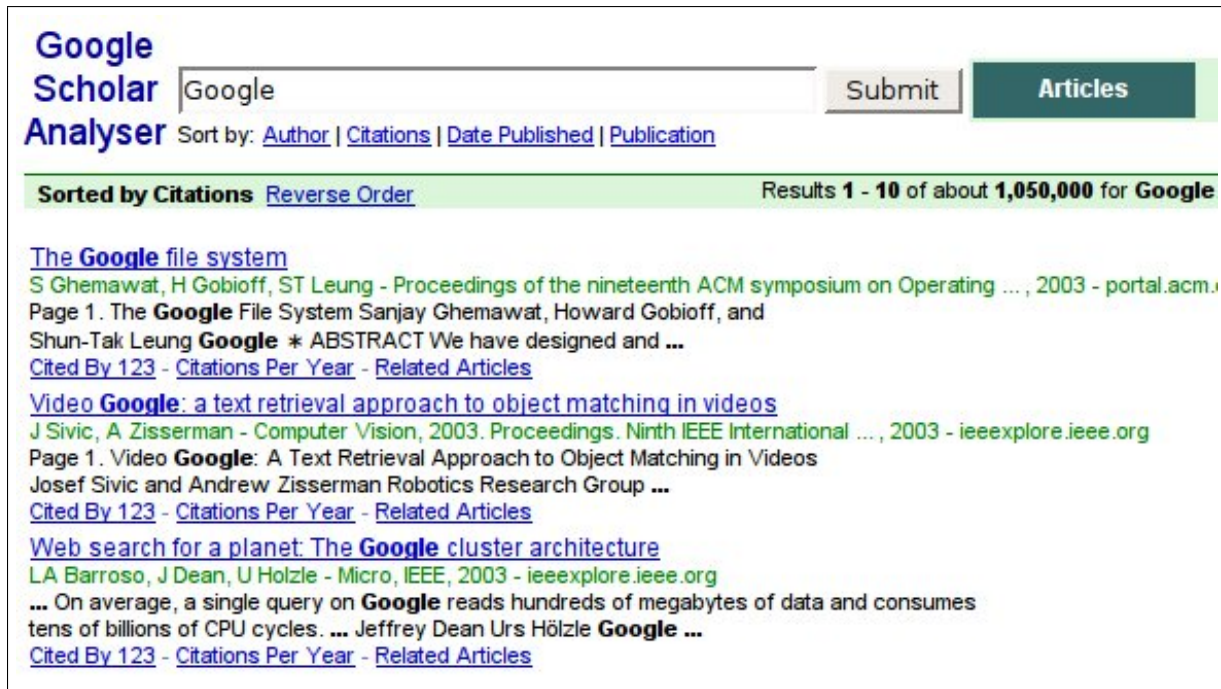


Figure 3.8: Prototyping Phase 2 - Results page

perform on individual articles. You can click on the "Citations Per Year" link and view the graph in a full page. Figure 3.9 shows an example of what this page would look like. The article which the graph belongs to is also shown in the page. This allows the user to perform other actions of the same article without having to go back to the previous page. Figure 3.10 shows the "Cited By" feature. The layout is essentially the same as Google Scholar, however it was decided that including the article which this action is performed on should be at the top of the page, in the same way it is for "Citations Per Year" page. This results in a consistent layout, so that user understands that they are performing an action on an individual article. Also notice that a green bar is used to display the type of action and the results are tabbed right to separate them from the article. There is also a similar page for "Related Articles".

3.3.3 Phase 3

This phase incorporates many new features to enhance Google Scholar. Figure 3.11 shows the results page. Now there are actions which apply to not only articles, but also authors and the results set. New features include pie charts to show statistics on the results show in figure 3.12, perform an action on combined multiple articles, combined citations per year graph, citations tree in figure 3.13 and citations per year for authors.

3.3.4 Phase 4

There were dramatic changes at this phase. It was decided to design a new style and formatting for this project. The design was kept simple and clean. The layout is consistent with Google Scholar, so therefore it should be familiar to the users. Figure 3.15 shows

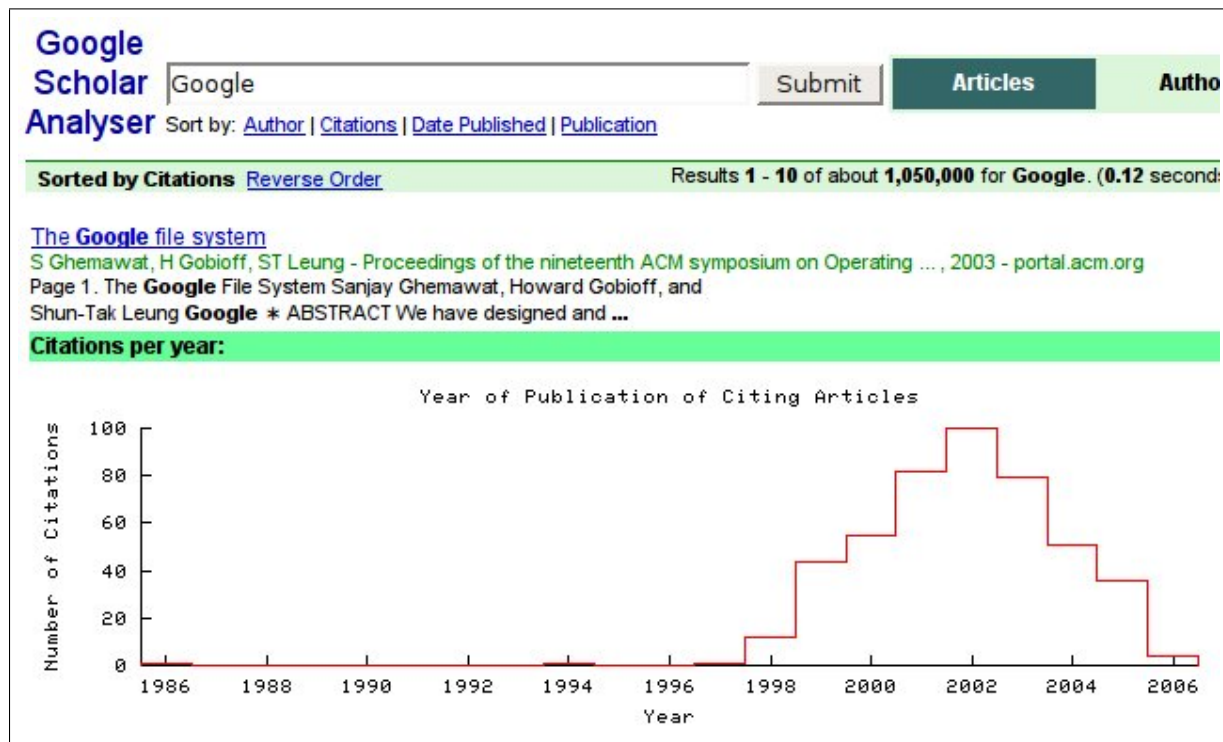


Figure 3.9: Prototyping Phase 2 - Citations per year

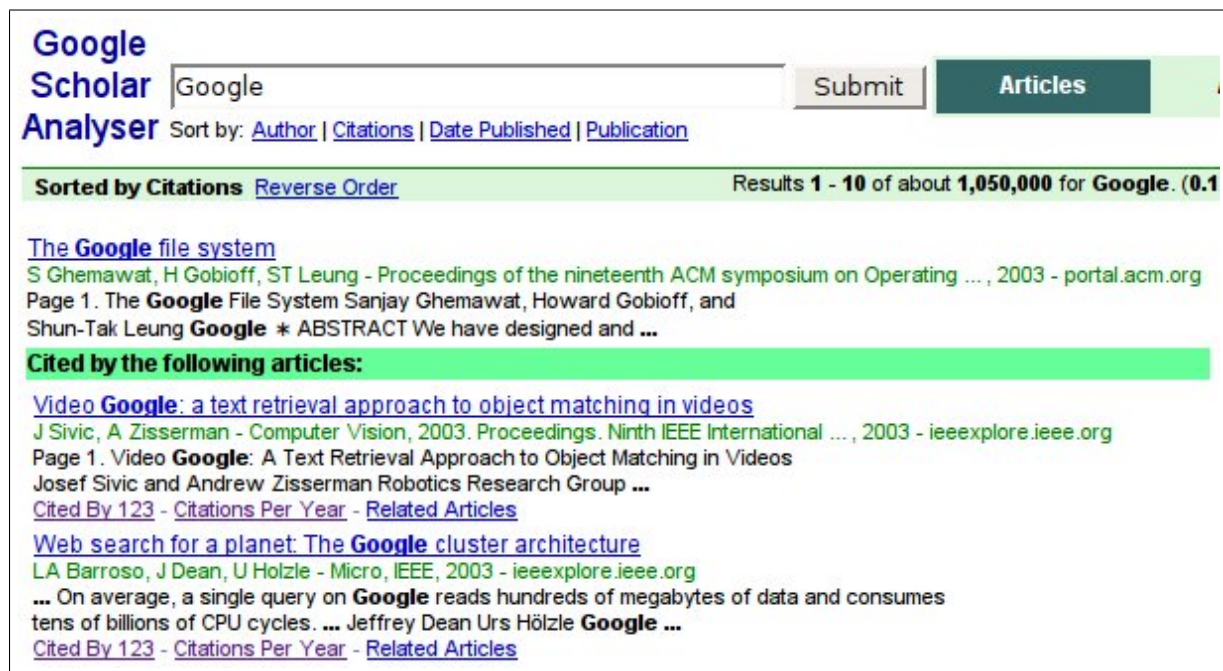


Figure 3.10: Prototyping Phase 2 - Cited By



Figure 3.11: Prototyping Phase 3 - Results page

the front page, which only holds a text box and a button. This follows the simplicity of Google Scholar front page. However a new features has been added. It has been decided to include a live search feature. Small set of results will be displayed at the bottom of the search box as the user types in the query.

shows an example set of live search results. Only article title and author name is searched and displayed. This search will be carried out on the cache only, so that there is minimum delay. Figure 3.17 shows an example of the results page. This new design is much cleaner, and provides greater flexibility to arrange the data. As you can see the layout is pretty much similar to Google Scholar. There is an author list of the left of the results. Actions which can be performed on individual articles are located just below each article. There is sorting which allows the user to sort the results page by citations, title, date and publisher. User can see statistics of the results by clicking on one of the links located at the top right corner. Combined multiple articles actions have now been moved to the bottom of the results page. This is a common location for actions which rely on the user having to select multiple items before pressing on the buttons. Since last phase, all actions which can be performed on authors have been moved to a new location. It was decided to create a page for authors and articles. This allows better flexibility because we can now include more data about each author and each articles. Figure 3.18 shows an example of the articles page. It is easy to distinguish that this page is about an article. There is a list of alternate locations for this article on the left of the page. There is a summary of the article at the top of the page. The screenshot displays a number of new features that has been included since last phase. There is a bar at the top of the page which displays a trail, or breadcrumbs, of all the pages which the user has previous

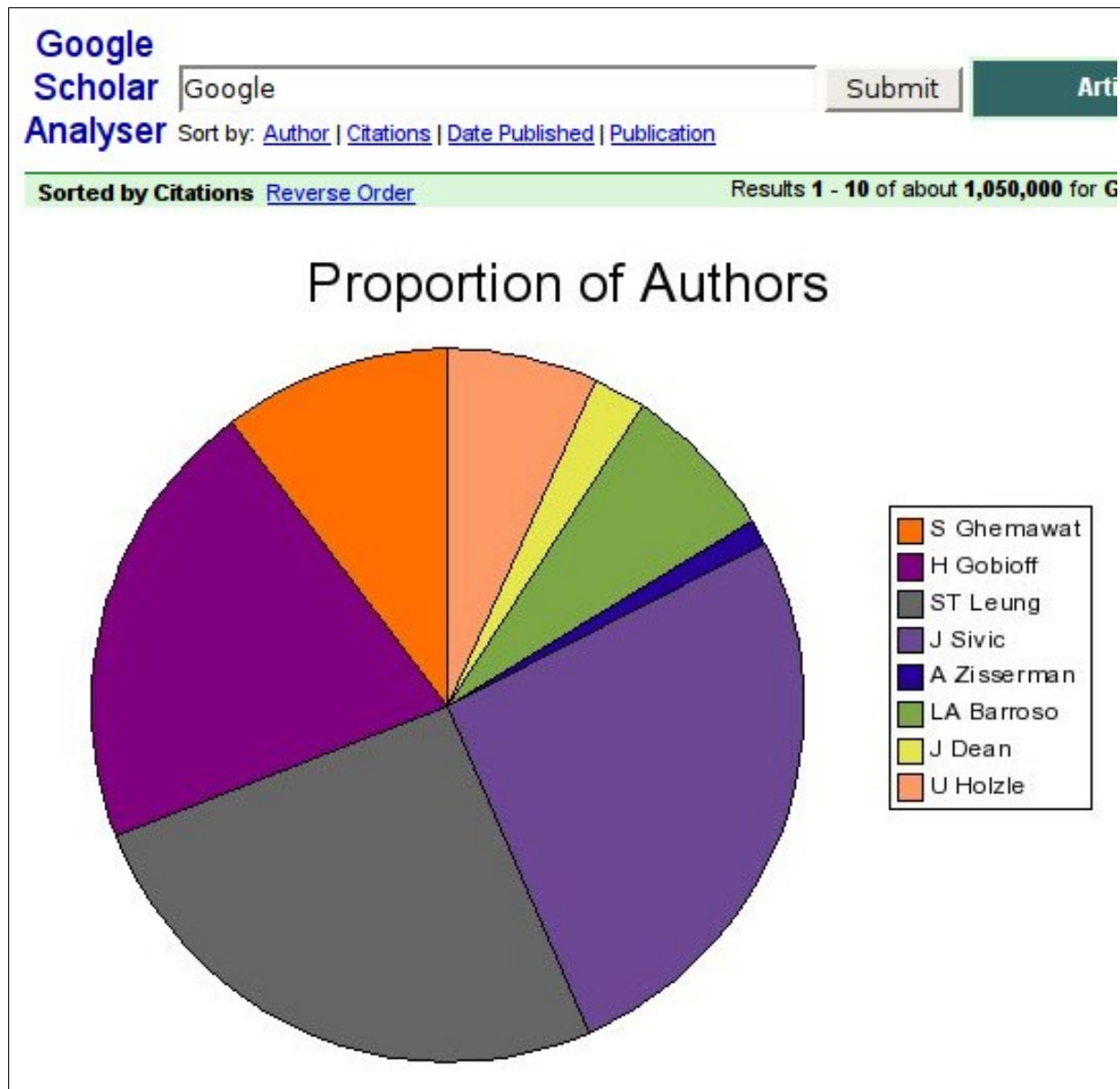


Figure 3.12: Prototyping Phase 3 - Statistics pie chart

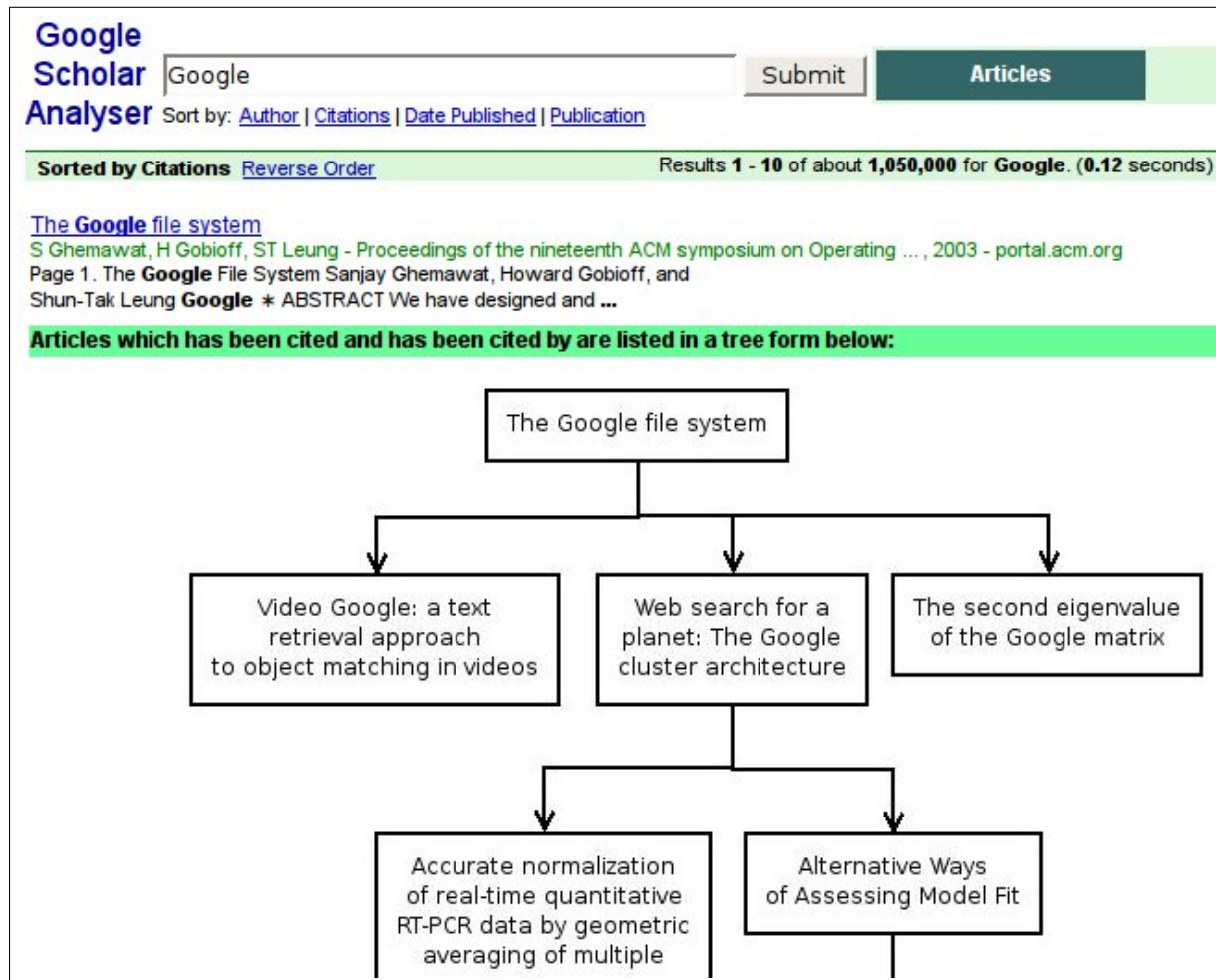


Figure 3.13: Prototyping Phase 3 - Citation Tree

Google Scholar Analyser

→ Search "Google" → "The Google file system" → "Conversation and Cooperatio..." → "Video Google: a text retriee..."

Showing article "Video Google: a text retrieval approach to object matching in videos"

Available Links:

- [ieeexplore.ieee...](#)
- [robots.ox.ac.uk](#)
- [cs.gmu.edu](#)
- [wisdom.weizman...](#)
- [vis.uky.edu](#)
- [vicos.fri.uni...](#)
- [graphics.stanf...](#)
- [lear.inrialpes.fr](#)
- [cogvis.fri.uni...](#)
- [vision.caltech...](#)
- [inrialpes.fr](#)
- [doi.ieeecomput...](#)
- [cs.duke.edu](#)
- [portal.acm.org](#)
- [robots.ox.ac.u...](#)

15 links

Video Google: a text retrieval approach to object matching in videos
Authors: [J Sivic](#) - [A Zisserman](#)
 Computer Vision, 2003. Proceedings. Ninth IEEE International ..., 2003

[Cited by 131](#) - [Citations Per Year](#) - [Related Articles](#) - [Import into BibTeX](#) - [Web Search](#)

Abstract

We describe an approach to object and scene retrieval which searches for and localizes all the occurrences of a user outlined object in a video. The object is represented by a set of viewpoint invariant region descriptors so that recognition can proceed successfully despite changes in viewpoint, illumination and partial occlusion. The temporal continuity of the video within a shot is used to track the regions in order to reject unstable regions and reduce the effects of noise in the descriptors. The analogy with text retrieval is in the implementation where matches on descriptors are pre-computed (using vector quantization), and inverted file systems and document rankings are used. The result is that retrieved is immediate, returning a ranked list of key frames/shots in the manner of Google. The method is illustrated for matching in two full length feature films.

Top 10 Authors Who Cited This Article The Most

Author	No Of Citations
C Schmid	8
A Zisserman	8
J Sivic	6
S Lazebnik	6

Figure 3.14: Prototyping Phase 4 - Articles page

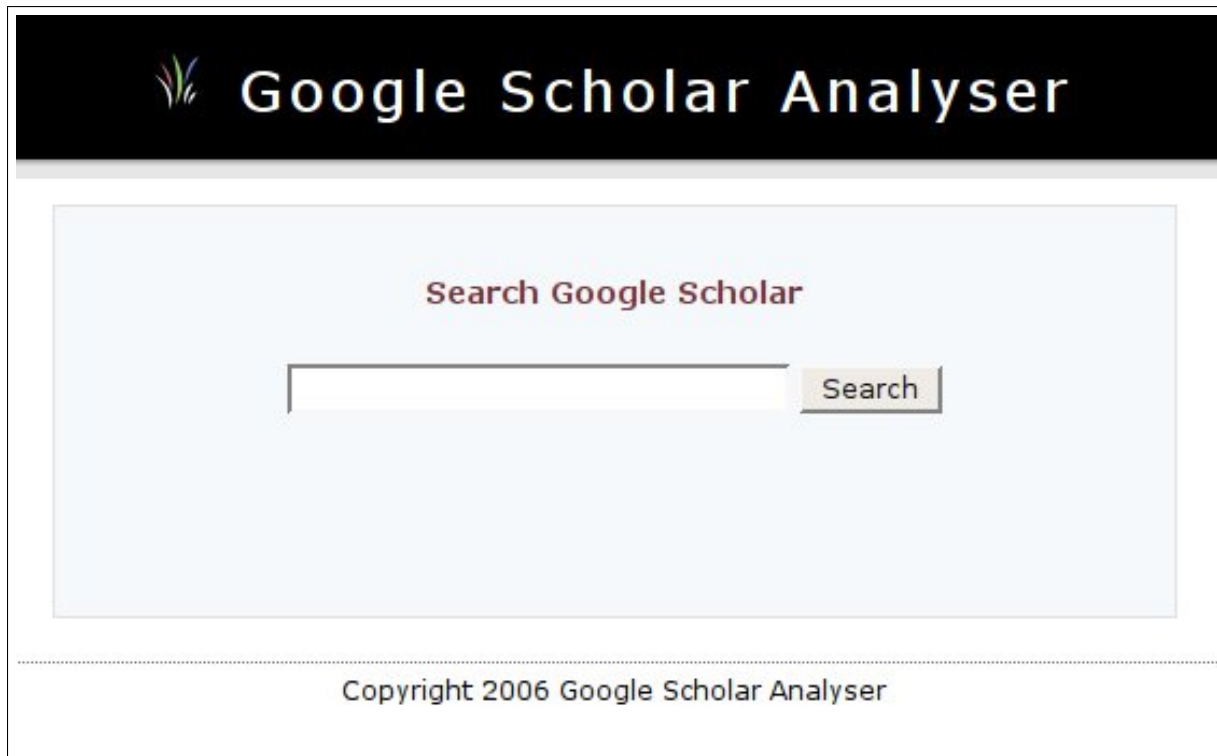



Figure 3.15: Prototyping Phase 4 - Front page

visited. The abstract of the article is displayed on this page also. Near to the bottom of the page shows a table of top 10 authors who cited the article the most. Since last phase a new feature has been added. Now you can also see citations per year which does not include self citations. This feature has also been included in the "Cited By" page. The authors page includes a list of coauthors on the left side of the page. There is a list of action which can be performed on this author. There is also a table containing the top 10 coauthors. One key aspect, which is common to all the mock-ups in this phase, is that all author names and article names are now hyperlinks. This allows the user to get information of a particular author or article without having to search for them. Along with the citations per year for authors, there is a new feature which has been added for authors. This feature is called publications per year.

3.3.5 User Evaluation

To allow users to properly evaluate the mock-ups a questionnaire was created. The questionnaires were given to PhD students who are regular users of Google Scholar and other academic search engines. Since the mock-ups were just simple static HTML pages, the users were able to use them by clicking on hyper links and get a feel for the real system. The questionnaire included statements which the user was required to rate true or false. Appendix A shows an example of this questionnaire which detailed a complete list of features of the mock-ups. The feedback from these mock-ups were then analysed to produce the user requirements of the actual system, which will be discussed in the following section.

 **Google Scholar Analyser**

Search Google Scholar

Available Articles

- [The Google file system](#)
- [Video Google: a text retrieval approach to object matching in videos](#)
- [Web search for a planet: The Google cluster architecture](#)
- [The second eigenvalue of the Google matrix](#)
- [Porous-coated hip replacement. The factors governing bone ingrowth, stress shielding, and clinical ...](#)

Available Authors

- [H Gobiuff](#)
- [M Gore](#)
- [A Goldfarb](#)
- [H Goldblatt](#)
- [LJ Goldstein](#)

Figure 3.16: Prototyping Phase 4 - Front page with live search



Figure 3.17: Prototyping Phase 4 - Results page

3.4 Feature Cost Analysis


Once a list of features has been drawn up which will be implemented in the real system, it was important to analyse execution of these features. The cost is represented as the amount of time required to complete each feature or action. Since we are interfacing with Google Scholar directly there is a bottleneck which needs to be considered. Google Scholar is an external application, and is not owned by this project. There is no direct access to the Google Scholar database. We can only gather the data through the interface that is provided. The application needs to query Google Scholar for every action that the user needs to perform. This cost of performing each action or feature will be discussed and addressed during implementation.

3.5 User Requirements

The following is a list of features generated from analysing the existing systems, prototyping and from user feedback:

3.5.1 Overall System

- Live search for author and article title
- Breadcrumbs for history of pages visited


Google Scholar Analyser

→ Search "Google" → "The Google file system" → "Conversation and Cooperatio..." → "Video Google: a text retriee..."

Showing article "Video Google: a text retrieval approach to object matching in videos"

Available Links:

- [ieeexplore.ieee...](#)
- [robots.ox.ac.uk](#)
- [cs.gmu.edu](#)
- [wisdom.weizman...](#)
- [vis.uky.edu](#)
- [vicos.fri.uni...](#)
- [graphics.stanf...](#)
- [lear.inrialpes.fr](#)
- [cogvis.fri.uni...](#)
- [vision.caltech...](#)
- [inrialpes.fr](#)
- [doi.ieeecomput...](#)
- [cs.duke.edu](#)
- [portal.acm.org](#)
- [robots.ox.ac.u...](#)

15 links

[Video Google: a text retrieval approach to object matching in videos](#)
Authors: [J Sivic](#) - [A Zisserman](#)
 Computer Vision, 2003. Proceedings. Ninth IEEE International ..., 2003

[Cited by 131](#) - [Citations Per Year](#) - [Related Articles](#) - [Import into BibTeX](#) - [Web Search](#)

Abstract

We describe an approach to object and scene retrieval which searches for and localizes all the occurrences of a user outlined object in a video. The object is represented by a set of viewpoint invariant region descriptors so that recognition can proceed successfully despite changes in viewpoint, illumination and partial occlusion. The temporal continuity of the video within a shot is used to track the regions in order to reject unstable regions and reduce the effects of noise in the descriptors. The analogy with text retrieval is in the implementation where matches on descriptors are pre-computed (using vector quantization), and inverted file systems and document rankings are used. The result is that retrieved is immediate, returning a ranked list of key frames/shots in the manner of Google. The method is illustrated for matching in two full length feature films.

Top 10 Authors Who Cited This Article The Most

Author	No Of Citations
C Schmid	8
A Zisserman	8
J Sivic	6
S Lazebnik	6

Figure 3.18: Prototyping Phase 4 - Articles page

- Cache database for already retrieved data

3.5.2 Results Page

- Complete author list
- Author list sorting
- Results sorting by number of citations, title, publish date, publisher
- Combined cited by
- Combined related articles
- Combined citations per year graph
- Combined citations per year graph without self citations
- Statistics for authors pie chart
- Statistics for publisher pie chart
- Statistics for publication year pie chart

3.5.3 Articles Page

- Display summary of article
- Top 10 authors who cited the article the most
- No. of Authors by No. of Citations graph
- Alternate links to the article
- Cited by without self citations
- Citations per year graph
- Citations per year without self citations graph

3.5.4 Authors Page

- Articles written by author
- Complete list of co-authors
- Top 10 co-authors
- Publications per year graph
- Citations per year graph
- Top 10 citing authors

3.6 System Requirements

The system requirements determine the function of the system. This does not necessarily effect the user requirements. The functional and non-functional requirements are described separately.

3.6.1 Functional Requirements

The following is a list of requirements which the system must implement:

- Must be able to generate different types of graphs.
- Must provide statistical data for analysis.
- Must include all existing features from Google Scholar.
- Must use a caching system to reduce number of queries to Google Scholar.
- Must use Google Scholar as the primary source of data.

3.6.2 Non-Functional Requirements

The following is a list of requirements which the application must abide by.

- It must be a web application.
- User interface must be intuitive and simple to use.
- User interface must be designed to be familiar to Google Scholar users.
- User interface must follow web standards, to allow maximum compatibility with different web browsers.
- Must follow a 3-tiered architecture, which includes separate application and presentation layers.
- The system must follow a modular design.
- Must be able to easily adapt system for new features and technology.

3.7 Data Modelling

We now need to describe the data and its relationship in a model so it can be used during design of the architecture. Entity relationship diagram can be used to represent a conceptual model of the data being stored or transferred. To build this diagram we analyse the all the data given by Google Scholar about each article, author and all other entities. We can then form the relationships between each entities. Figure 3.19 show a complete entity relationship diagram representing all the data involved in this project. This diagram will be used to design the database for the cache, and also will be used by the Google Scholar API.

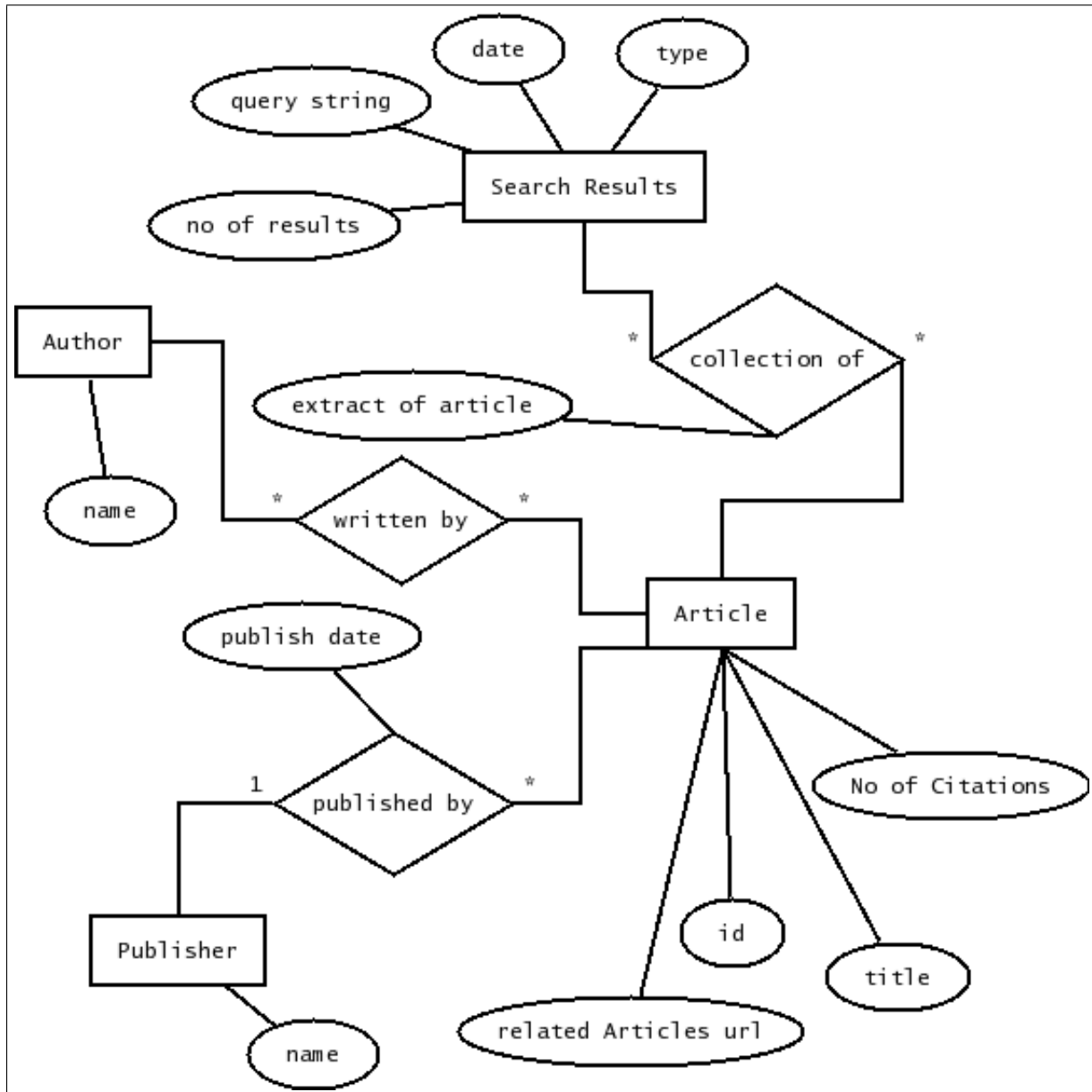


Figure 3.19: Entity Relationship Diagram

3.8 Conclusion

The requirements specifications which were generated through extensive analysis of existing solutions, user analysis and some imagination. The features which are to be implemented has been evaluated by users, through mock-ups, and have been judged to be very useful.

Chapter 4

Design

4.1 Introduction

This chapter describes steps taken to draw up a fully functional system design which will then be used during implementation stage. Most of the designing work will heavily involve creating a fully functional and modular architecture which can then be used to implement all the features that were decided during requirements analysis. It is essential that the architecture is extensible. The features will need to be implemented on top of the core application, one after the other, this type of development requires a very flexible architecture design.

4.2 Software Architecture

The architecture which this application will be built on is the Model View Controller architecture. This involves separating the data, user interface and the application logic. Most web applications on the Internet are designed using this architecture, where a relational database is used for data storage, dynamically generated HTML is used as user interface, and application logic is done through a server side programming language. Apart from the obvious modular design, this architecture will mean that changes to the user interface will not affect the data handling, nor will changing the data affect the user interface. Also the application logic, which usually contains complex algorithms, can be separated from the user interface and data handling. Figure 4.1 shows a simple Model View Controller architecture diagram.

4.2.1 Model

The Model is the domain specific data which the application operates on. This includes the data storage and handling of the data. In this application the Model will be the database used to cache data which has been collected from Google Scholar. However, since Google Scholar itself is an external system, which this application needs to connect to, it can be looked at as a Model and can be treated as such. The application will need to connect to the Google Scholar's database, through its web interface.

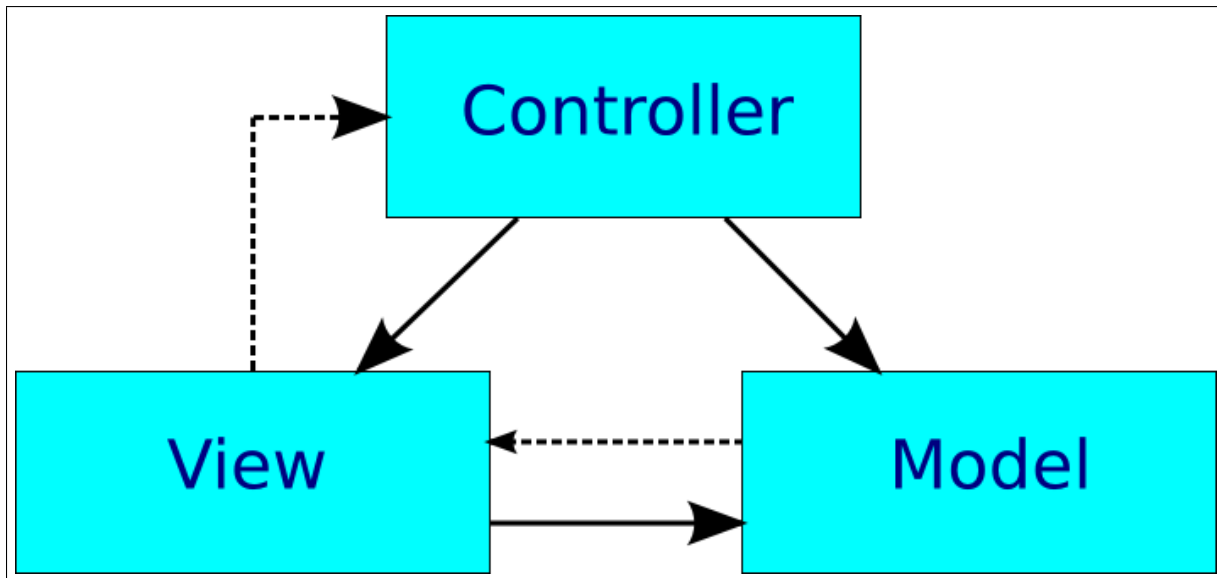


Figure 4.1: Model View Controller architecture diagram

4.2.2 View

This is the only part of the application which the user will ever be aware of. This is essentially the front end of the application. A web application generally has dynamically generated HTML web pages as its View. The view of this application is the mock-ups which have already been designed and evaluated by users. However, since the mock-ups are only static HTML web pages, they need to be converted and then integrated into the application View. Although most of the application logic will be inside the Controller, there may be some logic in the View. Such logic will only determine the presentation aspects, such as displaying certain items on a web page depending on who you are logged in as.

4.2.3 Controller

This is the brain behind the application. The Controller will work like a middleman between the View and Model. It will take the data from Model and apply application logic, which then will be displayed to the user through the View.

4.3 Overall Architecture Design

After considering the Model View Controller architecture and how we interface with Google Scholar a design of the overall architecture has been created, as shown in figure 4.2. This design outlines the major components of the architecture.

4.4 Class Diagram

Now we can draw up a detailed diagram which represents all the major components of the system. Figure 4.3 shows design class diagram of the whole system. You can

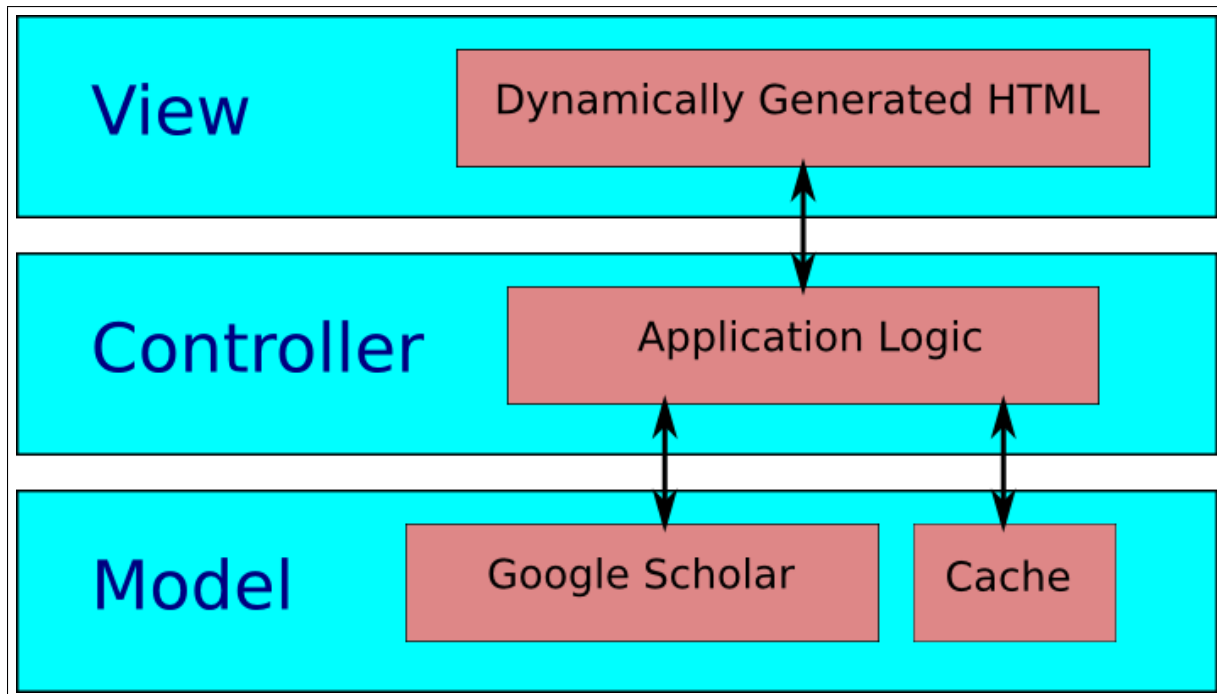


Figure 4.2: Overall Architecture Design

already gather that the main part of the application is the `GoogleScholarAnalyser` class. The `GoogleScholarAnalyser` class does not interface with `GoogleScholarAPI` class. It connects to `GoogleScholarAPI` indirectly, through the `Cache` class. This allows `Cache` class to determine if Google Scholar should be queried, or the data should be retrieved from cache. `Cache` class also interfaces with the `CacheStorage` class which manages the connection and the storage of the data. The reasoning behind separating `Cache` and `CacheStorage` was to break up the `Cache` class in to two distinct functions. This will provide low coupling to the `Cache` class, so if we ever have to change how we store our data we only need to modify the `CacheStore` class. The whole system is designed with high cohesion in mind. Each component only perform single precise task, meaning we have better modularity in our design.

4.5 Conclusion

This chapter has highlighted the overall architecture design. For each task, the system is broken in to classes to provide a modular design. The components only interface with other components that they need. This chapter only describes how the underlying system is to work. The user interface has already been discussed in previous chapter. We are now ready to implement the system, component by component.

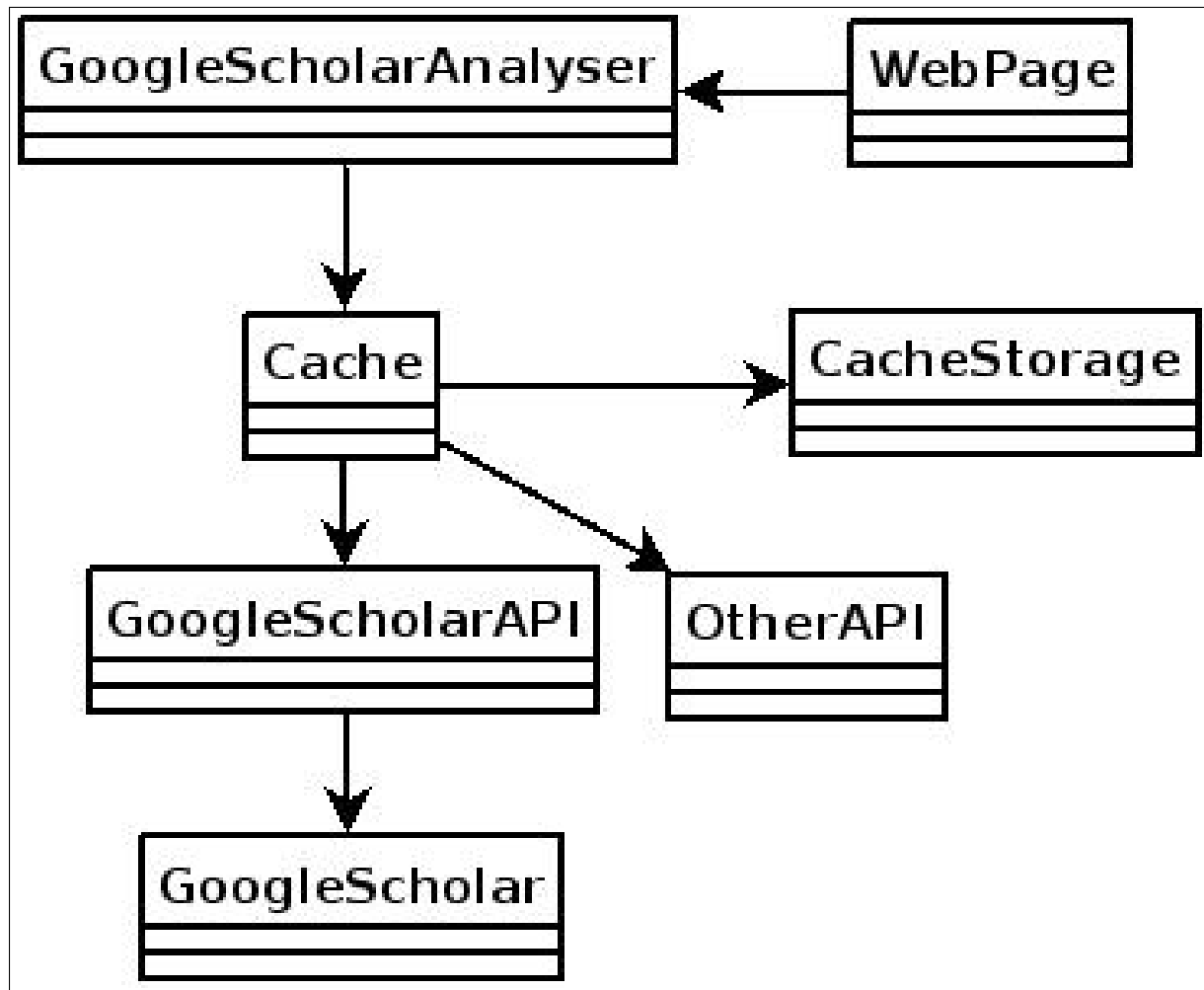


Figure 4.3: Design class diagram

Chapter 5

Implementation

5.1 Introduction

This project has been designed and implemented in an iterative process. The application was designed and built while requirements analysis was being carried out. This is a very flexible process which allowed this project to evolve into an application which is closer to what the users want. While the requirements analysis were being carried out the core architecture of the application was designed and implemented. Once the requirements analysis was completed, it was then implemented in intervals. At each interval there were implementation and testing. This proved to be a very natural process as a developer. The system could be changed significantly without losing much time and throughout the whole implementation process the system would be tested thoroughly.

This chapter looks at the implementation stage of the project life cycle. All available technologies are outlined. Technology which will be used to implement the application will be described and justified. Further detail of the the technologies, such as libraries of modules being used will also be discussed. There will be extensive writing on how the system is implemented, from the core architecture to all the user requirements. How the application is tested, with description of the testing framework being used, is also described. Problems which were faced during implementation will be described, along with solutions that were used. At the end of this chapter we should have a fully working application which has gone through thorough testing.

5.2 Programming Languages

The main focus of this project is to build a web application. To implement this application a programming language needs to be chosen which is appropriate for building web applications. Essentially any programming language can be used to implement a web application, using Common Gateway Interface (CGI)¹. However, not all programming languages are designed to be used this way. Some are just too low level, so therefore it will be very complex to create a fully fledged web application. Three popular program-

¹Common Gateway Interface is a standard protocol for interfacing a Web server with a common application. It allows requests to be passed from the client web browser to an application and then return the output of that application to the client.

ming languages have been chosen to be discussed in this section. These languages are very popular for web development. The advantages and disadvantages of each language will now be debated.

5.2.1 Java

Java is an object-oriented programming language developed by Sun Microsystems. The Java programs are compiled into bytecode, which is then executed in a virtual machine. This means that the Java applications are platform independent. The syntax is very similar to C and C++. Java is a popular language for web development. There are many technologies developed with Java which allow developers to creating advanced web applications with ease. Such technologies include Java Applets, Java Servlets and JavaServer Pages. Java Servlets and JavaServer Pages are executed server-side which then generated HTML web pages for the client web browser. This means that the clients do not require any special software to access the web applications, it is treated just like a web page. However, Java Applets is executed in the client side and therefore requires a special Java plugin to be installed in the web browser. Java Servlets will be considered as a viable technology for this project. Example: simple loop in java:

```
for (int i = 0; i < 10 ; i++)  
-  
    System.out.println("Number: " + i );  
"
```

5.2.2 PHP

Hypertext Preprocessor (PHP) is a programming language specifically designed for building dynamic web pages. PHP runs on the server-side, taking PHP code as its input and then generating Web pages as output. PHP can be deployed in most web servers and operating systems. The syntax is quite similar to C and C++. PHP5 was released to include robust object-oriented programming. PHP is a very popular language when developing dynamic web pages on the Internet. PHP is available for free, as it is opensource. PHP is supported in most Internet Service Providers which host websites. It is the most cheapest and easiest language to use for developing dynamic websites. There are many libraries available which simplify development in PHP. For example you can use PEAR (PHP Extension and Application Repository), which provides extensive reusable libraries and components for rapid application development. Example: simple loop in PHP:

```
for ($i = 0; $i < 10 ; $i++)  
-  
    echo "Number: $i";  
"
```

5.2.3 Ruby

Ruby is a dynamic object-oriented programming language which is fast becoming the most popular language for developing highly advanced web applications. Its simple, clean and human readable syntax has made it very popular for scripting and prototyping.

Its creator Yukihiro Matsumoto has been quoted to say "The purpose of Ruby is to maximize programming pleasure"². In recent years many new exciting applications have been developed in Ruby, including the Ruby On Rails, a rapid application development framework for web applications. Ruby has characteristics of other programming languages, such as Perl and Smalltalk. For example, Perl is well known for its built in support for Regular Expressions³. This feature has been implemented in Ruby. Example: simple loop in Ruby:

```
10.times do —i—  
  puts "Number: #i"  
end
```

5.2.4 Java Servlets

Java Servlets provide an API for web developers to include dynamic content to their Web server using the Java platform. The API provides a standard interface which defines how a Servlet interacts with a Web Container. A Servlet is essentially an object that receives a request and then it generates a response from this request. If Java Servlet is used to develop the web application, there are many technologies that need to be researched and considered. Java Servlets require a web server and a container to work. Without these two components you cannot run Java Servlets. This environment is very complex for a small project like this.

5.3 Web Framework

It is not enough to decide which programming language to use based on their individual programming features. Other considerations must be taken. Building a web application from scratch is a complex process. It is better to take advantage of frameworks, if available, so that your development time is reduced. The frameworks are designed to alleviate the overhead that is common when building an application this size. Since most web application follow a standard architecture there is no point in building something from scratch. Frameworks are tried and tested set of libraries, modules and management environment which should be utilised if available. There are components in place which you can take advantage of, for example most application framework's integrate unit testing. Following are three frameworks which can be utilised for this project.

5.3.1 Google Web Toolkit

Google Web Toolkit (GWT) is a Java development framework which allows you to design the user interface in Java, and then GWT compiler converts this browser compliant into HTML and JavaScript. GWT is designed for creating AJAX web user interface. GWT can directly interface with the web server using Java serialization⁴. If developing in Java

²<http://ruby.mirror.easynet.be/en/testimony.html>

³In computing, a regular expression is a string that is used to describe or match a set of strings, according to certain syntax rules. For more information visit <http://en.wikipedia.org/wiki/Regular-Expressions>

⁴Java serialization is encoding data structure as a sequence of bytes for the purpose of transfer objects from one application to another.

this framework is perfect for building your front end. Since you are writing Java code, you can take advantage of all the Java debugging technology that is available. GWT also integrated JUnit⁵ testing framework. Even though this is an excellent framework for building the front end, it does not provide for the back-end. You would still require some work in the back-end for databases access, for example.

5.3.2 Apache Struts

Apache Struts is a framework which uses Java Servlets. The goal of Struts is to cleanly separate Model from the View and the Controller. It follows the Model View Controller architecture design. The View takes advantage of writing templates, typically using JavaServer Pages, which is essentially HTML with embedded Java code. Struts is well documented, mature and very popular framework for building web applications in Java. There many IDE's⁶ which support Apache Struts and allow you to develop your application quickly and easily. Apache Struts can be quite cumbersome to set up, because it requires many configuration information and a web server which supports Java Servlets.

5.3.3 Ruby on Rails

Ruby on Rails is a web application framework built using the Ruby programming language. The main goal of this framework is to increase development speed of your application. The philosophy behind Ruby on Rails is "Convention Over Configuration"⁷ and the "Don't Repeat Yourself". The whole framework is designed specifically for database driven websites. The web applications can be quickly built using the skeleton code which Ruby On Rails comes with by default. For example, the most popular feature of the framework is "scaffolding". This feature allows you to have READ, WRITE, UPDATE and DELETE actions on a database table without having to write any code. The feature analyses the table structure and dynamically generates the web pages necessary to do all the actions on a table. This and many more features were specifically designed for ease of development. There is integrated testing framework for unit testing. Using this framework and writing code in Ruby you can increase productivity.

5.4 Webserver

No matter which programming language or framework is chosen, a webserver is still required to run the application. We will discuss three webserver, one for each programming language.

5.4.1 Apache with mod php

To run a PHP web application you need a server which supports PHP. Apache is a very popular webserver, which also supports PHP after installing a module. There is a lot of

⁵JUnit is a unit test framework for the Java programming language.

⁶Integrated Development Environment.

⁷http://en.wikipedia.org/wiki/Ruby_on_Rails

documentation available which describes how to install and set up an Apache webserver with PHP support. Configuring the server can be quite tricky, however it is manageable.

5.4.2 Apache Tomcat

To run a Java web application you need a a Servlet Container. Apache Tomcat is a very popular container which integrates with Apache webserver. To run a Java application you would need Apache Tomcat and the Apache webserver. Tomcat's configuration is very complex. It requires modification of XML⁸ configuration files.

5.4.3 Mongrel

Mongrel is a simple webserver which is written purely in Ruby. The webserver integrates seamlessly into Ruby On Rails framework. You can run Mongrel in a development mode where it provide more data for debugging purposes. Mongrel requires minimal configuration. It only needs the location of your Ruby On Rails application and which port to run on.

5.5 Chosen Technology

Having looked at three programming languages and a corresponding framework it seems like one language and framework consistently stand out. Ruby On Rails seems to have the best framework which would suit this project. The programming language Ruby is object-oriented and has Regular Expression feature which will allow easy implementation of the Google Scholar API. The webserver Mongrel is very simple to use, which makes Ruby On Rails a very convenient environment for developing web applications.

5.6 Cache Technology

The project requires a cache to save the data which has already been downloaded from Google Scholar. This should reduce the number of queries made to Google Scholar. Now we need to decide how this cache will be implemented.

We can store the data in a file where we choose our own data structure. However, storing the data in a file means that we cannot query the data. We can only read the whole file into memory and then access specific information. The flat file option might look easy, but it will add many complexity to the code. For example, we would need to create our own file structure to store the data.

Relational database would allow us to store the data in way so that we can execute queries over this data. This option would reduce a lot of coding complexity because querying a relational database is something very common for web applications, so it is easy to implement. Since we already have an Entity Relationship diagram of our data structure, we can use this to build a database schema.

Using a relational database is the best option. This would reduce the amount of work needed to implement the Model part of the application. Now that we have decided to

⁸XML (Extensible Markup Language)

use a relational database, it is important to choose the right database server. There are two choices in this regard. We can go with a fully fledged multi-threaded and multi-user SQL database management server, or we can use a self-contained, embeddable, zero-configuration SQL database engine called SQLite⁹. If we are to use a database management server we can use MySQL¹⁰. However, using a fully fledged server which requires configuring and managing for a small project is not the best way to spend the development time. SQLite is a database engine which does not require any configuration and yet still offer most of the features that are expected from a full server. Also SQLite does have overhead due to multi-user management and authentication. The database is stored in one file local to the application. Using SQLite is the best solution here, especially during implementation. Once the application is complete and has been deemed stable, then a full database server can be used to store the cache.

5.7 Screen Scraping Techniques

Web page screen scraping is the only way we can interface with Google Scholar, as Google does not provide an API for Scholar. There are a number of techniques that can be applied to extract data from a web page. Firstly, we can treat each web page like a string. We can do search and replace on that string. This is where Regular Expression would come in handy. Treating the web pages as string does mean that if Google changes web page for Scholar even slightly the API would break. The second technique which can be implemented to extract data from a web page is to use some code to parse the HTML before trying to extract data. This generally involves making a tree of some sort to represent the hierarchical nature of web pages. At glance this seems like the best option for parsing data. HTML web page might look good in a web browser, yet be seriously malformed, tags unclosed or misused. If a web page is malformed the parser does not work. The tree that is created might be totally different, so extracting data this way is also very unreliable. Parsing the web page also means that there is some preprocessing which must be done before actually trying to extract the data. Ruby has excellent string manipulation features, along with Regular Expression. The API would be better implemented if we treat the web page as a string. This would also mean that there is no need for preprocessing, or parsing, of the web page.

5.8 Further Detail on Ruby On rails

Figure 5.1¹¹ shows a complete architecture diagram of Ruby On Rails. The framework has been broken down into three components. The first is the ActiveRecord class. This class interfaces with the database and provides the Model part of the architecture. The View is provided by the ActionView class. The Controller, ActionController class, interfaces with both the View and Model. It contain all the logic behind the application. A number of libraries and plugins were used to implement this project. This section will try and go through each one and explain the reasoning behind why it was necessary.

⁹<http://www.sqlite.org/>

¹⁰<http://www.mysql.org/>

¹¹<http://www.intertwingly.net/slides/2005/imab/mvc.png>

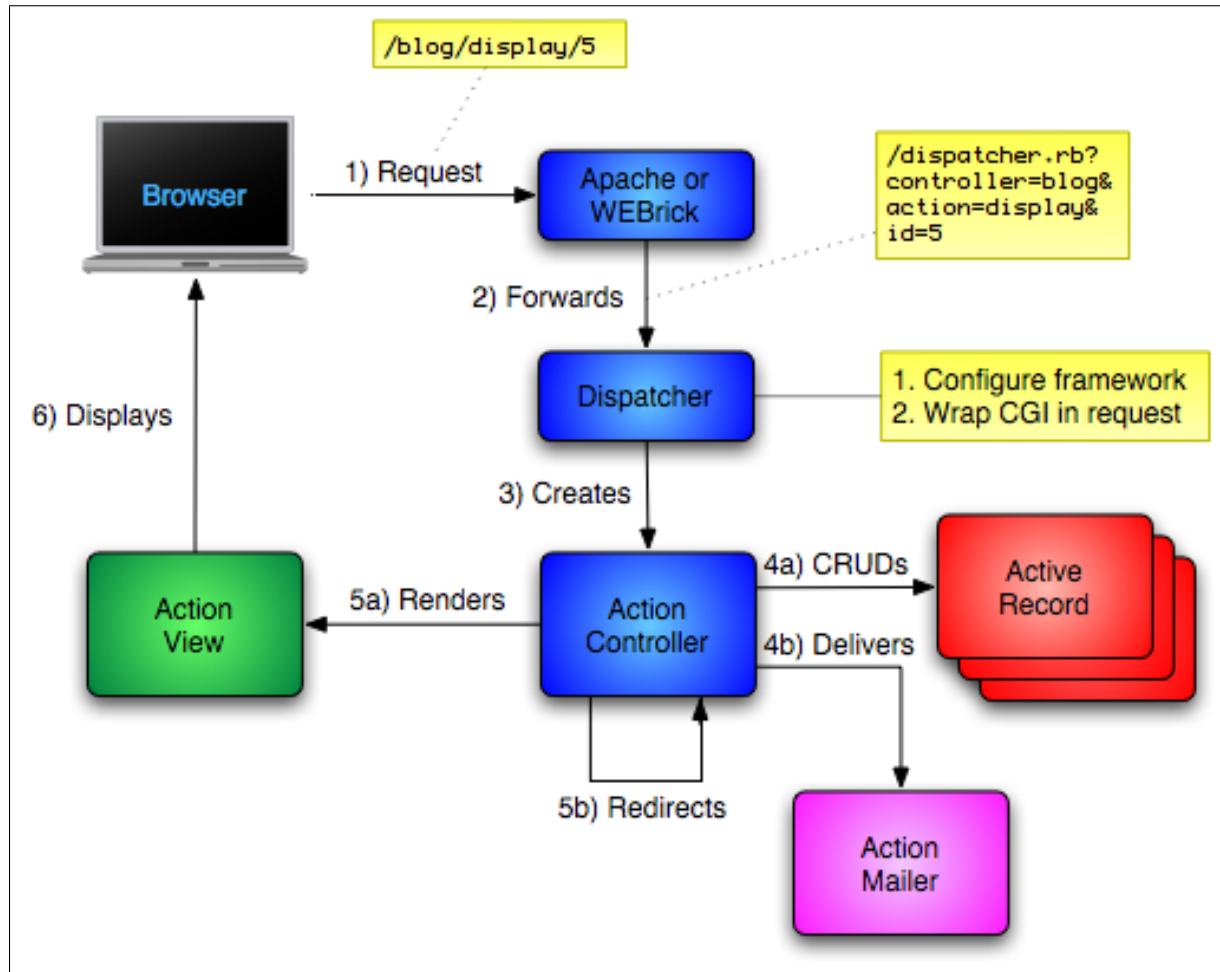


Figure 5.1: Ruby on Rails architecture diagram

5.8.1 HAML Template Engine

Using a Model View Controller architecture gives the flexibility to choose a template engine for the View. For Ruby On Rails there is a template engine called HAML¹². HAML is a template engine which dramatically reduces the amount of code necessary to produce HTML which fully conform to web standards. Example HAML code:

```
#main
.note
  %h2 Quick Example
  %p HAML is very neat and short

  produces:

;div id='main';
;div class='note';
;h2;Quick Example;/h2;
;p;
  HAML is very neat and short
;/p;
;/div;
;/div;
```

As shown in the example, the code that is required to generate the HTML is shorter than the actual HTML. The template engine also allows parts of web pages to be broken into chunks, and then you can reuse these chunks in different parts of the templates.

5.8.2 Regular Expressions

Regular expression is a string that is used to describe a rule to which a string can be matched to extract specific sub-string. Ruby has support for regular expressions which will be fully utilised in this project when implementing the Google Scholar API. For example we can use a method from the String class in Ruby called "scan()" extract only certain parts of the string which matches the regular expression and then return it as an array:

```
=; string = "Hello there, how are you?"
=; substring = string.scan(/e(.*)e/)
```

substring now contains an array of strings which begin with "e" and end with "e", excluding the "e", the array should look something like this:

```
=; [["llo th"], [", how ar"]]
```

Regular expressions can be used to separate each article in the results page in Google Scholar. Then each article can then be parsed using regular expressions to extract the URL, title and other data.

¹²<http://haml.hamptoncatlin.com/>

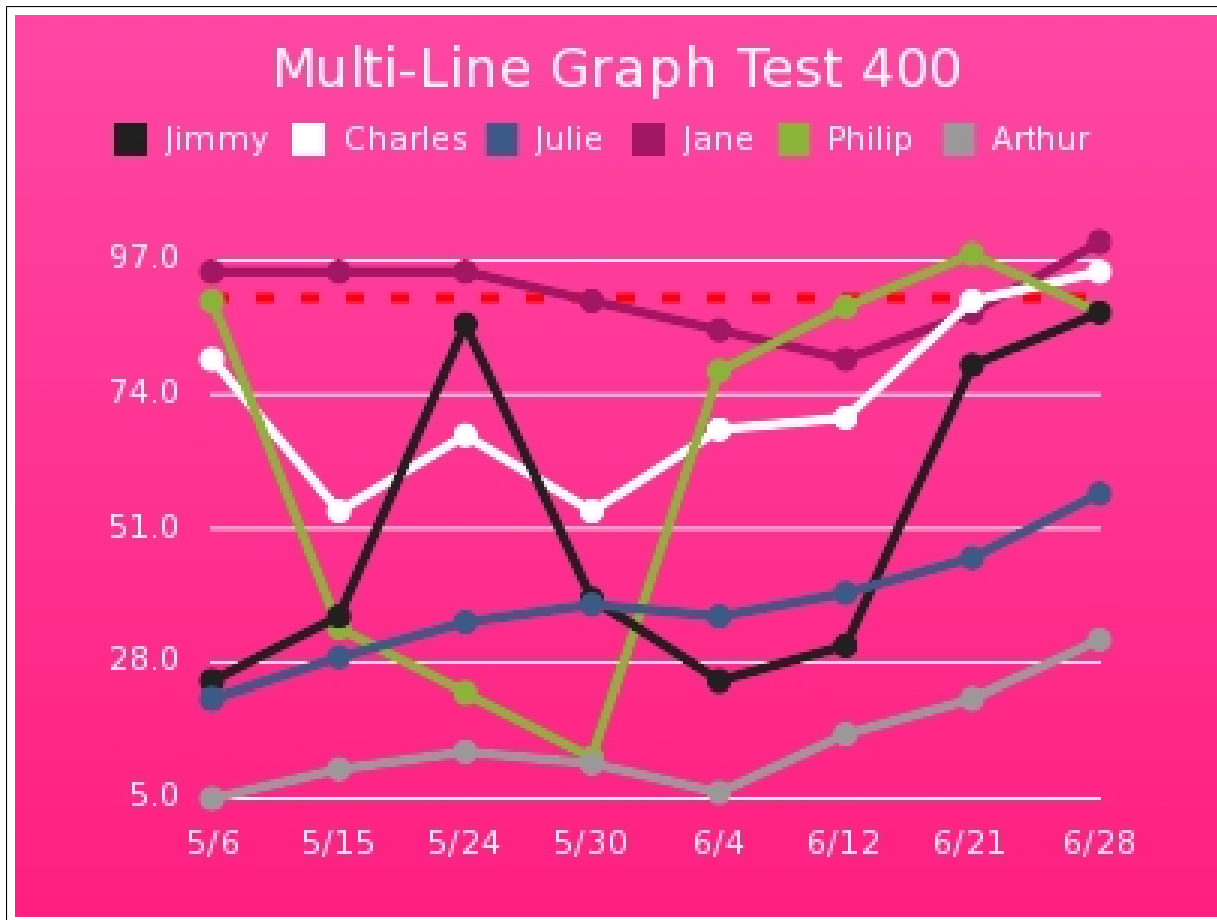


Figure 5.2: Example graph using Gruff

5.8.3 ActiveRecord

ActiveRecord is a major component of Ruby On Rails framework. ActiveRecord is a persistence engine which connects objects to database. It implements the object-relational mapping pattern by representing a table as an object, row as an instance of that object and a column as an attribute. By mapping the relational database to objects we there is no need to use SQL to query the database. ActiveRecord allows you to create, read, update, and delete data from the database without having the developer to write any SQL statements. ActiveRecord supports all major database servers, including SQLite database engine.

5.8.4 Gruff - Graphs for Ruby

Gruff is a library which allows you to create graphs using Ruby. Gruff takes advantage of ImageMagick¹³ library, to generate the graphs. It can be to create many different graphs dynamically. Figure 5.2 shows an example of a graph generated by Gruff.

¹³<http://www.imagemagick.org/>

5.8.5 Testing Framework

Applications are tested to make sure that it works as it should be. Enormous amount of work goes into testing of an application. However, people tend to wait until they have completed implementing the application before starting the tests. It is better to break the tests into smaller units. If you can test each method of each class on its own, then you can write a test after writing the method. This unit testing is what `Test::Unit`¹⁴ library is all about in Ruby. This testing framework has been integrated into Ruby On Rails. Every Controller and Model can be tested on their own. Every method can be tested. Tests are created using assertions. Each assertion needs to succeed for a test to succeed. There are many assertions, from a simple equality assertion to more advanced. This process of creating a test after creating the method will save a lot of time, because bugs will be detected early on the development. If bugs are detected you can turn these bugs into test cases. So that bug is always tested, this will mean that bug will not reoccur.

5.9 Architecture Implementation

After selecting the technology which is required to implement this project we now go into implementing the architecture. The architecture is very important for this project since all the other features will need to be implemented on top of it, one by one. To fully take advantage of the Model View Controller architecture the project has been split into multiple components, each with only single task. Figure 5.3 shows a complete class diagram, including the public method for each individual class. The implementation started with the `GoogleScholarAnalyser` class. Stubs were created to perform the tasks of the other classes which `GoogleScholarAnalyser` class depended up on. Since the API was not functional, a set of static data was created to test the `GoogleScholarAnalyser` class. Once `GoogleScholarAnalyser` class was implemented the work went into implementing the API. At first a static results page from Google Scholar was used to extract data. Later on the API was connecting directly to Google Scholar. After much testing and debugging, development work went into the `Cache` class. At this stage since we are able to query Google Scholar it was appropriate to be able to save this data to cache. Once all this was complete the rest of the classes were implemented. The Ruby On Rails testing framework was fully utilised. Every time a bug was discovered it would be added to the unit testing as a new test. This proved to be quite deterrent for bugs to reoccur. Figure 5.4 shows a complete class diagram after implementation. It contains all the classes that were built, including the Controller's and Model's.

5.10 Implementation of Requirements

Once the architecture has been implemented and deemed stable the requirements which were gathered during requirements analysis were implemented one at a time. The feature list was put in order of importance so the most useful features can be implemented first. Once again the testing framework is used considerably. At this stage lots of new features are added. After each new feature is added the whole application is fully tested to make sure that the new feature does not make anything else. Of course features which generate

¹⁴<http://www.ruby-doc.org/stdlib/libdoc/test/unit/rdoc/classes/Test/Unit.html>

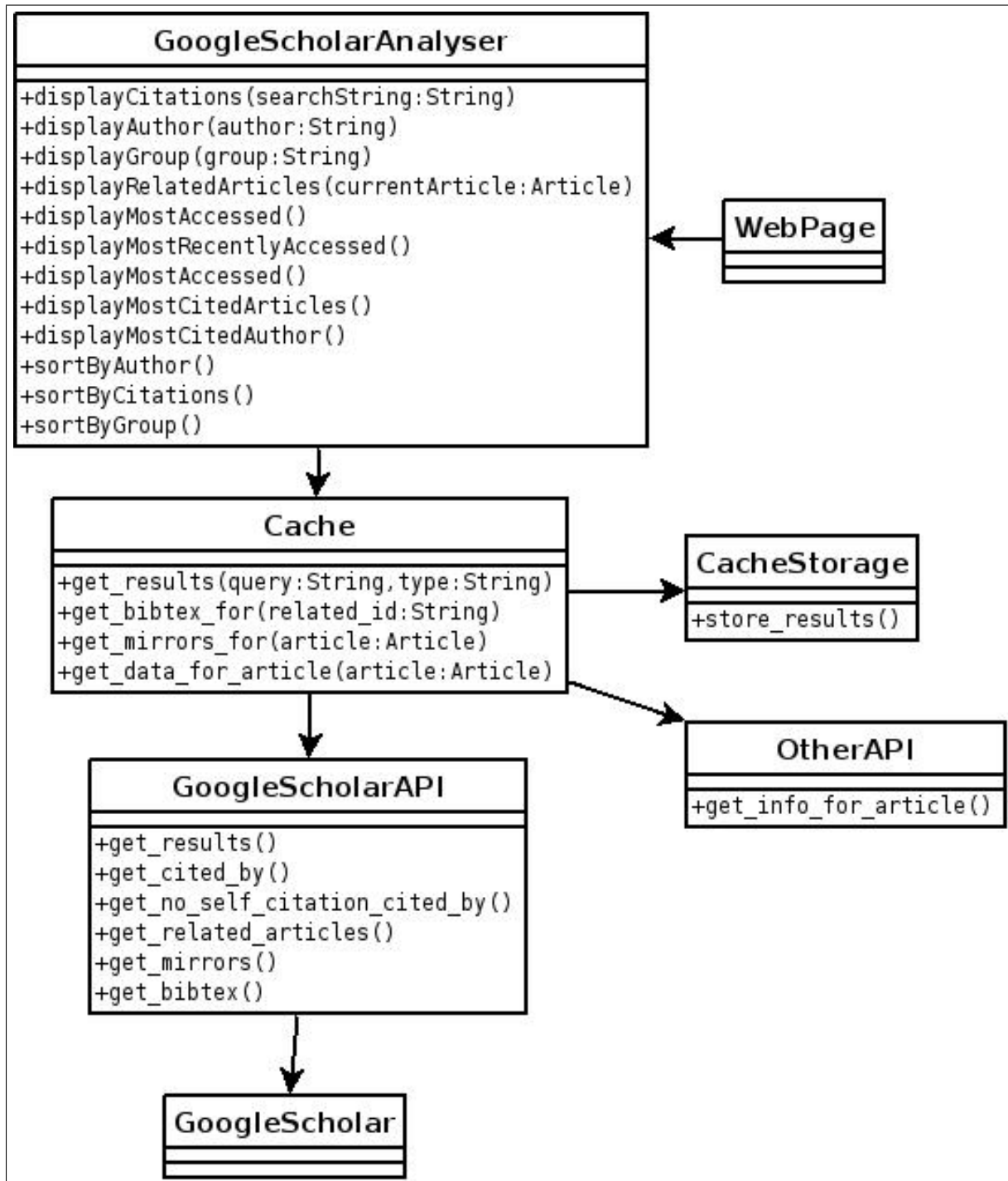


Figure 5.3: Class Diagram

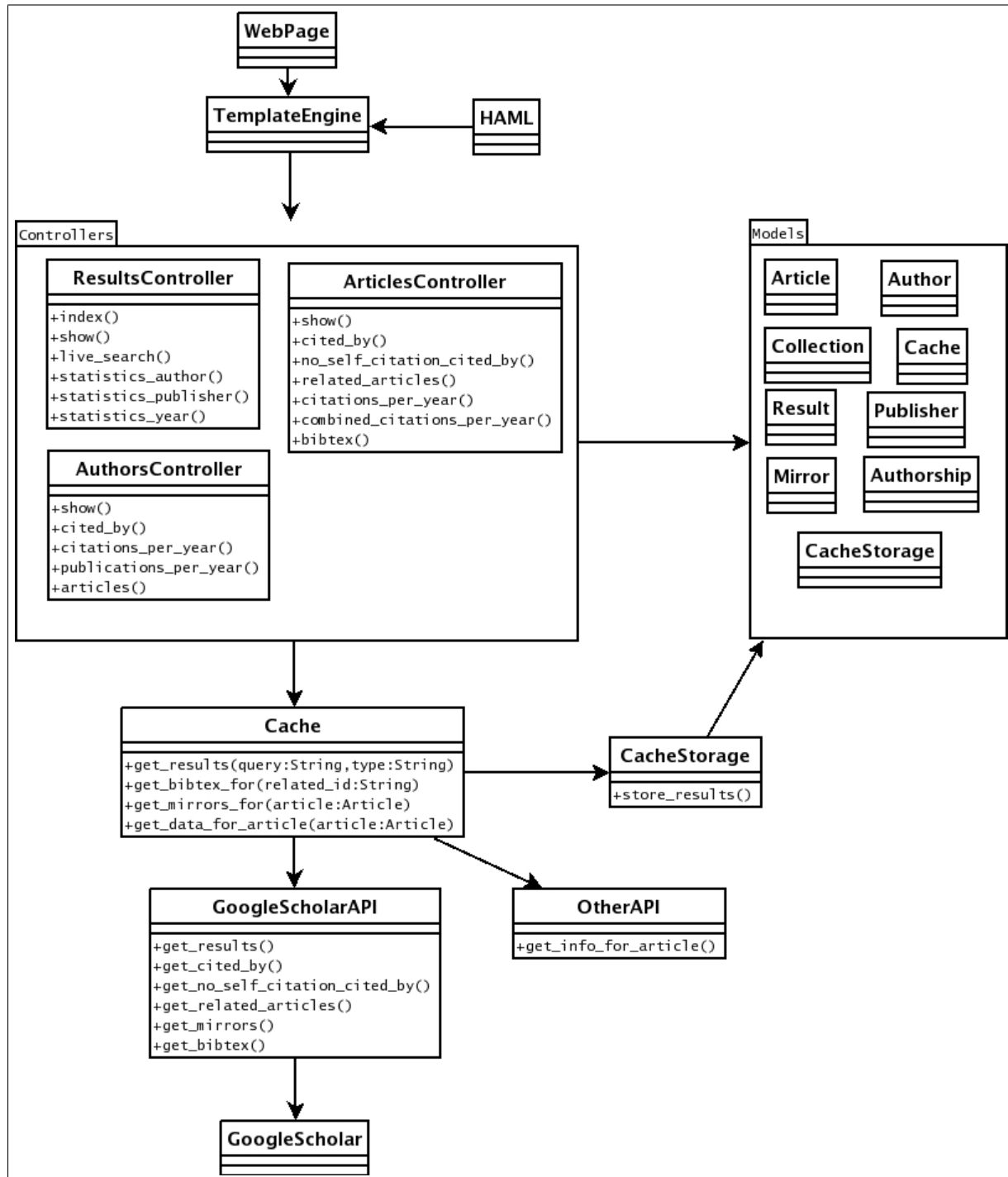


Figure 5.4: Complete Class Diagram

graphs or generate images cannot be tested automatically. However, since there are only few features which generate images, these were tested manually.

5.11 Conclusion

While implementing the system there have been many unforeseen problems. There were times when the API would stop working. This seemed odd since the testing passed. However, after much debugging it was realised that Google has a limit on how many queries you can make per second. This dramatically reduced the speed of some of the features of the system. The only way to avoid being blocked by Google, was to implement a delay after each query to Google Scholar. This solved the problem, but in some ways created a longer execution time for the features. Even though this is undesirable, the solution was accepted as there is no other way to solve this problem.

Chapter 6

Evaluation

6.1 Introduction

The evaluation of this project is concerned with the system evaluation and user evaluation. The system evaluation will assess the system as a whole, its strengths and weaknesses will be identified. The user evaluation is concerned with the user experience. The web user interface will be evaluated according to the user requirements.

6.2 System Evaluation

The system that was developed has been tested throughout the development process. This testing was carried out using unit testing. Every bug that was detected became a test case. The architecture of this system has been designed for flexibility in mind. The architecture should be able to withstand changes and new feature additions without compromising performance or stability. We can safely say the system has met this criterion. This is because the project was developed in an iterative process, where new features were added one at a time. So over time this system has been through many new feature additions and it is still performing as expected. Over the development life cycle Google Scholar has had many changes to its web interface. We would find that every few weeks the test cases for Google Scholar API would fail. At this point development would stop until the API was fixed. The API is designed in a way that each part of it is tested. The API is broken down into many private methods. Each method is used to parse specific data from Google Scholar. This means that if a test case fails it will show exactly which part of the Google Scholar web page is now different. The methods would then be quickly and easily updated to reflect the changes made by Google. This design has saved an enormous amount of time while trying to detect which part of the web page actually changed. Once again this proves that the system is very much capable of adapting to new situations without major changes to the code. This application has fulfilled all system requirements that were set during requirements analysis. This includes being able to generate graphs and the ability to cache data.

6.3 User Evaluation

User evaluation involved releasing the application to the users and letting them test the application. The feedback that were received were generally successful. However, there are some interesting observations that were made. Firstly everyone noticed that the application was much slower then using Google Scholar directly. They found that some of the graphing features were very useful. However, there are other features which would have been useful if implemented. This included showing references that were made by each article. Also being able to see full names of authors were mentioned. The pie charts that were generated were only limited to the number of results that were downloaded. The combining of multiple results is an interesting feature, however it is very rare to find two articles that share the same cited by articles. The combined results feature however did work well with related articles. The user interface was found to be quite intuitive and very familiar to the users. The layout was the same as Google Scholar and they were very happy with the pleasant style of the web pages. All the hyperlinks were named appropriately, and the action that they performed were expected.

6.4 Conclusion

To conclude the evaluation it has to be said that requirements that were set at the beginning of the project were all met. There were some unforeseen challenges that were faced which effected the performance of the application. Otherwise the application performs very well. When compared to Google Scholar, this project has added many new, and valuable, features to Google Scholar. This alone makes the project worthwhile. Not only does the application interface with Google Scholar, but because of its design it is able to interface with any other academic search engines. The user evaluation of the user interface was as expected since they were involved in the design process.

Chapter 7

Conclusion

7.1 Introduction

This chapter is intended to highlight the overall goal of the project and summarise how successful this project has been. The possibility of future development work will also be discussed.

7.2 Project Achievements

The main objective of this project was to take the results from Google Scholar and display it in alternate ways so that it can be used for analysis. This goal was achieved by creating many new features which were discovered through analysis of existing systems and through background research. Other goals included making the user interface intuitive and simple. This was achieved by getting the users involved in the user interface design, through mock-ups and prototypes.

7.3 Further Work

There are many new enhancements that could be made to this project. This is mainly due to its flexible architecture. One could combine data from many different academic search engines to display a much more complete and detailed description of articles and their authors. As mentioned in evaluation, having author's full name would improve the application. This is something that cannot be done using Google Scholar, since Scholar distinguishes authors by their first initial and last name. The biggest problem with this project has been that it requires Google Scholar. Instead of querying Google Scholar for data, if there was a local copy of the database then many new functionality could be added, that normally would be too complicated. Just by having the database locally we could run simple queries and generate many different views of the data. A number results can be generated for analysis.

7.4 Conclusion

During any development life cycle it is inevitable to face problems. This project was no exception. Many of the problems that occurred were due to the fact that this project required to interface with Google Scholar. This is a typical problem when trying to build applications which interface with other applications, which you do not own or have control over. There were times when Google Scholar would change its web page layout, ever so slightly, but it would end up breaking parts of my API. This did however teach me a very important lesson about designing API's: always make it adaptable. There were other problems such as Google would block your IP¹ if too many queries are made to Google Scholar. There were limitations on how many results you can actually view, this limitation was quickly realised when trying to gather data for a graph. There are some parts of the development cycle that have been very successful. This project was built using an iterative development process. During requirements analysis, mock-ups were built to determine what the user interface would look like and what features should be included in the final system. This process proved to be quite successful. The users were able to easily see what each feature would do, and also see what the application would look like when it would be finished. Many modifications were made from their feedback, even more so on the user interface. When it came to implementing the user interface the mock-ups were directly used with some minor changes. Not only did implementation finish quickly, but we knew it was exactly what the user wanted. After completing this project it was good to know that an application was created which can rival any existing solutions.

¹Internet Protocol

Bibliography

- [goo07] Google technology. Website, 2nd May 2007. <http://www.google.com/technology/>.
- [mas07] Mashup. Website, 2nd May 2007. http://en.wikipedia.org/wiki/Mashup_%28web_application_hybrid%29.
- [mvc07] Model view controller architecture. Website, 2nd May 2007. <http://en.wikipedia.org/wiki/Model-view-controller>.
- [N⁺94] J. Nielsen et al. Usability Engineering. Morgan Kaufmann, 1994.
- [Nie99] J. Nielsen. Designing Web Usability: The Practice of Simplicity. New Riders Publishing Thousand Oaks, CA, USA, 1999.
- [Rub04] Richard E. Rubin. Foundations of Library and Information Science 2nd ed. New York, 2004.
- [scr07] Screen scraping. Website, 2nd May 2007. http://en.wikipedia.org/wiki/Screen_scraping.
- [ucd07] User-centered design principle. Website, 2nd May 2007. http://en.wikipedia.org/wiki/User-centered_design.
- [web07] Web application. Website, 2nd May 2007. http://en.wikipedia.org/wiki/Web_Application.

Appendix A

Mock-up User Evaluation Questionnaire

Mock-up User Evaluation Questionnaire

Overall System

User interface

<i>Please rate how familiar the user interface is to Google Scholar:</i>												
very familiar	1	2	3	4	5	6	7	8	9	10	not familiar	
<i>Please rate the intuitiveness of the user interface:</i>												
very intuitive	1	2	3	4	5	6	7	8	9	10	not intuitive	
<i>Page layout is arranged in a sensible manner:</i>												
strongly agree	1	2	3	4	5	6	7	8	9	10	strongly disagree	

Live search

<i>Rate this feature:</i>												
very useful	1	2	3	4	5	6	7	8	9	10	not useful	
<i>Please rate how intuitive it is to use this feature:</i>												
very intuitive	1	2	3	4	5	6	7	8	9	10	not intuitive	

Breadcrumbs for history of pages visited

<i>Rate this feature:</i>												
very useful	1	2	3	4	5	6	7	8	9	10	not useful	
<i>Please rate how intuitive it is to use this feature:</i>												
very intuitive	1	2	3	4	5	6	7	8	9	10	not intuitive	
<i>The terms used on the interface is:</i>												
easy to understand	1	2	3	4	5	6	7	8	9	10	hard to understand	

Results Page

Author list

<i>Rate this feature:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Please rate how intuitive it is to use this feature:</i>												
very intuitive	1	2	3	4	5	6	7	8	9	10	not intuitive	
<i>Sorting author list is a good feature:</i>												
strongly agree	1	2	3	4	5	6	7	8	9	10	strongly disagree	

Results page sorting

<i>Sort by number of citations:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Sort by article title:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Sort by publication date:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Sort by publisher name:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Display results in reverse of sorting:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Sorting hyper link names:</i>												
easy to understand	1	2	3	4	5	6	7	8	9	10	hard to understand	

Combined multiple article actions

<i>The terms used on the interface is:</i>												
easy to understand	1	2	3	4	5	6	7	8	9	10	hard to understand	
<i>Combined cited by:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Combined related articles:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Combined citations per year graph:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Combined citations per year graph without self citations:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Ability to combine different actions is a useful feature:</i>												
strongly agree	1	2	3	4	5	6	7	8	9	10	strongly disagree	

Statistics

<i>Authors pie chart:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Publisher pie chart:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Publication year pie chart:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	

Articles Page*Summary of article*

<i>Please list of alternate links:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Abstract of article:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Article action button names:</i>												
easy to understand	1	2	3	4	5	6	7	8	9	10	hard to understand	

Top 10 authors who cited article

<i>Rate this feature:</i>												
very useful	1	2	3	4	5	6	7	8	9	10	not useful	
<i>Each author is a hyper link to their page:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	
<i>Number of citations is a hyper link to citations page for the corresponding author:</i>												
good	1	2	3	4	5	6	7	8	9	10	bad	

No. of Authors by No. of Citations graph

<i>Rate this feature:</i>												
very useful	1	2	3	4	5	6	7	8	9	10	not useful	
<i>Rate how easy it is to understand this graph:</i>												
easy to understand	1	2	3	4	5	6	7	8	9	10	hard to understand	

Cited by

<i>Rate this feature:</i>												
very useful	1	2	3	4	5	6	7	8	9	10	not useful	
<i>Cited by without self citations:</i>												
very useful	1	2	3	4	5	6	7	8	9	10	not useful	

Citations per year graph

<i>Rate this feature:</i>												
very useful	1	2	3	4	5	6	7	8	9	10	not useful	
<i>Citations per year graph without self citations:</i>												
very useful	1	2	3	4	5	6	7	8	9	10	not useful	

Authors Page

Articles written by author

Rate this feature:													
very useful	1	2	3	4	5	6	7	8	9	10	not useful		

List of co-authors

Rate this feature:													
very useful	1	2	3	4	5	6	7	8	9	10	not useful		

Each author is a hyper link to their page:													
good	1	2	3	4	5	6	7	8	9	10	bad		

Top 10 co-authors

Rate this feature:													
very useful	1	2	3	4	5	6	7	8	9	10	not useful		

Each author is a hyper link to their page:													
good	1	2	3	4	5	6	7	8	9	10	bad		

Number of article written by author is a hyper link:													
good	1	2	3	4	5	6	7	8	9	10	bad		

Top 10 authors who cited this author

Rate this feature:													
very useful	1	2	3	4	5	6	7	8	9	10	not useful		

Each author is a hyper link to their page:													
good	1	2	3	4	5	6	7	8	9	10	bad		

Number of citations is a hyper link:													
good	1	2	3	4	5	6	7	8	9	10	bad		

Citations per year graph

Rate this feature:													
very useful	1	2	3	4	5	6	7	8	9	10	not useful		

Citations per year graph without self citations:													
very useful	1	2	3	4	5	6	7	8	9	10	not useful		

Publications per year graph

Rate this feature:													
very useful	1	2	3	4	5	6	7	8	9	10	not useful		

Appendix B

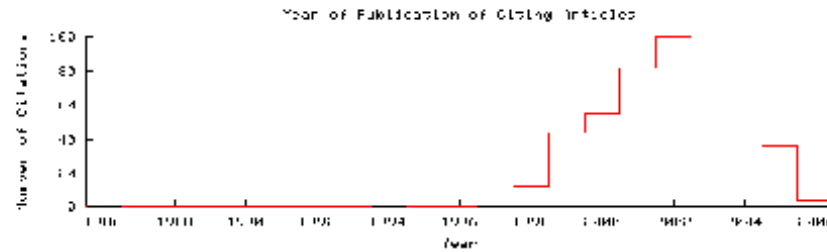
Different Phases of Mock-up Design

Citations per year for article:Sorted by Citations [Reverse Order](#)**[The Google file system](#)**

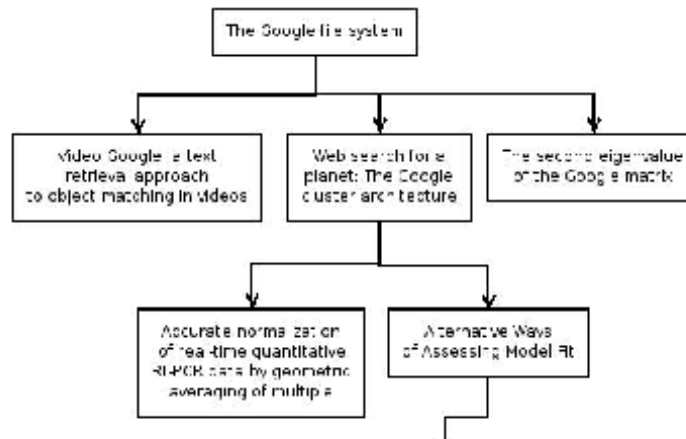
S. Ghemawat, H. Gobioff, E. Threlkeld - Proceedings of the nineteenth ACM symposium on Operating systems - 2003 - peridot.com.org

Page 1: The Google File System Safety Overview - forward Google, and

Shun-Tak Lung Google ARSTRACT We have designed and im

Citations per year:**Cited by tree:**

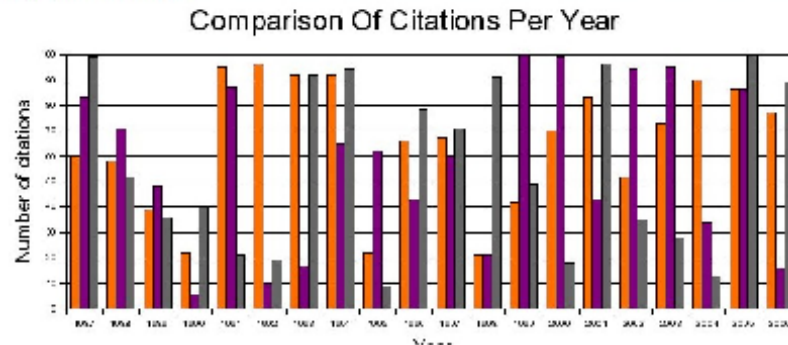
Articles which has been cited and has been cited by are listed in a tree form below:

**Combined citations per year:**

Number of Google Scholar citations per year for articles

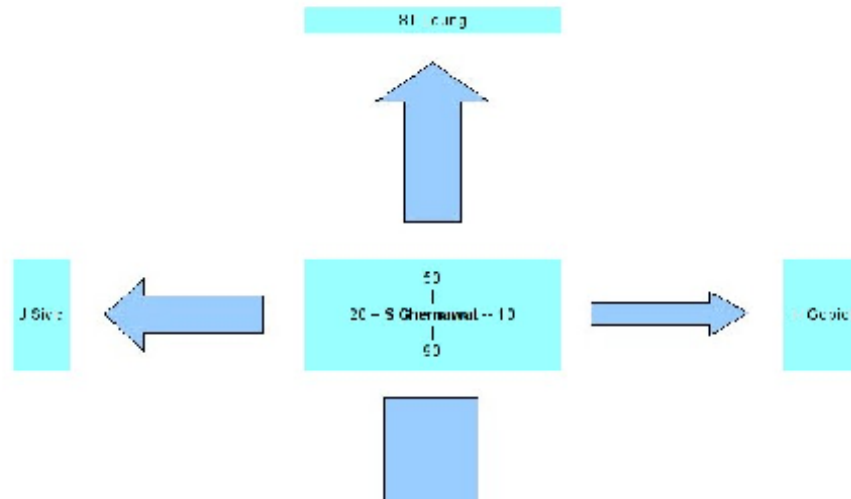
Number of Google Scholar citations per year for articles

Combined citations per year on the above articles:



Top citing authors:No operations on selected: [show citation network](#)

Top citers for author "S Ghemawat":

**Statistics pie chart:**

Proportion of Authors

