UNIVERSITY OF SOUTHAMPTON
Faculty of Engineering, Science and Mathematics
School of Electronics and Computer Science

A project report submitted for the award of
MEng Hons Computer Science with Artificial Intelligence

Supervisor: Dr Nicholas Gibbins
Examiner: Dr Alex Rogers

Project Triton : A study into delivering targeted information to an individual based on implicit and explicit data

by Liam Ranil Fernando

May 7, 2009

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING, SCIENCE AND MATHEMATICS
SCHOOL OF ELECTRONICS AND COMPUTER SCIENCE

A project report submitted for the award of
MEng Hons Computer Science with Artificial Intelligence

by Liam Ranil Fernando

The World Wide Web is frequently seen as a source of knowledge, however much of this remains undiscovered by its users. In recent times, recommender systems (e.g. Digg and Last.fm) have attempted to bridge this gap, alerting users to previously untapped knowledge. As more socially oriented services appear on the Web (e.g. Facebook and MySpace), it has never been easier to obtain information pertaining to an individual's interests. At present, solutions for automated data recommendation tend to be highly topic specific (recommending only a certain topic such as news) and often only allow access to the system using monolithic interfaces. This report hopes to detail the stages from research to evaluation involved in creating an extensible framework, which will operate without the need for human intervention. The framework will feature several proof-of-concept plugins residing in a custom workflow, which target information that is useful to the user. Information will be retrieved automatically through plugins involved with data gathering (such as feed processing and page scraping), while users' interests will be obtained implicitly (for example, using header information to derive location) or explicitly (taking advantage of Social Network APIs such as Facebook Connect). Finally, Third Parties will be able to integrate the framework into their own solutions using the customisable XML API (written in PHP), so that their products can provide custom user interfaces without style constraints.

Contents

Acknowledgements

I would like to thank Dr Nicholas Gibbins and Dr Alex Rogers for their time, input and continued support in Project Triton.

Chapter 1

Introduction

1.1 The Problem

In the market today, vast arrays of solutions exist to deliver news to an individual, from news vendors themselves [1], to themed sites that provide a plethora of information on a single subject [2]. However, few of these news delivery systems tailor the user's experience to find news relative to his or her known and likely interests. Furthermore, those that do are not extendable or implementable on a wide scale. As people like to read relevant news but seldom wish to search for it, they often stick to one news site, or perhaps have a large collection of sites that they check. Normally this results in the use of portals to organise syndication feeds from sites into something manageable. The trained solutions that currently exist are generally quite old and do not take advantage of the recent social revolution. No custom solutions provide an XML interface that continues to train the user's profile, further tailoring the delivered news.

1.2 The Goals

The goal of this project is to create a solution for delivering targeted news to an individual based on information gained using explicit or implicit methods. The system should consist of a framework on which plug-ins can be installed, customising the input, processing, and user display of news. The implementation should be such that each installation is fully customisable to both the user's needs and server's demands. The project is hoped to ship with several example plug-ins, which will be used as the basis for further development, while being part of a fully usable product, which will have a prominent role in the market. The project will essentially be separated into three distinct segments; the input phase, the process phase, and the output phase. The input phase will collate different input sources; these could be standardised sources such as XML feeds, or plugins could be developed to process un-standardised inputs using page-scraping techniques for news or other media (such as images). The processing phase will essentially involve plug-ins that process input or user data and return the targeted content in a correctly encapsulated form. The relationship data is also obtained during the processing phase. A typical plug-in may for example, tag articles by genre, or recognise different articles as the same story associating the source with a particular political stance (e.g. a news story from the Times, may be seen as Conservative). After the process phase, there will exist an XML interface written in PHP that delivers a range of outputs about the system based on given arguments. The XML interface should update the system appropriately, altering a given user's interest vector based on the news they request. The output phase will involve plug-ins that deliver the content of the system to different platforms, such as a web-front end, communicating with the system using the XML interface. While ideally it would be preferable to create wide range of plug-ins for the system, the focus of the project will be entirely on making a complete framework and the basis of a strong community based development. This will not only ensure that the plug-ins are varied, but push the development of the solution in the future.

Essentially Project Triton is intended to be a functional framework for news aggregation recommending articles based on the users' interests. Furthermore, the solution will consist of plug-ins to customise the experience to both the users' needs and the demands of the server.

Chapter 2

Literature Review

Shortly after work commenced on the project, it became apparent that there were several key areas where research would be essential. Initially these topics were divided between two main categories, those involving the development of a sustainable framework; and those involving the tagging and categorisation of stories. However as I researched more, the number of categories increased. Using academic paper search systems such as IEEExplore and Central Search, I was able to find papers that assisted my design process. The following sections provide brief explanations of the papers I found particularly useful or applicable to my system.

2.1 Background Reading

2.1.1 Guidance versus Automaton

When designing any system that relies on artificial intelligence, a topic of heated debate is the choice between an autonomous approach that trains itself and a trained system that learns through guidance. Many people have studied the comparison between the two approaches. It is often thought that trained systems offer a higher level of accuracy than autonomous systems, because of the ability to correct invalid knowledge in the system. Training a system is an intensive process, requiring an expert to guide the system until it is self-sufficient. It is not always the case that an expert will be available or willing to train a system. This is especially true in commerce, where users will generally prefer self-sufficient products.

2.1.2 Approaches to Classification

Through my research I have discovered that there are many different automated methods for text classification. Fabrizio Sebastiani [3] provides a comprehensive review of most of these techniques. From this paper, I have come to realise that there exist fuzzy classification areas, in which a corpus may not appropriately fit under any one single class, requiring the use of multilabel text categorisation systems and the use of fuzzy logic. I also learnt that the perspective of document classification largely determines how effective a classification algorithm will be. By using document-pivoted text classification (DPC) methods instead of category pivoted categorization (CPC), it is likely to be more efficient at filtering documents under a rapidly changing environment such as news aggregation. It is important to note that most methods of text categorisation will work in both CPC and DPC. The paper went on to describe many methods of text categorisation, comparing and contrasting the differences between them. It became apparent however, that with increased accuracy came decreased efficiency, with the most efficient approaches often using vast amounts of document simplification techniques before processing. Removing context sensitive words, which have different meanings in given scenarios, averted further problems. The use of guided topic dictionaries (such as MeSH) to help provide topics for a corpus also improved accuracy, but these are normally only available for restricted domain sets, which may cause issues in a rapidly changing environment such as news. This paper led me to believe that perfect text classification is extremely difficult to achieve; I have therefore chosen to use an automated supervised system. This will allow an 'editor' to correct any wrong classifications, improving the accuracy of the system, whilst leaving the system easy to maintain.

Topic classification is notoriously difficult to perform accurately, as a result it is a problem that has been studied throughout computer science history. Several papers analysed text classification in a more generic sense (such as 'Algorithmic Detection of Semantic Similarity' [4]), however some papers specifically looked at news articles and speech. Project Triton's main objective is to deliver personalised news, so I have decided to classify all articles in the system. When classifying an article it is important to realise that not all articles are written in the same style. This leads to complications in using traditional classification algorithms, which typically rely on the use of similar words and the frequencies of these words to detect similarity. The University of Ballarat produced a paper titled 'Using Corpus Analysis to Inform Research into Opinion Detection in Blogs' [5] which looked at detecting writing styles across a range of online media. The study showed that typically, opinionated articles contained more unique words. These were often slang terms. While the article did not extend the study to compare different types of blogs, it did hint that different writing styles contain different types of words, and changing proportions of nouns, verbs and adjectives. The use of dictionaries for different writing styles may be useful in article categorisation. However, this has its drawbacks as the system would require training data, and as a result would be only as good as the data that trained it. A further notable article was 'Text Categorization using Feature Projections' [6] which suggested a new method (TCFP) of categorizing documents, comparing it to commonly used alternatives.

2.1.3 Detection of Semantic and Stylistic Similarity

A large issue when dealing with multiple data sources is removing redundancy. This is especially difficult when the sources contain different writing styles. In my system, I want to group articles about the same story together, offering different writing styles based on the users preference. Ian Garcia and Yiu-Kai Ng's paper; 'Eliminating Redundant and Less-Informative RSS News Articles Based on Word Similarity and a Fuzzy Equivalence Relation' [7] compared several techniques such as word frequency (which classify documents based on the frequency of the words they contain), and keyword distance and connectivity (classifying documents by the distance between keywords) to determine how best to detect similar news stories. An interesting method hinted at by the paper was the use of meta data in detecting similarity, for example, if two items have similar pubDate RSS tags, it is likely they are about the same story. The research in this paper is particularly relevant to Project Triton as the study specifically looks at RSS news feeds (a major input of Triton).

2.1.4 Name Extraction

The approaches to classification described in Section 2.1.2 are often quite accurate. This accuracy is further improved by eliminating stop-words (inconsequential words) from the data. However, when operating on large data sets, the results they produce often show decreasing accuracy. It is for this reason, that alternative methods of classification have been researched.

Places, Identities and Events typically have uniquely identifiable names. By analysing news articles it has been noticed that most stories are concerned with one such item. Indeed, the notion of classification by name extends far beyond news data, and can be applied to many different sources such as Images and Videos. As data typically includes fewer proper nouns than other grammar, this reduces the subset on which to operate upon, thus increasing accuracy. By calculating word frequencies, or distances using only the proper nouns, both the intensity requirements and accuracy of the algorithm are improved.

The extraction of proper nouns from corpora has been studied by many linguistic academics. Thierry Poibeau and Leila Kosseim noted that much of this research has used formal documents such as newspapers as analysis material [8]. Although this is unlikely to cause an issue for Project Triton, it is important to contemplate the application of name extraction on less formal documents. Thierry Poibeau and Leila Kosseim, described alternative techniques for the extraction of ENAMEX (proper nouns), TIMEX (temporal logic) and NUMEX (numerical logic) from e-mails and oral transcripts. This required a different approach as traditional extraction techniques relied on proceeding identifiers to names such as Mr or Mrs. 'Proper Name Extraction from Non-Journalisitic Texts' [8], details the short-comings of grammar-based analysis and need for a learning machine - highlighted by the poor performance of Lexis. The report also showed how the use of discourse structures can improve accuracy. Discourse Structures identify names by a set pattern of words. When applied with a Learning Machine, it is possible to learn the discourse structures, increasing accuracy. However, problems occur when a word appears both in the set of learnt and programmed rules, as this creates a conflict. The article details several reasons why such a conflict might arise, and stresses that a perfect system is elusive, despite the techniques described in the paper - which typically showed a 34% increase in accuracy.

'Intelligent Content Based Title and Author Name Extraction from Formatted Documents' [9] uses a variety of novel techniques to extract the Title and Author names from published articles. The paper avoids using stylistic based detection methods, and focuses on content and format based approaches. The content-based methods first re-format any articles into a more amenable form, before applying a variety of regular expressions and dictionary tests to decide the likelihood that the said data was either a Title or Author Name. The Format-based approaches described, used the internal file structure of reports to recognise the target elements.

While novel, much of the research seemed naive when applied to the requirements of this project. In practice only the regular expressions hold any true value when concerned with Project Triton.

'Person name extraction from Turkish financial news text using local grammar-based approach' [10] uses a technique known as local grammar to extract proper nouns. Although aimed at the Turkish language, the paper highlights the difficulties in obtaining local grammar, particularly with a complex language such as Turkish. The paper proved that the use of local grammar is effective once obtained, and if Project Triton were ever to process foreign languages, local grammar may provide a method of name extraction.

2.1.5 Truth Detection and Information Fusion

'Information extraction based on information fusion from multiple news sources from the web' [11] looks at combining extracted information on a single topic or entity over multiple sources. The paper entertains the notion that some sources are more credible than others. By use of a Multi-Layered Perceptron called mFuse, the authors compare data about an entity, accounting for source credibility before outputting the most truthful data. This could be useful in Project Triton, as it would allow the creation of composite news stories, or an encyclopedia of knowledge about an individual. Furthermore, by associating truthful facts with an individual, Project Triton would be able to recommend more accurate data. There is however, a downside, designing a MLP such as mFuse in PHP would be computationally expensive. It remains that despite the increase of accuracy experienced when using mFuse, the research is highly experimental and prone to failure.

2.1.6 Monge Elkan Distance and the Field Matching Problem

Field matching is the process of determining similarity of syntactically different, but identical entities. In 'The field matching problem: Algorithms and applications' [12], Monge and Elkan discuss a basic field matching algorithm, which when applied recursively can detect identical entities with differences such as complex abbreviations. The basic matching algorithm is as follows:

$$match(A, B) = \frac{1}{|A|}\sum_{i=1}^{|A|} \max_{j=1}^{|A|} match(A_i, B_j) \tag{2.1}$$

After describing the functionality of the basic matching approach, the paper goes on to compare the Smith-Waterman algorithm (also known as the Monge Elkan distance). Smith-Waterman uses a weighted matrix of alphabets, and penalties for starting and continuing gaps in digests to increase accuracy. As Smith-Waterman allows for missing characters, it should perform better than basic matching.

2.1.7 Server-Side Performance PHP

A key consideration in the development of Project Triton is the choice of development language. Performance benchmarking of computers is a common affair. As a result, many papers have been published comparing the benefits and disadvantages of different server-side languages.

In the market today, two of the major front-runners are PHP and JSP. 'Performance comparison of PHP and JSP as server-side scripting languages' [13] compared the performance of PHP and JSP as server-side scripting languages using SPECweb2005. The study showed that while Java (JSP) greatly out performed PHP when executing native code, PHP beat Java when using built-in functions. Throughout the report, Java generally performed better than PHP; this is likely because of Java's Just In Time compiler. It should be noted however, that although Java significantly beat PHP in some tests, PHP showed particular performance advantages in I/O processing and algorithms such as MD5 and Levenstein.

'A Performance Comparison of Dynamic Web Technologies' [14] benchmarked PHP, Java and Perl comparing their ability to process dynamic content. The paper concluded that Java generally out-performed PHP and Perl. Interestingly, the paper suggested that PHP's failings were due to apache (a server platform with 60% of the market share). Furthermore, it stated that PHP showed competitive performance levels on small dynamic content and was robust when under heavy load.

Finally, I looked at 'Some Experiments with the Performance of LAMP Architecture' [15]. The report compared the performance of PHP code against C code. Unsurprisingly, the C code out performed PHP. Interestingly, some results, such as the MySQL database access test were almost identical when performed in PHP and C. The report also compared different implementations of AMP (Apache, MySQL and PHP) over multiple operating systems, concluding that a Linux based solution was fastest. The report also found that WAMP was significantly faster than IIS when performance benchmarking under Microsoft Windows. I can conclude that Java is the fastest server-side language, followed by PHP, then ASP. Despite this, a major goal of Project Triton is to create a system which is highly portable. As Apache and PHP control the market, it is vital the project be developed using these technologies so that the project is available to the widest audience.

2.1.8 Oliver in PHP

'Decision Graphs - An Extension of Decision Trees' [16] forms the basis of a built-in PHP function called similar_text(). The report starts by creating an inefficient classifier which divides objects into uniquely defining attributes, creating easily distinguishable classes. This classification uses decision trees, which have fragmentation problems and biases when using large data sets. While the report recommends solutions to these problem, it contrasts the use of decision graphs for a more efficient implementation. The creation of decision graphs requires one first to grow a decision graph. Previous studies involved in growing decision graphs fixed the shape of this graph to suit a defined structure, partitioning the data when it did not conform to this. By partitioning the data, meaning about the domain was lost, creating a dis-advantage. The report compares preexisting grow and prune methods with their own suggested iterative growing technique, demonstrating their method's ability to create high-purity, low-noise decision graphs. PHP utilises this research to create an efficient implementation of string similarity. Although the algorithm complexity is O(n^3), it compares favourably with more efficient non-built-in algorithms due to it's C implementation.

2.1.9 Applications of Historical Learning Techniques

'Behavior Based Web Page Evaluation' [17] and 'A Web Usage Mining Framework for Mining Evolving User Profiles in Dynamic Web Sites' [18] both show the importance of logging in detecting user behaviour. 'Behavior Based Web page

Evaluation' [17] has particular relevance to Project Triton as analyses AJAX based techniques for user feedback - which will be used in the Output Phase. While the methods described in these papers are not new or innovative, they provide examples of how useful it is to log every aspect of a user's activities and confirm that my proposed methods of behavioural tracking (using AJAX and header information) are creditable.

2.1.10 Location Detection by IP Address

'A Location Information Retrieval System using IP Address' [19] aims to derive a geographical location for any given IP address for use in targeted advertising. The report recognises that there exist two types of IP address; static and dynamic. Static IP addresses do not change regularly, and therefore can be cached for long periods of time. Conversely, dynamic IP addresses change rapidly, and are required to be 'lively' calculated. The article states that locationary information is stored in an IP address as a 8 byte local code, this can be cross referenced to a look-up table denoting geographical location. To increase efficiency, and reduce load on the system, it is recommended that blocks of IP address space be cached for short periods of time on the server, this will negate the need to look up a location every time a user visits the page. The paper proceeds to look at optimisations such as early warning techniques (flagging) to detect if an IP address is not in the geographical look-up, before comparing the investigated techniques. It is noted that peak throughput was achieved when caching and flagging was enabled. The work of this and other similar papers has spawned the creation of several libraries for PHP involved with geo-location, and will be particularly revelant if integrating location information into Project Triton.

2.1.11 Representation of XML Workflow

A core focus of Project Triton is to be flexible, I quickly realised that a predefined structure would need to be created for plugins to be designed upon. I soon decided that a dynamic workflow was required, so that the WebMaster could alter the execution of plugins with ease. I found two papers comparing approaches for dynamic workflows. While both papers discussed alternative techniques (using configuration files and object oriented programming), it was clear that using an XML based standard would be easy to configure and implement given PHP's ability to process XML. Both papers suggested XML solutions to solve the problem;

however they were often too complex for the existing problem. Instead this led me to create my own XML schema that I could implement offering only the services I required, discussed in 'The Design and Implementation of XML-based Workflow Engine' [20]. This meant that my system would remain efficient while still offering a dynamic workflow.

2.1.12 Persistent Data Connection Techniques with PHP and AJAX

In the modern age, the need for up-to-date information is key. In the production of Project Triton, an example user interface will be created to interact with the API. It is likely that certain elements of this website will interact with Third Parties (such as Social Network APIs). In situations such as that, it might be preferably to provide instant feedback on the progress of the script. While the research of such techniques is quite young, many solutions exist to provide this functionality. 'A comparison of push and pull techniques for Ajax' [21] details a number of methods used to transfer data persistently between a server and client. Bozdag, Mesbah and Deursen explain the concept of an Ajax XMLHttpRequest to make use of REST-ful services, before explaining how persistence can be added through the inclusion of a Time to Refresh (forcing the page to re-load every time period). Instead of using REST-ful services, an alternative method might be to create an HTTP Stream. In PHP, this is achieved by creating an infinite while loop, inside which flush() commands are called. When combined with an Ajax XMLHttpRequest, this streams data to the page.

Sometimes, a web page may need to respond to a certain event invoked on the server. For this reason, a technique known as Reverse AJAX or Comet was introduced. Comet enables the server to send a message to the client when an event occurs, without the client having to send a request explicitly. The paper proceeded to analyse the performance of these methods. It should be noted that all the described techniques scale well, although the paper hastens to point out that naive implementations would be likely to negatively impact on performance. To aid the development of such services, 'Comet and Reverse Ajax: The Next Generation Ajax 2.0: The Next Generation Ajax 2.0' [22] suggests several practical techniques such as intelligently altering the Time to Refresh of an HTTP Pull, based on the rate of previously received new data. Further-more a framework called the Ajax Push Engine (APE) implements many of these design practices, forming a robust JavaScript Ajax library.

2.1.13 WordNet

WordNet is a collection of words which act like a thesaurus. 'The Design and Implementation of an Electronic Lexical Knowledge Base' [23] looks at the failings of automated thesauruses, and compares the success of WordNet - a manually constructed thesaurus - with an electronic version of a traditional thesaurus (Roget's). Through quantitive testing, it was shown that the Roget's Thesaurus provides a wider range of words than WordNet. This may be explained by the number of years that Roget's have been entering words into their thesaurus. Despite this initial victory, other quantitive measures showed that WordNet far exceed the implementation of Roget's. The report concludes, that while they remain very different in their internal structure, both thesauruses are very useful in natural language processing. If compiled correctly, the Roget's Thesaurus would probably exceed the performance of WordNet. However, creating an electronic representation of Roget's is very difficult, and the authors hasten to add that several improvements would be required in the future. For that reason, WordNet is the obvious choice for NLP in Project Triton.

2.1.14 RSS XML

RSS has been prevalent in the World Wide Web for many years now. It is an XML variant that provides a lightweight means of content distribution, relieving load on the server. It also provides the opportunity for Third Party developers to integrate feeds from external sources into their own projects. In his article 'RSS' [24], Tom Barnes describes how RSS can also be used for pushing email to a client, and driving many other of a typical website's services. The paper then goes on to point to future implementations of RSS, chiefly Media RSS (now specified by Yahoo). Media RSS adds a name space to the RSS specification allowing for the insertion of data regarding videos, images and other embedded content. Integrating feeds such as RSS and MRSS into Project Triton will enable the system to stay up to date easily, without burdening the server.

2.2 Previous Work

While I have not been able to find any existing solutions that provide the exact functionality of my proposed system, some websites provide services that attempt to bridge the gap between what the user wants and standard news delivery systems. These services can generally be divided into the following categories;

2.2.1 Tagged News (DayLife)

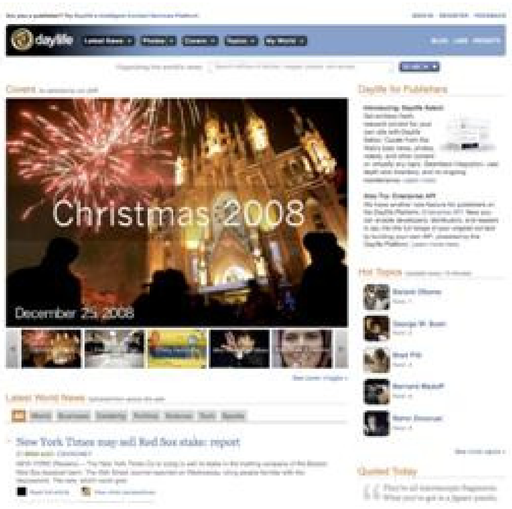


FIGURE 2.1: DayLife

Tagged News websites use tags to group news stories under similar topics. For example, using DayLife it is possible to find out about news stories regarding 'Barack Obama'. DayLife is an example of an automated tagging system. However, it only uses one news source, avoiding the possibility of duplicated news. Automated tagged news sites are rare, and DayLife is one of the few successful implementations on the market today. One of DayLife's key successes has been its clean layout, using large images to replicate traditional magazine covers drawing the user to particularly popular stories or information. Another aspect of DayLife which has proved extremely popular is the inclusion of an API (DayPI) which provides a PHP driven XML interface on which developers can request news on specific topics, along with images and extra information. Sadly, this is the limit of developer support DayLife provides, as DayLife remains a closed source project. The system also suffers from a poor implementation of tagging, often returning no results when using common terms. This in part could be traced to a poor database of news, although it is likely that poor implementation of tagging is to blame.

2.2.2 Social || Median

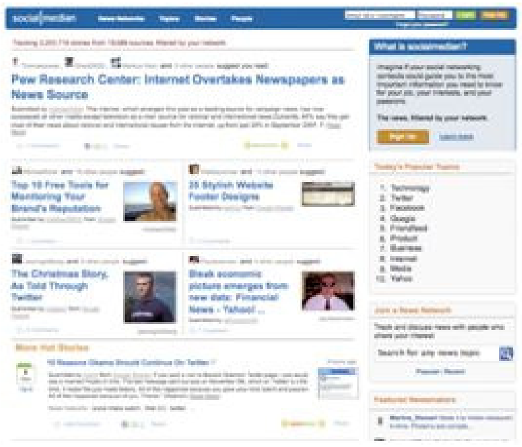

social|median
Pew Research Center: Internet Overtakes Newspapers as News Source
Top 10 Free Tools for Monitoring Your Brand's Reputation
25 Stylish Website Footer Designs
The Christmas Story, As Told Through Twitter
Break economic picture emerges from new data: Financial News - Yahoo! ...
More Hot Stories
What is socialmedian?
Today's Popular Topics
Join a News Network
Featured Newsmakers

FIGURE 2.2: Social || Median

Social || Median uses a collaborative approach to news recommendation to provide users with popular stories. The idea behind this approach is that if many people read an article, it must be interesting to the user. However, this is not always the case as each individual has different tastes. To achieve increased accuracy levels, it is essential that many users interact with the service (diluting the more diverse interests). This approach does however have some advantages: as the system only cares about other users' interests, new users are not required to divulge any information about their interests or social networks they belong to; and the system does while still delivering a tailored approach to news. Ultimately, this means that the accuracy of Social || Median is highly dependent on the number of users who use the system.

2.2.3 Personalised News (MySun, MyTelegraph, etc.)

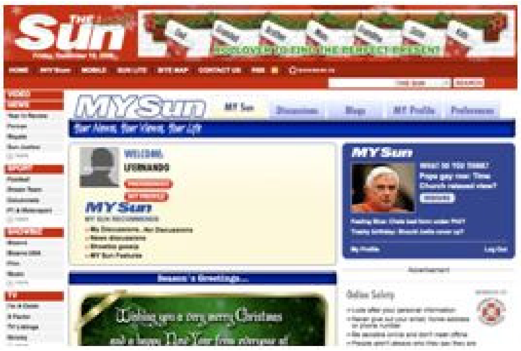


FIGURE 2.3: MySun

In the last few years, large news organisations have decided to offer personalised versions of their websites. These typically ask you to specify some interests from a discrete list, and then deliver news that falls under these categories in the future. More advanced systems might use the user's history and combine this data with of other users to generate more accurate recommendations. Typically, these websites simply act as filters attached to their existing site, and generally will only be providing news from one news source. While the user may receive information that he finds interesting, this means that all the news stories will be one-sided and the depth of news also be poor.

2.2.4 Competitor Comparison

Clearly, some of the mentioned solutions are very successful, generating large amounts of traffic on a regular basis. However, all the systems mentioned above suffer from one major problem, a lack of knowledge in the system. These systems rely on collaborative filtering approaches to recommending news, and those which tailor to your interests tend to use harsh methods of interest elicitation - often providing a list of categories from which to select interests.

	Sources	API	Recommendation	Tagging	Accuracy	Social	Extensible	Reusable	User Setup	Maintenance
DayLife	1	Y	-	Y	Low	N	N	N	-	?
Social || Median	∞	N	Collab	N	Low / Medium	Y	N	N	-	User
MySun	1	N	History	N	Medium	N	N	N	Manual	?
Project Triton	∞	Y	Social Behavioural	Y	?	Y	Y	Y	Auto / Man	Optional

Table 2.1: Comparison of Competitor Services

Chapter 3

Requirements Analysis

Before I began the design phase of Project Triton, it was important to realise the goals of my system and consequently what is required of my project to achieve them.

3.1 Goals

The core functionality required of Project Triton is:

Table 3.1: Core Goals

G1	To deliver targeted news to an individual based on the users interests.
G2	To create a sustainable and flexible framework for news delivery.
G3	To create an API which other applications can use to integrate with Project Triton.
G4	To create a catalogued encyclopaedia of news.
G5	To detect different news topics and / or author styles.

Further functionality could be added to the system by:

Table 3.2: Further Goals

G6	Delivering targeted adverts to a user based on their interests.
G7	Generating demographic information based on the interested users.
G8	Learning a user's interests based on their behaviour.
G9	Running scripts after the process stage (perhaps to email the latest news to a user).

3.2 Requirements

The functional requirements of Project Triton are:

Table 3.3: Functional Requirements

F1	G5	The system must be able to detect different styles and topics in news articles.
F2	G4	The system must be able to classify news into given topics and interests.
F3	G1	The system must be able to provide news to given interests.
F4		The system must be able to retrieve news articles autonomously.
F5	G2	The system must be able to process news in a variety of different ways depending on its configuration.
F6	G1	The system must be able to retrieve a user's interests through explicit means (using social networks such as Facebook).
F7	G1 G8	The system must be able to retrieve a user's interests through implicit means (using header information and or browsing habits).

Project Triton must provide the following non-functional requirements:

Table 3.4: Non-Functional Requirements

N1	Deliver news in a timely manner.
N2	Be able to be implemented on a wide range of servers.
N3	Be an extensible solution to the problem.
N4	Be easily maintainable.
N5	Be quick to install.
N6	Be easy to develop.

Chapter 4

Project Planning

4.1 Risk Analysis

Before begining the development of Project Triton, it was important to identify weaknesses in the development plan. This would ensure that contingency be in place should something adversely affect the project. During this analysis I identified the main causes of project failure, and planned potential solutions to these problems.

4.1.1 Loss of Hardware

During long projects, some loss of hardware capabilities are to be expected. Often developers will have backup machines to continue development on should this scenario occur. For this to be feasable, work needs to be mirrored to a remote location. This will help ensure that a copy of the project is available in the event of a disastor. In Project Triton, the system is served from a Mac Mini, the data from this computer is mirrored to another computer 100 miles away when uploaded to the SVN directory.

4.1.2 Corruption of Data

Corruption of Data often occurs quite frequently in all software projects. Typically, a developer would use a source control system such as SVN to backup all source files in the project. In Project Triton, an SVN server is located on the Mac Mini server, this SVN repository is then automatically backed up onto another computer in a remote location, adding a further layer of security. Should any data become corrupt, it would be simple to revert to a previous version of the source using SVN.

4.1.3 Unrealistic Project Scope

The setting of an unrealistic goal can be disasterous for a project. Keeping a realistic idea of where the project is at all times helps negate this issue. Through the frequent setting of realistic goals, it is possible to remove this element of danager. Should any of these goals not be met, it would be easy to adapt the project to suit the new scenario by restricting the goals of the project.

4.1.4 Work Overload

As Project Triton is subject to a limited development schedule, it is vital not to allow work to build up. While certain amounts of slippage are acceptable, when the end-goal no longer becomes attainable it is vital to restrict the projects goals accordingly. Through the close following of weekly targets, it is hoped that Project Triton remain true to the Gantt Chart, without the occurance of work overload.

4.1.5 Poor Health

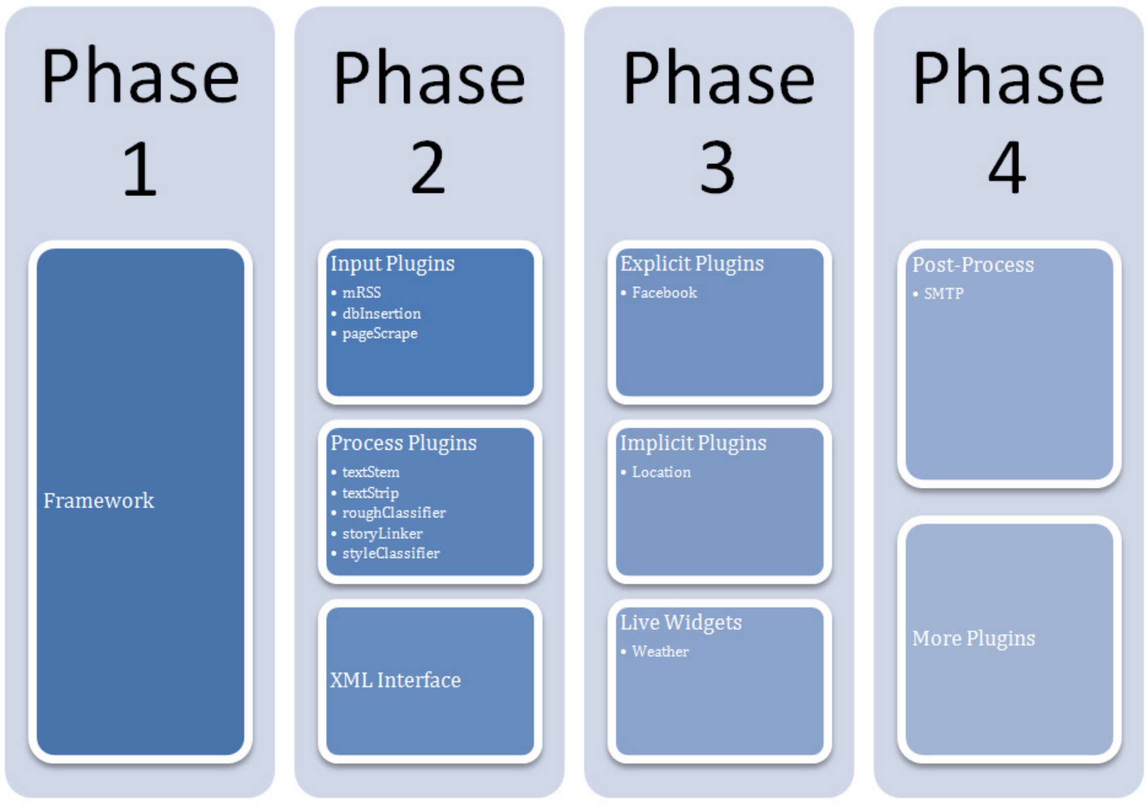
Poor health can strike at any time. Typically when ill, time is lost due to an inability to work. With a time limited project such as Project Triton, it is vital that any lost time be caught up, through the adding of additional work hours. It is important to remain flexible with the scheduling of the project incase the time deficite is too great.

4.1.6 Miscellanious Misfortunes

It is impossible to plan for all scenarios of missfortune. For this reason, the following details typical approaches to ensure the project stays on track. Good practice of source control and data mirroring should negate most mishaps due to corrupt of hardware or software. In addition to this, the settting of realistic short term goals, leading towards an attainable target will ensure that the project never get out of control. It is vital that the Gantt chart accurately portrays the progress of the project. Should any slippage in predicted progress occur, then an increased effort is required to return to normality. If this does not happen, then the scope of the project must be altered.

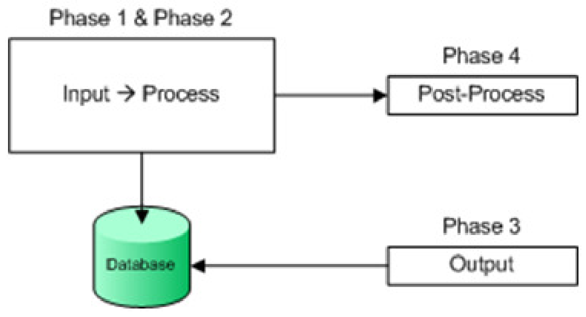
4.2 Project Segmentation

Figure 4.1: A phased overview of Project Triton



After creating a preliminary design, it became clear that the proposed system would have to be constructed in clear phases. A phased approach to development would help ensure that the project be completed in the available time, while provide strong foundations and exemplar material to continue development after the project is complete. I decided to divide the project into four distinct phases, with phases one and two being developed in parallel, while the subsequent phases would only be completed if possible in the time-scale. This phased approach links in with the projects goals, with phases 1 and 2 correlating to the core functionality of Project Triton, while the subsequent phases are covered by further functionality.

FIGURE 4.2: A diagram showing the flow of development between phases



To ensure the correct operation of Phase 1, it was important to develop it at the same time as Phase 2. This ensures that the Framework fully supports the developed plugins and that any conflicts are resolved immediately.

4.3 Initial Project Planning

While I decided on a phased approach early on, there existed a lengthy set of pre-requisites to be completed before I could even begin the project. As I wanted to be able to develop the project from remote locations, the ability to SSH into the system was essential, for this I setup a Dynamic DNS and SSH server on my chosen machine. Another vital feature was the inclusion of source control; this ensured that I could roll back any changes with adverse affects on the system while at the same time creating a log of changes to the project. Finally, I wanted the ability to track and plan the project. For this I created two websites; one for project management, allowing me to plan tasks and make Gantt charts, and another to provide information about the project to the public and link to the test-bed.

Beyond the pre-requisites, I planned the project to conform to a traditional waterfall development model. As a modular framework, the integration between plugins and the framework was crucial. To ensure that each plugin was optimally integrated with the framework, I chose to use an iterative design cycle at a module level, allowing plugins to progress quickly. Before any coding could begin, a lengthy design and research process was necessary. By thoroughly investigating much of Project Triton, it meant that any changes required in the future could quickly and easily be incorporated into the designs with little extra research.

Figure 4.3: Gantt Chart - Part 1

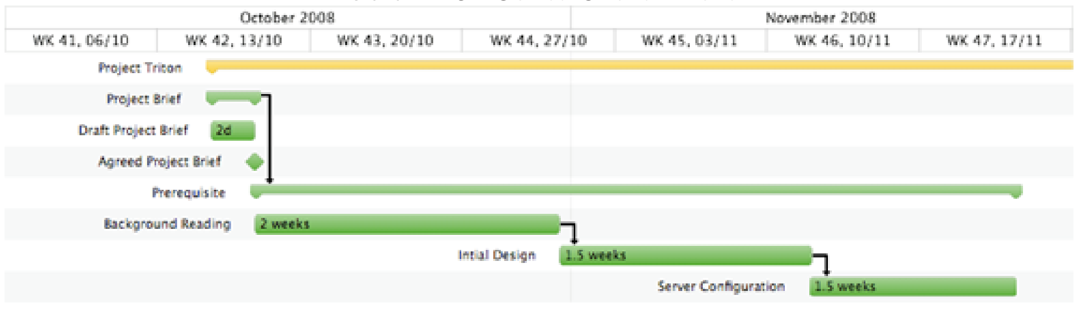
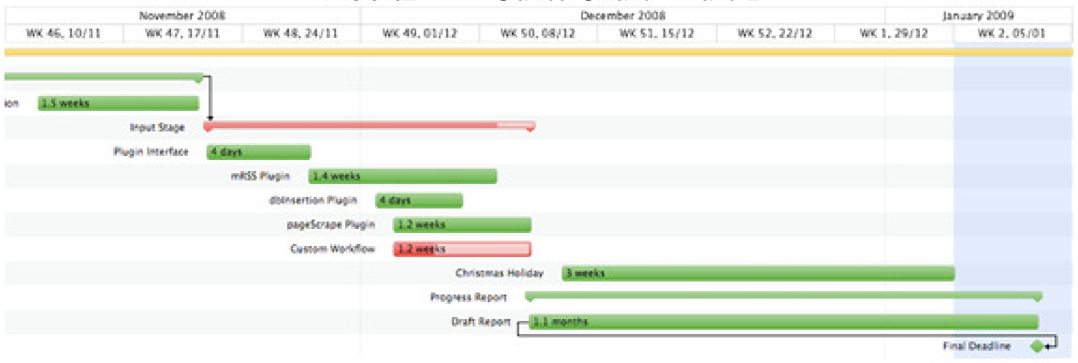


Figure 4.4: Gantt Chart - Part 2



As the plugins would be used very frequently it was vital that time was spent researching their implementation. This helped ensure they remain as efficient as possible.

The development of Project Triton focused primarily on completing Phases 1 and 2, principally the Process Plugins and XML Interface. On reaching this stage the system was able to classify articles. From here Phase 3 began and Project Triton started to provide relevant news to the users.

FIGURE 4.5: Gantt Chart - Part 3

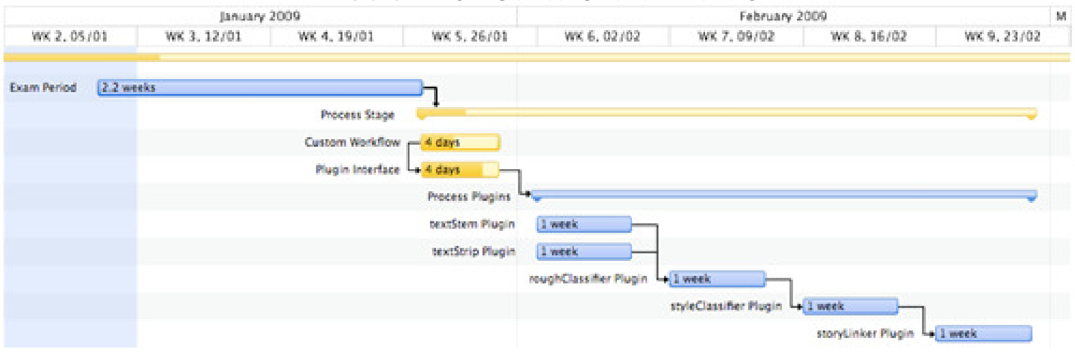
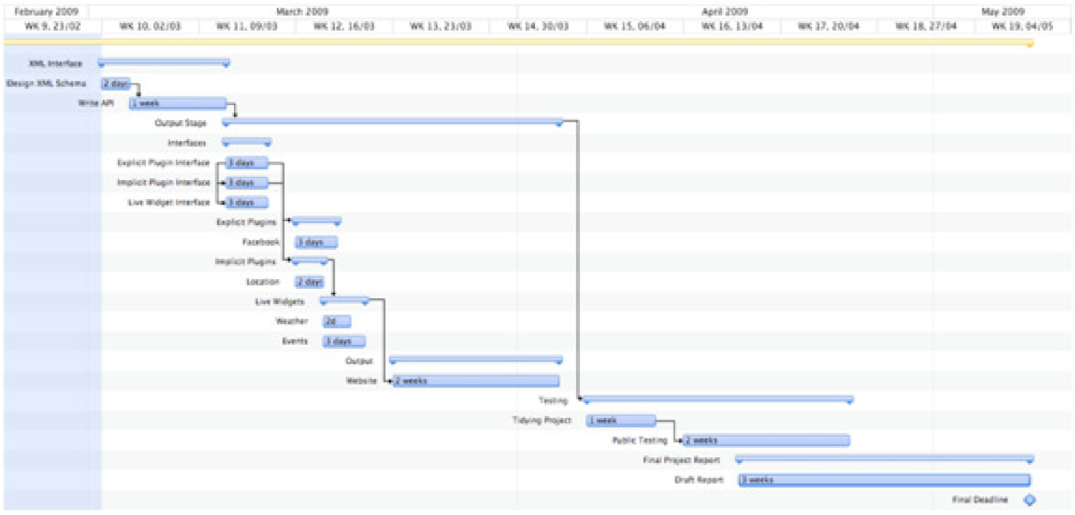


FIGURE 4.6: Gantt Chart - Part 4



Although it would be nice if all four phases of the project were completed in the given time-scale, this was not the target of Project Triton. The objective was that only beyond Phase 2 be completed.

Chapter 5

Design

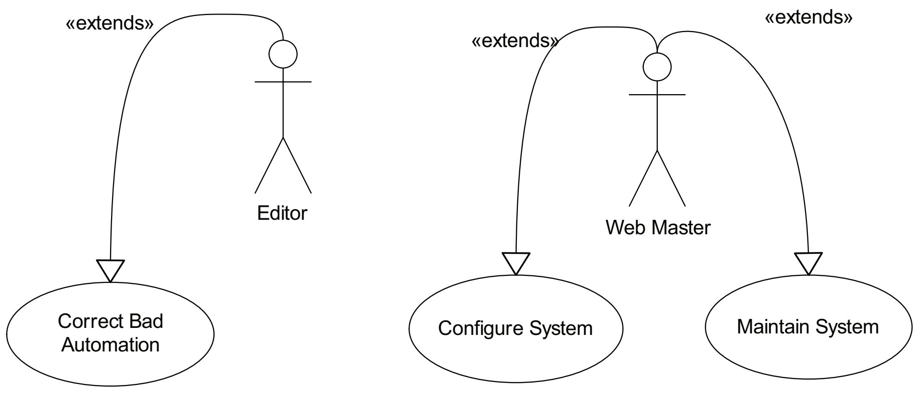
5.1 High level design

5.1.1 Use-Case Analysis

Before designing the system, it was important to recognise what was required of the project. To better understand this, I created a use-case diagram, to determine exactly what all the actors of the system needed.

There exist five possible actors to the system. These actors can be divided into three groups; maintainers; developers; and users.

Figure 5.1: Administrative Use-Case Diagram



The maintainers are responsible for setting up and monitoring the system, and correcting any errors in automation (such as incorrect classification). While an editor may not be required in an ideal system, it is advisable to have one in the proposed system, as the accuracy of classification is highly dependent on the quality of the sources. The Web Master is required to install the system initially, update any settings that should change the server configuration as required, adding and disabling sources and user accounts. Developers might use the system to design applications that require personalised news; this can be done using the system's XML interface.

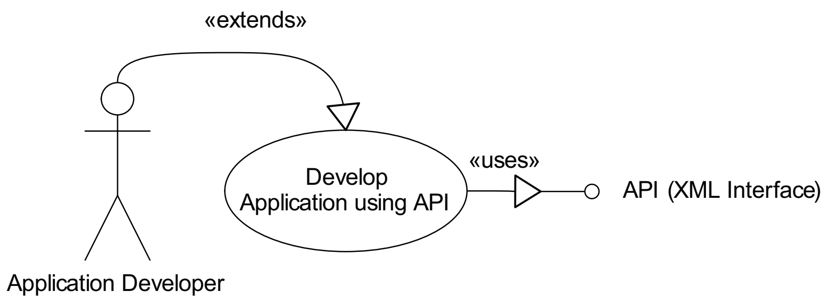
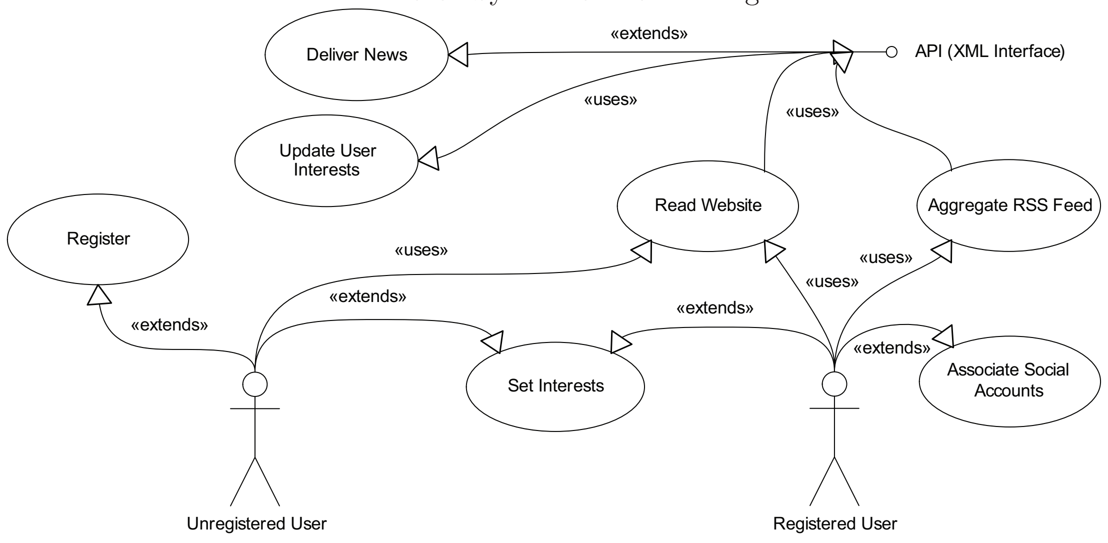


Figure 5.2: Developer Use-Case Diagram

The users will be divided into two categories. Un-registered users will be required to set a brief set of interests by the system so a more accurate set of interests can be acquired. in addition to this implicit data will be used to infer details such as location and ISP about the user. Un-registered users are only allowed to read news through the website, although this will still tailor information based on the users' behaviour. An un-registered user is allowed to register with the system; at this point he or she becomes a registered user. Instead of setting interests manually, registered users might prefer to associate a social network account with their login. By associating a social network account (such as Facebook), the system will extract interest details about the user and deliver news based on his or her interests listed in Facebook. As well as using the website, registered users are also allowed to use the XML Interface in RSS mode, which will deliver the tailored news to them in syndication format.

In the use-case diagram, the inner workings of the system have been disregarded. The inner workings of Project Triton (diagram below) contain three main elements to the project; Input, Process and Outputs, all three of which contain plugins to execute discrete operations. The input phase of the project is involved with finding and adding news to the system. The process phase is responsible for re-formatting and classifying news stories. Both the Input and Process phases of the project will sequentially execute plugins in an order dictated by a custom workflow. Between the database and Output phases sits an XML interface; this will essentially act

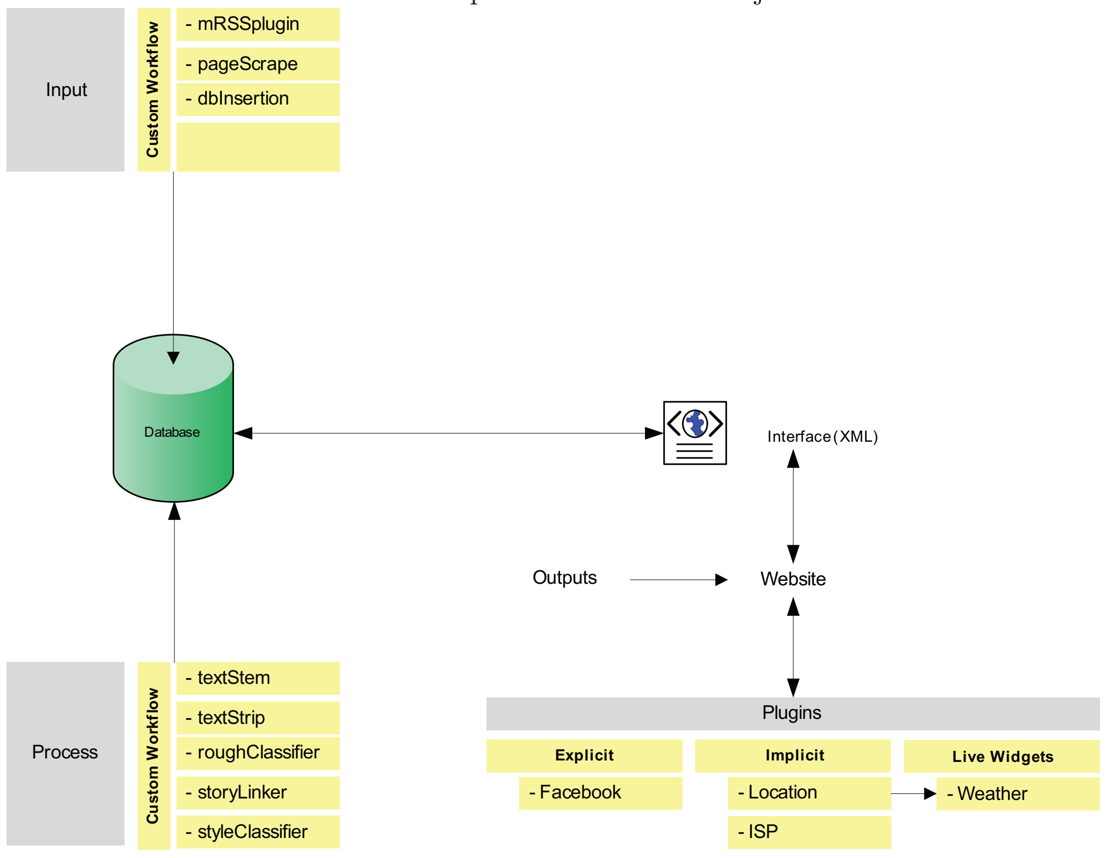
Figure 5.3: System Use-Case Diagram



as an API allowing developers to interact with the system without having to understand how it works. The Output phase utilises this API to provide tailored news to an individual.

5.1.2 Conceptual Overview

Figure 5.4: Conceptual Overview of Project Triton

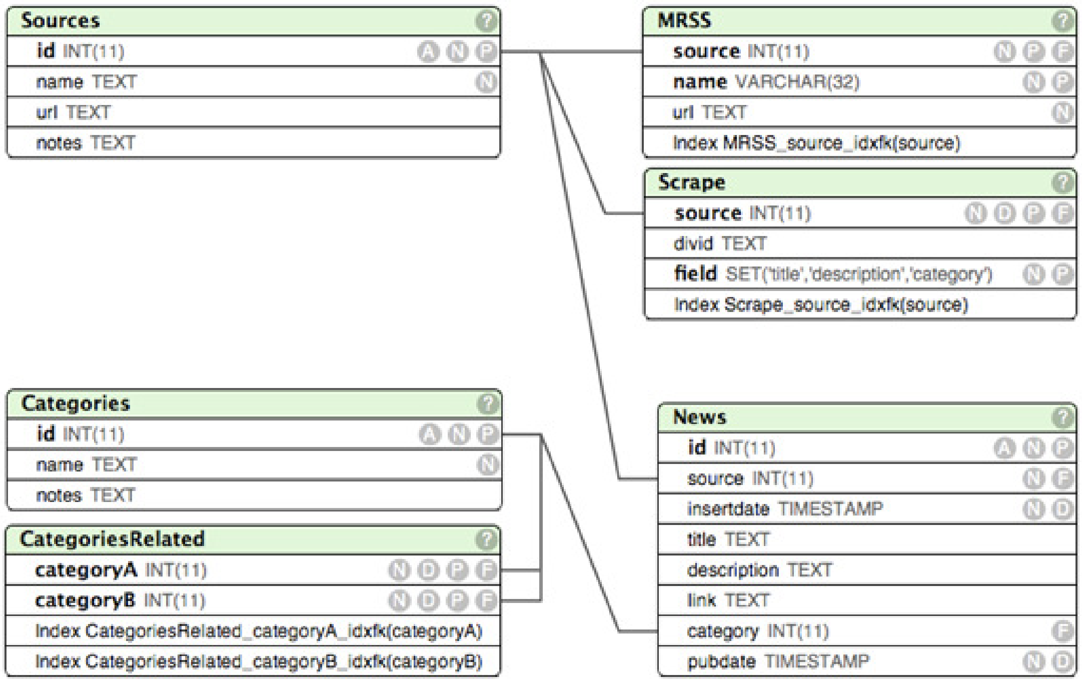


As represented in the conceptual diagram [5.4], Project Triton has four main elements. A data gathering stage - known as the Input Stage; a data processing stage - known as the Process Stage; a high-level interface to access the system, and Plugins to customise the provided service.

5.1.3 Database Design and Schema Overview

When I began the system, a key feature of my project was to create an expandable system. This focus on expandability meant that a rigid structure needed to be designed so that it was easy for other developers to integrate with the project. With an understanding of the basic structure of my project, I was ready to start formalising the system, with the specification of interfaces and a framework on which they can run. To do this, I needed to decide what information should be stored in my database on a simple implementation.

FIGURE 5.5: A Proposed Database Schema of Project Triton



Essentially, the basic framework of the system contains two main relations, called Sources and News with plugins generally creating their own tables on setup. The relations created by each plugin would typically reference one of the main tables (Sources for an input plugin or News for any other type of plugin). The relation named Sources contains information about each individual input source, while News contains any data about a news story. In the diagram above, you will notice that there exist additional tables 'Scrape' and 'MRSS', these have been created by plugins. Both of these relations contain relationships to the Sources table, and are used in the operation of these input plugins.

The use of a plugin-based system requires the system to be very formalised. To help create a rigid structure to the system, I designed a set of interfaces and directory structure to a plugin. For example, the inputPlugin interface specifies that each input plugin should return strings giving information about the plugin's name (getPluginName()), description (getPluginDesc()), and author (getPluginAuth()). In addition to this, each plugin will have methods for setting inputs to the plugin (setInputs($in, $src)) and to operate the plugin on the set input (doProcess()). Aside from implementing the appropriate interface, each plugin must contain files called install.php and db.sql in their root directory. These files are called by the system to install and initialise the databases for the plugin. In the future it is likely that methods designed for user-friendly debugging will be included in the design, although this is likely to occur in a later phase of development.

5.2 Evolution of Design

During the construction phase of the project, it became apparent that certain aspects of the project could be altered to suit their functionality better. Many interfaces became more generic, to allow for more complex plugins and customisation, which made other interfaces redundant.

5.2.1 Oversights and a Shift to Genericism

One of the initial changes made to Project Triton was the re-naming of methods in the interfaces. As input plugins exhibited increased variation and became more complex, methods were given more generic names such as 'doProcess()'. Similar re-factoring occurred throughout the project, as it was realised that certain methods did not suit the names they were written under.

The changes made to Triton however were not merely cosmetic. Chief among the changes made to Project Triton were the alterations made to 'Process Plugins'. When designing the system, it was initially thought that there would exist five different process plugins with very clear-cut roles. The first types of process plugins were the filter plugins, textStem (for stemming words in an article) and textStrip (for removing stop-words in an article). These filter methods were to be joined by the classification plugins; roughClassifier, styleClassifier and storyLinker. All process plugins were to operate under a single interface, and would operate on data in a sequence. This ideology was a major failing in project design. Upon developing the interfaces for these plugins, it was noticed that it was difficult to achieve a sufficiently generic interface for both the filter and process plugins. This resulted in the split between filter and process plugins. A stripped down interface was created for filters that would accept and return a data type specified by the filter class itself. In addition to this, a process plugin interface was created adding several extra methods and restricting the output of doProcess() to an array of internal references. A further failing of this stage of the design was that it was assumed that accurate topic and style detection could be achieved using one algorithm. Through extensive research it was found that their exist many different techniques to topic detection and similarity [2.1.2]. As such, the simplistic view of having three process plugins was scrapped in favour of a more flexible system with the ability to use many different algorithms in harmony to affect data recommendation. The resulting designed plugins ranged from the computationally expensive mongeElkan technique, to trivial plugins built around PHP's similar_text() method.

5.2.2 Optimisation Through Good Design Practice

A positive side effect from the re-factoring of the internal structure of Triton's process stage was that it allowed for the concatenation of the 'Output Stage' into 'Process Plugins', reducing the overall weight of the system. Before construction to the system had begun, it was thought that further segmentation of the project was required between process plugins and the output stage. Because of the increasingly generic design of 'Process Plugins', it was realised that it was possible to replicate the complete functionality of the Output Stage behind Project Tritons API.

As part of the 'Input Stage' of development, a custom XML workflow was written to allow full customisation of the inputs and order of execution of the input plugins. This was very heavily over-engineered, and during the later phases of the project, it became apparent that the design of the system had implied a tightly coupled structure to the project. By loosening the coupling of the project, a much cleaner and clearer project was created. While at the moment, the system currently runs in serial, it is hoped that a future evolution of the 'Input Stage' would see a spooled per-input implementation, running a full workflow as a new thread (using Alex Lau's thread simulation class [25]) for each new input to the system.

5.2.3 The Social Divide

Although many of these changes have occurred because of a highly generic approach to earlier modules, other plugins, such as the social network and third party interaction required changes as they where unfeasible in their original implementation. Originally the system was designed so that social network integration would occur as a batch process, where by a plugin - such as the Facebook plugin - would iterate through the associated accounts with each user and request interest information for each user to insert into the database for more processing. Unfortunately, while the designed system is very capable of performing this task, the Facebook API does not support such operations. Instead, the Facebook API requires any login to come direct from the user, redirecting any requests to their authorisation page first. While it is possible to automate this login process, using CURL commands, setting login details from a database, this violates the Facebook API terms and conditions [26]. For this reason, I decided to partition the Facebook plugin, creating plugins for interest processing (as a Process Plugin), while account association is to occur via a website.

5.2.4 Outgrowing Expectations

During the initial design phase of Project Triton, it was always envisaged that the databases for the system in part be maintained by database level rules, such as relationships. For this reason, when developing the Triton, all tables in the database were of the type InnoDB, which contains features such as foreign key support, and row level locking. However, when testing the input stage plugins, it became apparent that the database was far bigger than previously anticipated, with the main news table expanding by 70MB per parse despite all tables being in third normal form. Because of the vast quantities of data being entered into the database, the length of internal records had to be reduced so that 10e6 records where allowed in each table (compared to the original 10e11). In addition to this, all relationships were removed, and all tables in the database changed to the type MyISAM. These changes typically saw records reduce by a scale factor of 95

5.2.5 Adaptable Indexing

When designing Project Triton, the idea was to create a system capable of recommending news to a user, however, it was realised towards the end of the project that by re-factoring the system, it was possible to create a more generic data recommender. By creating a database driven wrapper, it was possible to add a notion of sources, which would cater for news, images, events and any other type of data to be recommended. In addition to this, by implementing the notion of indexable fields, the currently existing plugins could be extended to index elements of other data such as events or captions on images. This re-factor lead to a far more generic system, with very little overhead, and increased the flexibility of currently existing plugins.

5.3 The Prototype

As stated before, the primary focus of Project Triton, was to create a system framework, with example plugins which all sit behind an XML interface. These tasks make up Phases 1 and 2 of development [4.2], and would be developed in parallel. Further functionality was to be added after the completion of this in Phase 3, where Social Networking plugins, Implicit Plugins and the introduction of more sources where to be catered for.

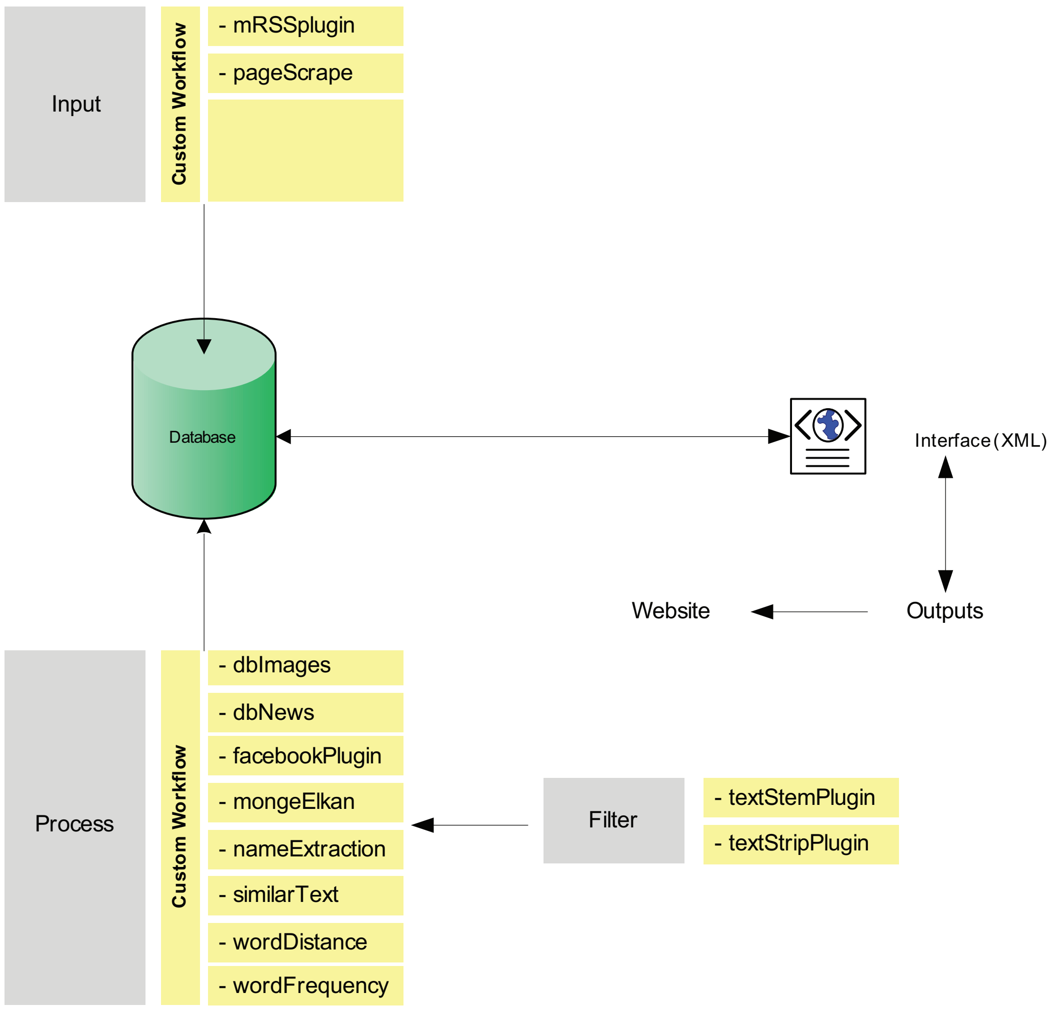


FIGURE 5.6: An Updated Concept Diagram of Project Triton

The prototype constructed, saw both Phases 1 and 2 complete, albeit with significant re-design and re-factoring. This allowed for Phase 3 to begin. The notion of Live Widgets was replaced with a Sources system, which allowed other non-news items to be catered for behind the PHP/XML based interface. Similarly implicitly plugins were moved behind the backend of the system, while Explicit plugins were split, sitting partially behind, and in front of the API due to security constraints. The move from Figure 5.4 - showing a conceptual model of Project Triton - to Figure 5.3 illustrates the significant changes re-factoring has had on the project while essentially keeping many of the same concepts and ideals of the original plans.

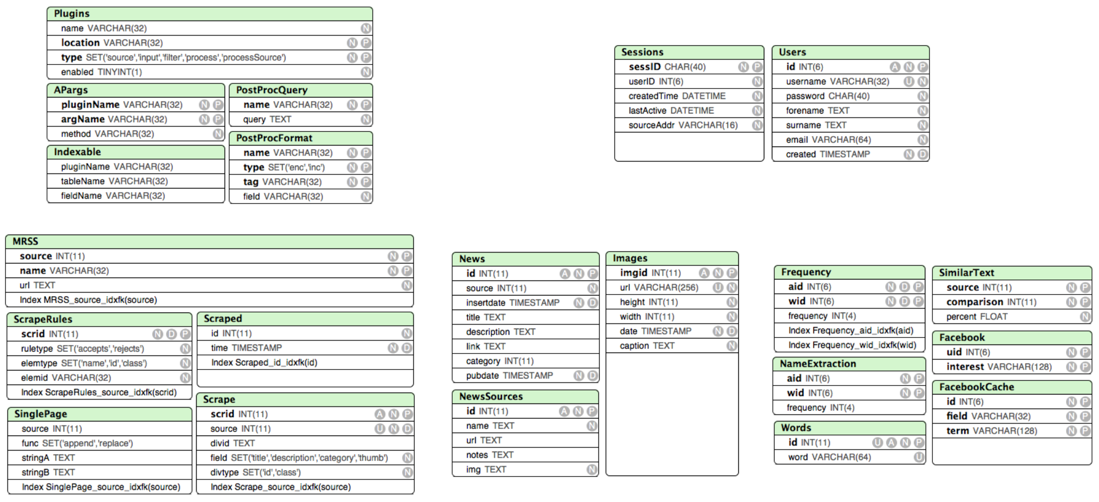


FIGURE 5.7: An Updated Database Schema of Project Triton

Figure 5.5 - detailing Project Triton's database schema - has also changed significantly. The project now ships with a default database schema shown in Figure 5.7. Further relations are added to the database when installing plugins. For example, by installing the dbNews plugin - a source plugin for news - the relations News and Categories will be added to the database. This self-installation system ensures that all distributions of Project Triton remain lightweight, containing no irrelevant data in the database. The functionality described is made possible by a structured interface that all plugins are required to implement.

Within Project Triton, there exist four types of plugin: input plugins; filter plugins; process plugins; and source plugins. Input plugins are involved with automated collection of data; while Process plugins apply algorithms to the collected data to infer knowledge. Filter plugins are responsible for removing a subset of data from their input, making processing less computationally expensive; while

Source plugins are required to interface with source databases, performing insertion, updating, deletion, and other database interaction. Each plugin contains an install and uninstall method. The install method adds the necessary configuration lines to the database if all pre-requisites are installed. Conversely, the uninstall method removes the plugin's tables from the database, leaving the pre-requisites intact. Typically process plugins add several records to the database involved with configuration, adding their own arguments to the API so that they may receive interaction from the outside world.

Despite some significant divergences, much of the system remained true to original designs. With the exception of the 'Editor', all actors stated in the use-case [5.3] were catered for. An executive decision was made to remove the 'Editor' actor from the system as the sheer volume of data passing through the system made editing a very active responsibility (with even the smallest systems increasing by hundreds of articles a day).

Chapter 6

Implementation

6.1 Technological Justification

A key consideration of any software project is to choose which language to develop the project in. As stated in my non-functional requirements, a vital component of the design is the need for a portable system. In addition to this, as a technically taxing project, it was important for me to use a language in which I had significant personal experience. These realisations left two major contenders in Java and PHP. Another factor to consider was the range of development hardware available. ECS provided a partial implementation of PHP on their servers with a limited installation of MySQL. I also had a Mac Mini, which contained more complete versions of PHP and MySQL by default. The use of Java in web applications is rapidly declining. Considering the speed and coverage losses when compared to PHP, it was obvious that the use of PHP and MySQL was correct for this project. As a result, I chose to implement my system using PHP and MySQL, which is installed in a large proportion of Apache distributions (the majority market-share web-server). I also felt it best to implement Project Triton on my Mac Mini server (found at http://27point3.com) as it provided a more realistic representation of a typical web server than the ECS provided solution.

6.2 Developing the Framework, Plugins and API (Phase 1 & Phase 2)

6.2.1 Working with Inputs

After an initial server configuration period, work began on the Input Stage of the framework (Phase 1). Following several iterations of development, an optimum implementation of an Input Plugin' was decided [6.2.1]. A common feature of all plugins in Project Triton is that they contain identification methods, which detail a plugin's name (getPluginName()), description (getPluginDesc()), and author (getPluginAuth()). These methods enable the system to display information about an active plugin to the user. Beyond these standard methods, a source plugin is required to contain a setInputs - which accepts two arguments to set the internal parameters of the plugin, and a doProcess which enacts the processing functionality of the plugin using the parameters set by setInputs.

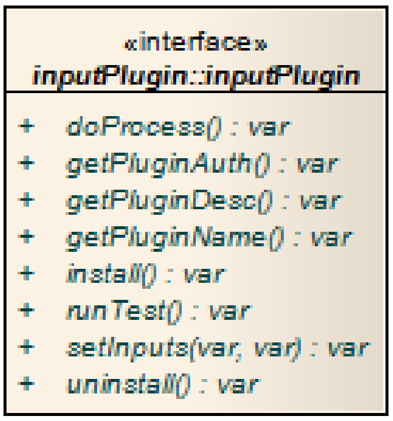


FIGURE 6.1: The inputPlugin Interface

Initially, it was thought that Project Triton contain two simple Input Plugins; namely a RSS reader, and page scraping plugin. In these early stages of development, simple functionality of the Input Plugin's was achieved ahead of schedule. With a small amount of surplus time, extra functionality of Triton was envisioned, resulting in added layers of complexity for the designed plugins. The main aim was to deliver a complete multimedia system. By expanding the RSS plugin to read other XML variants such as MRSS, it would be possible to extract further information such as images and videos relating to particular news stories, or even events. In addition to this, a more generic approach to page scraping was devised, which allowed complete programmability by utilising a MySQL database, enabling a plethora of features including scraping rules (allowing acceptance and rejection of data within tags) and URL re-writing (catering for single-page views).

During the early stages of development, it was thought that Project Triton should contain a custom workflow, which would be capable of passing around standardised vectors throughout the system. These standardised vectors would be inserted and updated using plugins expecting a vector of that order. This notion resulted in the creation of two extra Input Plugins; dbInsertion and dbUpdate. These plugins, respectively inserted and updated standardised news vectors into the News table.

6.2.2 Controlling the Flow - Developing a Custom Workflow

With the Input Plugins completed, development began of a custom workflow to allow easy re-ordering and editing of Project Triton's workflow. During the research stage of the project; several academic papers were read detailing different approaches to writing a custom workflow. While each different approach had its advantages and disadvantages, most of the papers agreed that an XML based standard was the most logical implementation. Despite one paper suggesting a XML based workflow language [20], this implementation was far too detailed for Project Triton's simple needs. For this reason, a smaller XML language was devised which allowed the administrator to vary the sequence of plugins, specifying their expected inputs and outputs, which were either query or vector based. This XML workflow was to be stored inside inc' of the projects folder structure named input.xml'. A further class was then created called workflow', which allowed for the iterative execution (runWorkflow) of a specified (setInput) workflow. This custom workflow indeed fully met its requirements, however in reality it forced the Input Stage to become very tightly coupled due to the notion of a standardised vector. Ultimately this lead to the depreciation of some of the above-described functionality as a revised system was introduced later in the project.

6.2.3 Process Plugins

Project Triton was now able to collect a wide range of data and provided an active encyclopaedia to operate upon. To provide a layer of intelligence to Project Triton, it was deemed that Process Plugins should be developed. The use of Process Plugins allowed Project Triton to perform analysis on the data gained from the Input Stage. These plugins would be required to include methods for plugin identification, maintenance and operation (importantly setInput, doProcess and getSimilar) as defined by the processPlugin interface [6.2.3]. It was recognised that a typical Process Plugin might require both online and offline processing capabilities. As a result of this, two operational methods doProcess' and getSimilar' were introduced allowing a Process Plugin to pre-process data before receiving live queries, improving performance.

In the initial designs for Project Triton, the Processing of data in the system was considered trivial. Designs dictated that there exist three stages to recommendation, namely rough classification, style classification, and story linking. Through research and experimentation, this task proved much more complex. Each different algorithm provided varying rates of success, and it became apparent that there was a difference between the notion of string and topic similarity. As a result, a wide variety of different Process Plugins were implemented; from the complex MongeElkan Plugin to trivial methods using PHP's built in text similarity operations. However, despite many approaches, few reached the level of accuracy or performance required by Project Triton. In all, just 3 classification process plugins were left activated in the final prototype, with several alternatives disabled. Figure 6.1 compares the developed plugins.

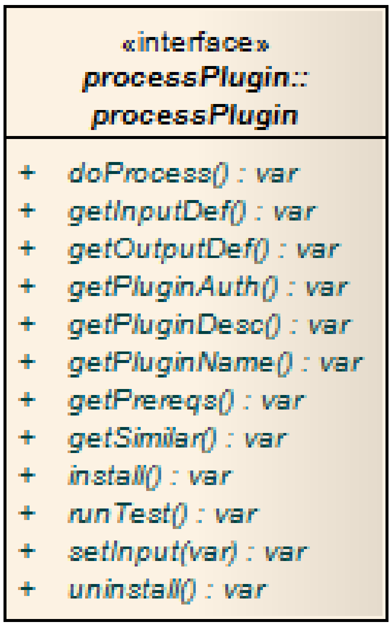


FIGURE 6.2: The processPlugin Interface

Table 6.1: A Comparison of Process Plugin Complexity

Name	Description	# of Lines	Complexity	Percentage Accuracy
mongeElkan	Translation from a Java implementation	358	O(n^4)	10%
nameExtraction	Word Frequency on words starting with a capital letter	200	O(n^2)	80%
similarText	Base on a built-in function of PHP	145	O(n^3)	50%
wordDistance	MAX, MIN and AVG calculations per word	156	O(n^2)	-
wordFrequency	Counts the occurance of each unique word in the corpus	205	O(n^2)	60%

6.2.4 Optimisation through Filters

During the development of the Process Plugins, it was realised that performance could often be improved by filtering out some of the input data. This led to the creation of Filter Plugins. Filter Plugins would strip an input - typically text - of certain features before returning. This allowed Project Triton to remove stop words (using textStrip), or reduce a string to its base form (using textStem). The use of these filters showed a positive improvement to all Process Plugins involved with topic detection. In contrast, several studies showed that reasonable style detection is possible by calculating the frequency of stop words in a document [5]. Filters followed a simple interface, which required the plugin to specify an operating data type, and supply methods for setting the input (setInput) and operating the filter (doProcess) in addition to the identification methods [6.2.4].

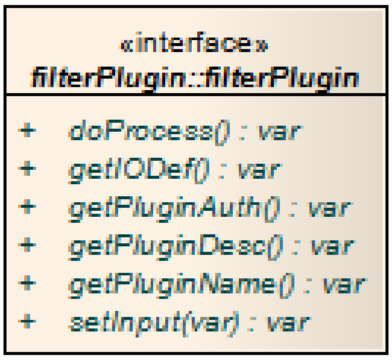


Figure 6.3: The filterPlugin Interface

6.2.5 Handling Requests - The API

With Phases 1 and 2 almost complete, it remained for a PHP / XML interface to be created, binding the full functionality of Project Triton. After a proof of concept implementation, which included significant amounts of static code, an expansion-centric approach was created making it simple to request data in XML or JSON format. Essentially, the Project Triton API would iterate through the active plugins, setting any parameters from the URL in sequence before executing the required plugins, applying any customisations based on user preferences. The result of this request would be formatted based on database set rules, and outputted in the requested format. By configuring the API via a database, it was simple to add commands to the API (using the APargs table) and vary the schema output (using the PostProcFormat table) with little or no hastle. This ensured the framework remained highly adaptable and easy to maintain as plugins could tailor all aspects of the framework within their install method to provide complete compatibility. The code extract in Figure 6.4 shows the novel approach to integrating a limitless variation of plugins.

NOTE: The code has been striped of all debuging methods to preserve modisty.

Figure 6.4: Code Extract from API.php

```
foreach ($plugins as $name => $plug) {
        // Load APargs and call appropriate method
        // APargs(pluginName, argName, method)
        $name = $plug->getPluginName();
        $query = "SELECT * FROM APargs WHERE pluginName = '$name'";
        $result = mysql_query($query) or die(mysql_error());
        $argued = false;

        while($row = mysql_fetch_array($result)) {
                if(isset($_GET[$row[1]])) {
                        $argued = true;
                        $arg = $_GET[$row[1]];
                        $mtd = $row[2];
                        $plug->$mtd($arg);
                }
        }

        if($argued) { $tmpRes = $plug->getSimilar();
                // Filter IN/OUT or MIX into $list
                foreach($tmpRes as $id) {
                        if($listIDs[$id] == null) {
                                $listIDs[$id] = 0;
        } else {
                $listIDs[$id] = ($listIDs[$id] + $weights[$name]);
        }
    }
        }
}
```

6.3 Phase 3 and Beyond

6.3.1 Source Integration

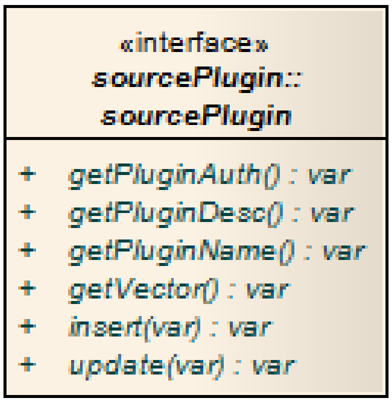
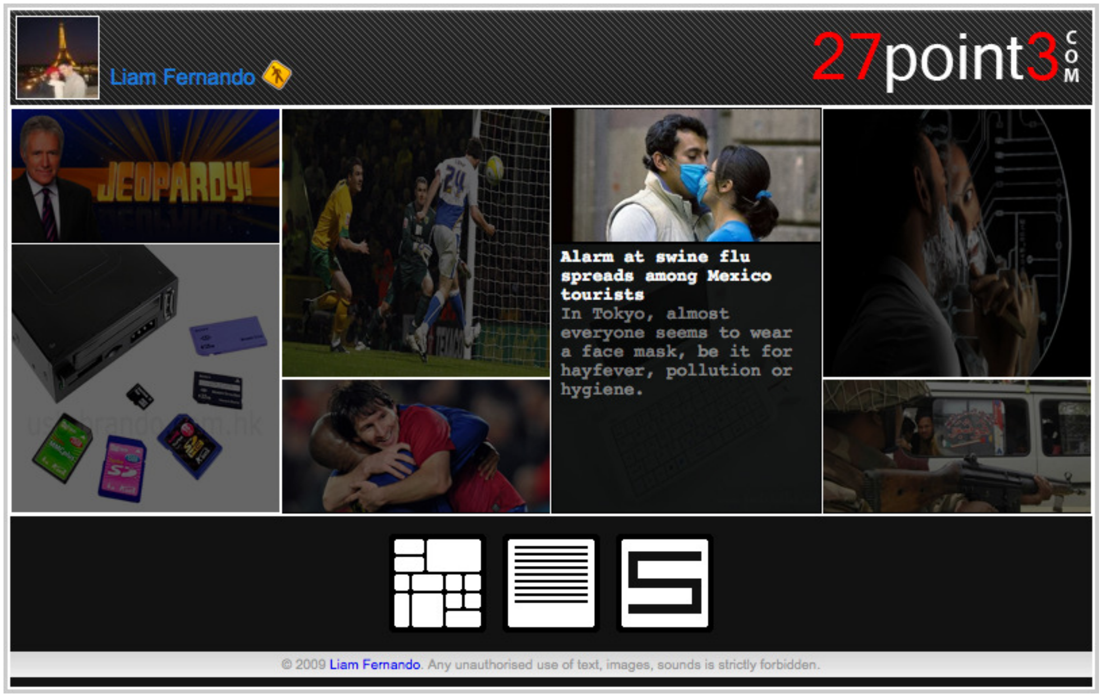


FIGURE 6.5: The sourcePlugin Interface

As Project Triton evolved, it was clear that some areas of the project required changes. Critically, the proposed implementation of using multiple different sources was problematic. Originally, it was thought that Project Triton would recommend news, attaching extraneous information to any associated news. However, due to the generic nature of Project Triton, it was possible to change the project's functionality so that it could recommend any data, using a notion of sources. Essentially, this required the project to be re-factored, such that all plugins within Triton were capable of running atomically in unison. Ultimately this lead to the introduction of Source Plugins, which provided both the means of describing source dependency and a method of interacting with a source library [6.3.1]. Despite the bonuses that sources added to Triton, further re-engineering was required to ensure that the project remained loosely coupled. With the introduction of Source Plugins, a situation was created where Process Plugins were required to know which fields of a source they were to operate on. By creating an Indexable table, it was possible to construct a system that could provide recommendations easily over many different sources.

6.3.2 Outputs and Social Interactions



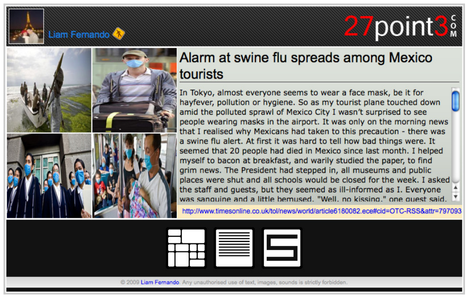
Alarm at swine flu spreads among Mexico tourists
In Tokyo, almost everyone seems to wear a face mask, be it for hayfever, pollution or hygiene.

© 2009 Liam Fernando. Any unauthorised use of text, images, sounds is strictly forbidden.

FIGURE 6.6: An early implementation of ITDS in Story Image Mode

A solid framework was now in place for Project Triton, with many proof-of-concept plugins designed to demonstrate system functionality. At this stage it was vital to create a mock application of Project Triton, which utilised the project's API, demonstrating that the system could recommend useful information, personalised through the use of explicit and implicit information. For this reason, the Intelligently Targeted Delivery System (ITDS) was created, forming a front-end to Project Triton. ITDS made use of the Project Triton framework by querying commands to the API via CURL, which responded with formatted (XML or JSON) replies. Implicit plugins (Process Plugins) for history-based (histRec) and location-based (ipLocate) recommendations, situated completely behind the API, provided further recommendation for users simply by extracting header or URL encoded information. Registered users were able to benefit from the introduction of social network integration, using facebookPlugin. facebookPlugin, was responsible for the filtration of any recommended data, ensuring the inclusion of any particular interests of a user's associated Facebook account. In stark contrast to the framework of Project Triton, ITDS was not intended to exhibit good design practices, and as such, the performance and reliability of ITDS sometimes dwindles. However, ITDS provides a proof of concept implementation, which sits atop of a highly advanced platform (the Project Triton Framework).

The implementation of Project Triton saw the completion of Phases 1 and 2 of development [4.1]. In addition to this, ability to cater for ranging Sources and Implicit Plugins stated in Phase 3 were fully realised by Triton, while the remaining Explicit functionality required was provided with assistance from ITDS. Overall, I feel this development cycle has been successful.



Alarm at swine flu spreads among Mexico tourists

In Tokyo, almost everyone seems to wear a face mask, be it for hayfever, pollution or hygiene. So as my tourist plane touched down amid the polluted sprawl of Mexico City I wasn't surprised to see people wearing masks in the airport. It was only on the morning news that I realised why Mexicans had taken to this precaution - there was a swine flu alert. At first it was hard to tell how bad things were. It seemed that 20 people had died in Mexico since last month. I helped myself to bacon at breakfast, and warily studied the paper, to find grim news. The President had stepped in, all museums and public places were shut and all schools would be closed for the week. I asked the staff and guests, but they seemed as ill-informed as I. Everyone was sanguine and a little bemused. "Well, no kissing," one guest said.

http://www.timesonline.co.uk/tol/news/world/article6180082.ece#cid=OTC-RSS&attr=797093

© 2009 Liam Fernando. Any unauthorised use of text, images, sounds is strictly forbidden.

FIGURE 6.7: An early implementation of ITDS in Read Mode

While the quality of recommendation by the system isn't always perfect, Project Triton has many strengths, and provides an extremely good platform on which to build a recommender system.

Project Triton provides a fully autonomous approach to data recommendation. The framework features a generic feed reader - capable of processing Media RSS - and a fully featured page scraper, allowing the system to amass large quantities of data. In addition to this, the system is able to aggregate a limitless range of different sources through the use of Source Plugins. These Sources are able to be specifically indexed using the Indexable table and generic Process Plugins. This means that it is possible to recommend data using any classification technique accross different data types; e.g. nameExtraction might occur accross news (using titles and descriptions) and images (using captions). By integrating with Social Networks, Process Plugins are able to target information directly to users without the need for tedious questionnaires. All this fuunctionality is provided behind a REST-ful API. This API allows Third Party applications to integrate fully with Project Triton wihtout the need to implement the whole system themselves. Furthermore this means that users of system enjoy transitive user profiles, as their interests follow them throughout the framework. The API offers huge benefits for developers as well. The Project Triton framework guides developers without dictating the exact behaviour of their applications. The API allows the specification of custom XML outputs, meaning that incredible freedom is given to developers in terms of customising and tailoring the API to their application.

Chapter 7

Testing and Evaluation

Throughout Project Triton, testing was persistent. In the initial stages of plugin development, testing would occur on a unit level, testing the individual elements of each plugin, such as properly initialising arrays and populating databases. When correct operation was proven, testing began on an integration level, first testing plugins on static inputs, then moving to dynamic database driven data, before finally integrating the plugin completely with the framework. Many of these tests while comprehensive, were not suitable for shipping with the final product. For this reason, the unit tests were re-written into module level tests, while the plugin interfaces and a testing wrapper offered some integration testing.

When formally testing Project Triton, I felt it was best to view the system as two distinct sections; the framework, and the plugins that operate on it. All plugins developed for the project are defined by interfaces. These interfaces require plugins to provide a testing method (runTest()'). The runTest() method allows the developer of a plugin to provide black-box testing on a per-module level. By including an automated testing method for plugins, unit testing is made much easier, and it is relatively simple to reduce errors into three categories: a loading error (typically a syntactical PHP error); an object error (where the plugin class does not match it's interface or an interface is not found); or a low level error (where an internal test of runTest() has failed - this will often be accompanied by an in-depth explanation of the error).

To supplement unit testing, I compared Project Triton's functional requirements with the resulting system, forming integration testing. Project Triton was designed to be a highly adaptable and extendable framework, which others can interface with to form user interfaces. It was therefore deemed unnecessary for the platform to require any user evaluation, as the system behaves the same way for all user interfaces, and can be customised to the individuals needs. By proving that the functional requirements have been completely satisfied, it is correct to say that Project Triton has been a success.

7.1 Unit testing

As my system was designed to be modular, I decided to use an automated approach to testing. This ensured correct operation at module level and made it extremely easy to detect and trap errors. The runTest()' method performs a series of tests specified by the module developer, echoing to the standard output (typically a web-page), returning a float where 1 notes a completely successful set of tests. By running these tests, I was able to complete and formalise the majority of unit tests for the system.

In the systems default state, Project Triton will ship with three filter plugins, two input plugins, eight process plugins and two source plugins. With the exception of the input plugins, these ship with fully automated testing methods, which test the full functionality of each plugin. Input plugins are harder to test, as they require static sources to use as operational data, therefore, the runTest() method in input plugins is used to check for failures in standard operation rather than a comprehensive black box test of the plugin.

The following table lists a brief summery of the results from the back-end unit testing of Project Triton. For an in depth detailing of the individual components of these tests, see Appendix A.

Backend Unit Testing					
Test ID	Plugin		Description	Results	
	Name	Type		Expected	Actual
1	textStem	Filter	Checks stemming result against a pre-calculated String.	1	1
2	textStrip	Filter	Checks stripping against a pre-calculated String.	1	1
3	thesaurus	Filter	Checks similar words against a pre-calculated String.	1	1
4	mrssPlugin	Input	Tests CURL and DOM Loading against google.co.uk	1	1
5	pageScrape	Input	Loads and tests the current page scraping queue.	1	1
6	facebookPlugin	Process	Ensures database integrity and connection to IMDB.	1	1
7	ipLocate	Process	Tests the retrieval of geo-relevant data.	1	1

8	nameExtraction	Process	Runs the name-Extraction daemon, performing variable analysis.	1	1
9	similarText	Process	Checks the response against a pre-calculated input.	1	1
10	wordFrequency	Process	Carrys out a variable level analysis of the algorithm on an active data source.	1	1

7.2 Integration testing

To demonstrate that Project Triton functions holistically, it is vital to perform integration testing by comparing the functionality of the project against Project Triton's functional requirements. Using the functional requirements from Table 3 the following analyses the success of Project Triton.

7.2.1 F1 : The system must be able to detect different styles and topics in news articles

Project Triton can be customised with a range of different plugins. In theory, this is only limited by the limitations of PHP as a programming language. By default, Project Triton ships with several plugins to detect and classify different styles and topics. The system is very capable of detecting different topics, and while algorithms such as word distance have been implemented to detect different styles, they often exhibit lower success rates than there topic detection counterparts. Using Corpus Analysis to Inform Research into Opinion Detection in Blogs' [5] is extremely revalent when undertaking such a task as style detection, and suggests that a better method of topic detection could be obtained by counting word frequencies of joining words (which would be removed by the textStrip filter).

7.2.2 F2 : The system must be able to classify news into given topics and interests.

In contrast to F1, F2 has been comprehensively met. Plugins exist to classify articles notably using word frequency and name extraction. Typically, the classification plugins for Project Triton are document pivoted, which has been proven to increase accuracy of matches.

Using the notion of indexable fields, Project Triton is capable of interest and topic classification over multiple sources (in addition to just news).

7.2.3 F3 : The system must be able to provide news to given interests.

While F3 has formally been satisfied, it is noted that the quality of some recommendations is not always perfect. This is not a failing of the framework, but of the recommendation algorithms implemented in the proof of concept plugins. The quality of the recommendations varies depending upon which plugins are enabled; those which showed particularly accurate recommendations are the nameExtraction and wordFreqeuency plugins.

Although the quality of recommendations can sometimes waiver, the system is able to cope well with recommendations across sources, using a notion of indexable fields. This means that recommendations are able to occur on news, images and events seamlessly through the application.

7.2.4 F4 : The system must be able to retrieve news articles autonomously.

Functional requirement 4 has been completely satisfied. Project Triton has a fully functional content retrieval system by default, which includes MRSS and Page Scraping plugins as standard. The system is managed using an XML based workflow and it is recommended that the script be run either as a cronTab or perpetual

PHP script (using sleep to wait between calls appropriately).

The designed solution actually goes beyond the functional requirement specified, fully allowing for not only news retrieval but other sources, such as images and events.

7.2.5 F5 : The system must be able to process news in a variety of different ways depending on its configuration.

F5 has been satisfied exceptionally. As a modular system, Project Triton contains a flexible framework that allows plugins to be installed altering how news or other sources are interpreted and processed.

7.2.6 F6 : The system must be able to retrieve a user's interests through explicit means (using social networks such as Facebook)

Project Triton is able to retrieve users interests through a combined use of process plugins and front-end integration. Due to the secure nature of many social API's, many social network require re-direction to their own server for a secure login. This means that batch processing all the users at the same time, using only a process plugin is not recommended. While this is possible to do, it often violates the terms and conditions of the social networks API. For this reason, it is necessary to add the social API interfacing code into a front-end (such as a website) and then use a process plugin to use the database cached interests to perform recommendations.

7.2.7 F7 : The system must be able to retrieve a user's interest through implicit means (using header information and or browsing habits).

In contrast to F6, the functionality of this requirement is able to be completely satisfied behind Project Triton's API. By creating the appropriate process plugins, it is possible to extract header information (as plugins are included into the called page) and cache requests by user (creating a history of browsing habits).

7.3 Reflection on Testing

Generally, I feel that Project Triton has succeeded. While Project Triton has met all of it's functional requirements, the system in practice remains a proof of concept because of the quality of recommendation algorithms used. This has been because of the time constraints placed upon the project. However, it is important to note that the projects framework has been very thoroughly developed offering vast amounts of flexibility whilst being easy to develop for, and this means that with the right recommendation algorithms, Project Triton has a real chance at being commercially viable.

Chapter 8

Project Management

8.1 Methods of Project Management

Throughout Project Triton, several different evolutions of project management systems have been implemented. Originally, it was decided that one of the main factors of creating a commercially viable product such as Project Triton would require an active developer community. As a way of enticing developers into creating plugins for Project Triton, it was thought best to create a publicly accessible website, containing up to date project information and details about the progress and project direction in the future. For this reason, a website was created which ran a version of activeCollab - a project management system which contains time and ticker tracking with the ability to log and plan jobs. In addition to this website, a SVN repository was created on a private computer (accessible through SVN over SSH). Hosting the SVN server privately was preferred over an ECS offered solution as the implementation offered by ECS was heavily restricted, lacking some of the major functionality of SVN. By installing some modules into activeCollab, it was possible to link the SVN and activeCollab installation in sucha way that the SVN repository was accessible through activeCollab.

In principle the designed developer community seemed perfect for Project Triton's needs. Sadly, however, the Gantt Charts produced by activeCollab where unconventional, and did not compare favourably to offline software packages such as Microsoft Project or ProjectWizards Merlin. When compounded with the large overhead involved in maintaining a project through a web-interface, it was ultimately decided that activeCollab be replaced by an offline solution. This resulted in the disbandment of the project's progress website, while a combination of SVN and ProjectWizards Merlin was used to manage the remaining elements of the project with focus changing to the developer community after the initial project completion.

8.2 Evolution of Project Triton

While at the beginning my project, progress remained true to the original predictions, several significant deviations occurred towards the later stages as major re-factoring and re-designs occurred in the project. One area of the project that underwent a particularly large change was the Output Stage', which was replaced with a Platform Extension' phase. In the platform extension phase, the system was completely re-factored to include the notion of source plugins, which enabled Project Triton to target different types of information to the user using the same plugins that were originally designed for targeting news. Additionally, it was realised that the already designed process plugins contained the required functionality specified by the explicit and implicit plugins. For this reason, the output plugins were scrapped and changed into process plugins, creating a lighter system.

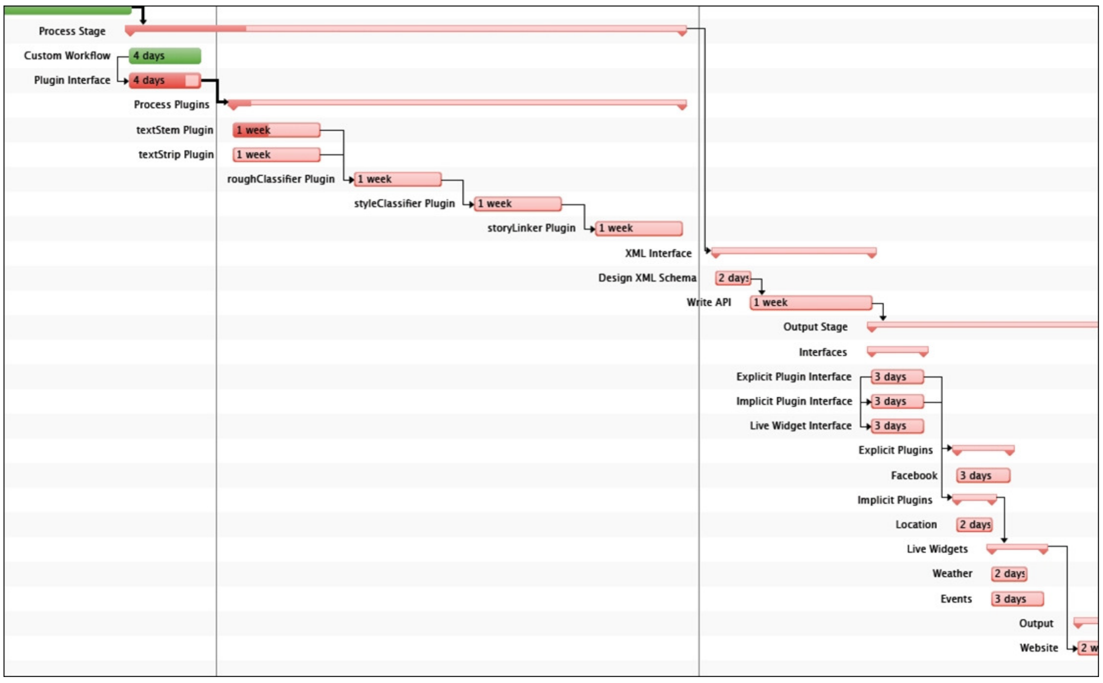


FIGURE 8.1: Comparison: The Original Gantt Chart of Project Triton

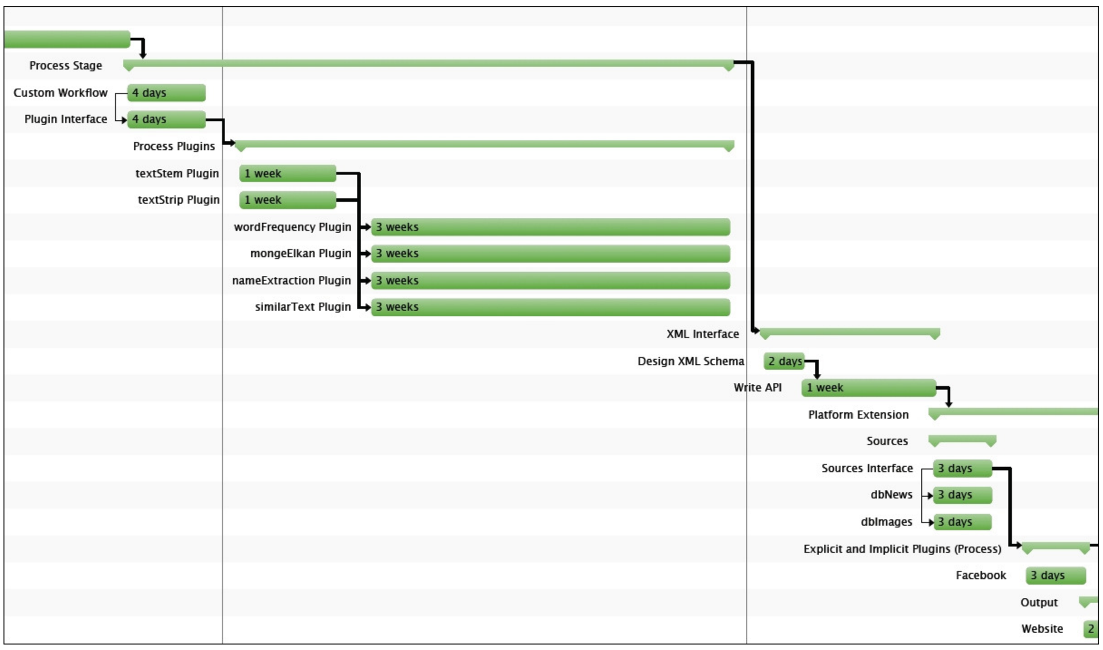


FIGURE 8.2: Comparison: The Final Gantt Chart of Project Triton

Chapter 9

Conclusions and Further Work

9.1 Achievements of Project Triton

On reflection, I feel that Project Triton has generally been successful. The designed system remains both flexible and easy to develop for, and could easily be used for recommending data to a user. Rather than creating a complete mediocre system, Project Triton has developed into a highly adaptable framework with several proof of concept plugins. While not all of these plugins provide a streamlined approach to the tasks they perform, they show the range of complex tasks able to be performed by the system.

9.2 Extending Project Triton

After completing the prototype system, further opportunities for enhancing the operation of Project Triton have become apparent. However, due to the rapid approach in development used in Project Triton, it is important to further stabilise the current implementation before adding functionality. The implementation of extra testing and validation scripts, with the possible introduction of standardised error codes would provide Project Triton further ease of development. In addition to this, the removal of excess functionality remaining from previous re-factors would reduce the system footprint. Once the consolidation process is completed, it is envisaged that the design and implementation of more accurate recommender plugins (Process Plugins) be the most important extension of the system, providing the real world quality expected by users. After this, the integration of more social networks would continue to draw more users to the system. Social networks which it is deemed Project Triton would have a strong affinity to are del.icio.us, Digg, Last.fm, Stumbleupon, Twitter and YouTube.

Beyond the implementation of more Process Plugins, another vital area of expansion is to create more outlets of the system. At present, the only method of interaction for the public is by using ITDS. ITDS is a mock system, and as such merely scratches the surface of the possibilities presented by Project Triton. In the future it is envisaged that implementations of Project Triton could be seen in widget form, integrated with Facebook, MySpace, NetVibes or iGoogle. It is thought that simply replicating much of the code found in ITDS could provide this functionality.

Many of the recommended extensions to Project Triton would increase popularity with either developers or users. To further increase the competitive edge Project Triton has, it is vital to fully utilise the Source extension features the project contains. The introduction of Events, Video, Adverts, Shopping and wikiPedia processing capabilities would enhance the system greatly.

The techniques described so far are technically relatively simple; the introduction of computer vision into Source Image Processing would give the system a level of accuracy unseen before in such a system. It is important to note however, that this would dramatically increase demand on the server, and the move to an enterprise solution such as GoogleApps [27] might be considered at this stage. It is also noted that because of the increased traffic experienced by the system, that the rate of data update (Input Stage) may be much slower. A possible resolution to this would be to distribute the system, separating the Input and Processing Stages. The introduction of Threads [25] in the Processing Stage would also enhance performance. This is not possible in the Input stage as plugins often require execution in a pre-defined order. However, due to the atomic nature of Process Plugins, it is possible to execute plugins in any order.

The changes mentioned above would probably push Project Triton towards success. It is important however to ensure that the API does not get abused. One might consider in the future selling the services of Project Triton, through the use of a subscription model. This might be introduced in such a way that only high volume users of the system are charged. An implementation of this might limit a user to a designated number of requests a day. By counting the number of requests by each user, it would be possible to bill their usage.

9.3 Creating a Commercial Viable System

Project Triton does however have weakness. While the framework is nearly perfect, the quality of plugins developed for the system is sometimes indifferent. If Project Triton were to succeed commercially, I feel a larger base of quality plugins would need to be developed, particularly those involved with recommending data. A major issue with the project is that the plugins developed tend to pertain to text similarity, rather than topic similarity; this was largely due to time constraints placed on the project as a whole. Aside from this, the inclusion of more social network integration and more resources for the developer community would help growth. It would be ideal at this point to create more outlets for the project to spread its popularity, perhaps using NetVibes, iGoogle or Facebook widgets that would interface with the API.

Beyond affirming the currently implemented system, extra contingency would be required in preventing the abuse of the system. This could occur when someone wishes to hot-link the API without having paid for a subscription.

9.4 Adapting Project Triton for Re-use

Ultimately, Project Triton could be used for many similar tasks in the area of data aggregation and recommendation. However, using the system in isolation, so that it becomes an RSS merger would perhaps seem somewhat wasteful of the system's power.

Because of the innate degree of customisation offered by Project Triton, many parts of the system can be easily tailored to perform other un-related tasks. The pageScrape plugin is a prime example of this; I am currently in the process of designing a web-based application called intelliChef that recommends recipes based on ingredients you currently own. As part of the preliminary research, I needed to collect recipes and extract lists of ingredients to insert into a database. I was able to use the pageScrape plugin to search the BBC Food website, and populate a database of ingredients and recipes by adding the necessary scraping rules into the database and writing a simple wrapper.

9.5 Summary

Project Triton successfully created a scalable framework with several proof-of-concept plugins demonstrating a range of complex tasks able to be performed on the system. Project Triton not only has real potential as a topic recommendation framework, but it is also highly amenable to re-use, demonstrating the flexibility of the project. While the implemented system is stable, work in the future might include further solidification of the foundations of Project Triton, before extending the range of plugins available. By improving the quality of the recommendation algorithms, and extending the range of sources that Project Triton can process, it is hoped that Project Triton's success will measure far beyond any current recommender solutions.

Bibliography

[1] TimesOnline. ”timesonline”. Jan 2009. http://timesonline.co.uk/.

[2] Engadget. ”engadget”. Jan 2009. http://engadget.com/.

[3] Fabrizio Sebastiani and Consiglio Nazionale Delle Ricerche. Machine learning in automated text categorization. *ACM Computing Surveys*, 34:1–47, 2002.

[4] Ana G. Maguitman, Filippo Menczer, Heather Roinestad, and Alessandro Vespignani. Algorithmic detection of semantic similarity. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 107–116, New York, NY, USA, 2005. ACM Press.

[5] Deanna Osman, John Yearwood, and Peter Vamplew. Using corpus analysis to inform research into opinion detection in blogs. In Peter Christen, Paul J. Kennedy, Jiuyong Li, Inna Kolyshkina, and Graham J. Williams, editors, *Sixth Australasian Data Mining Conference (AusDM 2007)*, volume 70 of *CRPIT*, pages 65–75, Gold Coast, Australia, 2007. ACS.

[6] Youngjoong Ko and Jungyun Seo. Text categorization using feature projections. In *COLING*, 2002.

[7] Ian Garcia and Yiu-Kai Ng. Eliminating redundant and less-informative rss news articles based on word similarity and a fuzzy equivalence relation. *Tools with Artificial Intelligence, IEEE International Conference on*, 0:465–473, 2006.

[8] Thierry Poibeau and Leila Kosseim. Proper name extraction from non-journalistic texts. In *In Computational Linguistics in the Netherlands*, pages 144–157, 2001.

[9] Eric Berkowitz, Mohamed Reda Elkhadiri, Tim Sahouri, and Michel Abraham. Intelligent content based title and author name extraction from formatted documents. pages 119–124, 2004.

[10] O. Bayraktar and T.T. Temizel. Person name extraction from turkish financial news text using local grammar-based approach. pages 1–4, Oct. 2008.

[11] Yang Lv, W.W.Y. Ng, J.W.T. Lee, Binbin Sun, and D.S. Yeung. Information extraction based on information fusion from multiple news sources from the web. pages 1471–1476, Oct. 2008.

[12] Alvaro Monge and Charles Elkan. The field matching problem: Algorithms and applications. In *In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 267–270, 1996.

[13] Scott Trent, Michiaki Tatsubori, Toyotaro Suzumura, Akihiko Tozawa, and Tamiya Onodera. Performance comparison of php and jsp as server-side scripting languages. In Valrie Issarny and Richard E. Schantz, editors, *Middleware*, volume 5346 of *Lecture Notes in Computer Science*, pages 164–182. Springer, 2008.

[14] Lance Titchkosky, Martin F. Arlitt, and Carey L. Williamson. A performance comparison of dynamic web technologies. *SIGMETRICS Performance Evaluation Review*, 31(3):2–11, 2003.

[15] U. V. Ramana. Some experiments with the performance of lamp architecture. In *CIT*, pages 916–921. IEEE Computer Society, 2005.

[16] J. J. Oliver. Decision graphs - an extension of decision trees. In *Proceedings of the Fourth International Workshop on Artificial Intelligence and Statistics*, pages 343–350, 1993. Extended version available as TR 173, Department of Computer Science, Monash University, Clayton, Victoria 3168, AUSTRALIA.

[17] Ganesan Velayathan and Seiji Yamada. Behavior-based web page evaluation. *Web Intelligence and Intelligent Agent Technology, International Conference on*, 0:409–412, 2006.

[18] Olfa Nasraoui, Maha Soliman, Esin Saka, Antonio Badia, and Richard Germain. A web usage mining framework for mining evolving user profiles in dynamic web sites. *Knowledge and Data Engineering, IEEE Transactions on*, 20(2):202–215, 2008.

[19] Min jeung Cho and Hae chang Rim. A location information retrieval system using ip address. *Advanced Language Processing and Web Information Technology, International Conference on*, 0:480–485, 2007.

[20] Xin Jin, Jing Xu, and Xuemeng Li. The design and implementation of xml-based workflow engine. *Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, ACIS International Conference on*, 3:137–142, 2007.

[21] E. Bozdag, A. Mesbah, and A. Van Deursen. A comparison of push and pull techniques for Ajax. In *Proceedings of the 9th IEEE International Symposium on Web Site Evolution (WSE07)*, pages 15–22.

[22] D. Crane and P. McCarthy. *Comet and Reverse Ajax: The Next Generation Ajax 2.0: The Next Generation Ajax 2.0*. APress, 2008.

[23] Mario Jarmasz and Stan Szpakowicz. The design and implementation of an electronic lexical knowledge base. In *AI '01: Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence*, pages 325–334, London, UK, 2001. Springer-Verlag.

[24] Tom Barnes. Rss. *Journal of Website Promotion*, 1:15 – 30, 2007.

[25] Alex Lau. "multi-thread simulation (thread) - php classes". May 2009. http://www.phpclasses.org/browse/package/3953.html.

[26] Facebook. "facebook — statement of rights and responsibilities". May 2009. http://www.facebook.com/terms.php.

[27] Google. Google app engine. May 2009. http://code.google.com/appengine/.

Appendix A

Project Brief

A.1 Problem

In the market today, vast arrays of solutions exist to deliver news to an individual, from newsvendors themselves, to themed sites that provide a pleura of information on a single subject. Few of these news delivery systems tailor the users experience to find news relative to their known and likely interests. Furthermore, those that do aren't extendable or implementable on a wide scale.

A.2 Goals

The goal of this project, is to create a framework on which plug-ins can be installed, customising the input, processing and user display of news. The implementation of this system should be such, that each installation is fully customisable to both the users needs and servers demands. The project is hoped to ship with several example plug-ins, which will be used, as the basis for further development, while at the same time be part of a fully usable product, which will have a prominent role in the market.

The project will essentially be separated into three distinct segments; the input phase, the process phase and the output phase. The input phase will be interested in reading in different input sources; these could be standardised sources such as XML feeds, however it equally likely that a plug-in for the input could process un-standardised inputs using page scraping techniques to needs or other media (such as images). The processing phase, will essentially involve plug-ins that group data and infer knowledge about data based on binding data, which may come from the input itself, or the users registered on the system. A typical plug-in may tag articles by set genres, or recognise different articles as the same story associating the source with a particular political stance (e.g. a news story from the Times, may be seen as Conservative). The output phase will involve plug-ins that deliver the content of the system to different platforms, such as a web-front end, XML feeds, an API for other websites to interface with the system, creating a fully extendable service.

While ideally it would be preferable to create wide range of plug-ins for the system, the focus of the project will be entirely on making a complete framework and the basis of a strong community based development. This will not only ensure that the plug-ins are varied, but push the development of the solution in the future.

A.3 Scope

To design and build a functional framework for news aggregation, which comprises of plug-ins to customise the experience to both the users needs and the demands of the server.

Appendix B

In-depth Testing Results

B.1 Backend Unit Testing

textStem (Filter plugin) *Checks stemming result against a pre-calculated String.*			
Test ID	Description	Expected Results	Actual Results
1a	Runs textStemPlugin=>doProcess() on the string: ”The quick brown fox jumps over the lazy dog” to produce the string: ”the quick brown fox jump over the lazy dog”. Success = 1, Fail = 0	1	1
textStrip (Filter plugin) *Checks stripping result against a pre-calculated String.*			
Test ID	Description	Expected Results	Actual Results
2a	Runs textStripPlugin=>doProcess() on the string: ”The quick brown fox jumps over the lazy dog” to produce the string: ”quick brown fox jumped lazy dog”. Success = 1, Fail = 0	1	1

thesuarus (Filter plugin) *Checks thesaurus result against a pre-calculated String.*			
Test ID	Description	Expected Results	Actual Results
3a	Runs thesaurusPlugin=>doProcess() on the string: ”quick” to produce an array of strings: ”active”, ”excitable”, ”fast”, ”fast”, ”hurried” and ”intelligent”. Success = 1, Fail = 0	1	1
mrssPlugin (Input plugin) *Tests CURL and DOM Loading against google.co.uk*			
Test ID	Description	Expected Results	Actual Results
4a	Successfully Populated the Method Array	1	1
4b	Successfully Queried Database for Sources	1	1
4c	Class dbNews Exists	1	1
4d	Class dbImages Exists	1	1
4e	Initialised CURL Object	1	1
4f	Valid CURL URL	1	1
4g	Successfully CURLED Data from http://google.co.uk/	1	1
4h	Successfully Created DOMDocument(doc)	1	1
4i	Successfully Loaded DOMDocument	1	1
4j	getElementsByTagName(”item”) Exists	1	1
4k	bkmon Array Successfully Populated	1	1
pageScrape (Input plugin) *Loads and tests the current page scraping queue.*			
Test ID	Description	Expected Results	Actual Results
5a	Successfully Queried for Scraped Information	1	1
5b	Successfully Queried for Enabled Sources	1	1
5c	Created dbNews Successfully	1	1
5d	Created dbImages Successfully	1	1
5e	Output Formed Correctly	1	1

facebookPlugin (Process plugin) *Ensures database integrity and connection to IMDB.*			
Test ID	Description	Expected Results	Actual Results
6a	Facebook Exists	1	1
6b	FacebookCache Exists	1	1
6c	IMDB Connection Successful	1	1
ipLocate (Process plugin) *Tests the retrieval of geo-relevant data.*			
Test ID	Description	Expected Results	Actual Results
7a	Implicitly Obtained IP Address	1	1
7b	Successfully Queried Geo-Location IP Address Information	1	1
7c	Located Country	1	1
7d	Located City	1	1
7e	Successful Query for Geo-Relavent Data	1	1
nameExtraction (Process plugin) *Runs the nameExtraction daemon, performing variable analysis.*			
Test ID	Description	Expected Results	Actual Results
8a	Indexing Query Successful	1	1
8b	Indexes found for ‘nameExtraction’	1	1
8c	INDEX: Source Table identified	1	1
8d	INDEX: Index Name identified	1	1
8e	Un-Processed Query Successful	1	1
8f	Successfully Initiated textStripPlugin()	1	1
8g	Loading Word Cache - Successful	1	1
8h	Successfully Cached Words to Array	1	1
8i	* Data Processing: Found Index Name [...]	1	1
8j	* Data Processing: Found Source Field [...]	1	1
8k	* Data Processing: Found ID Number [...]	1	1
8l	* Data Processing: Escaped Subject Exists	1	1
8m	* Data Processing: Corpus Successfully Split	1	1

similarText (Process plugin) *Checks similarText result against a pre-calculated String.*			
Test ID	Description	Expected Results	Actual Results
9a	similar_text("Test String 1", "Test String 2") = 11	1	1
wordFrequency (Process plugin) *Carrys out a variable level analysis of the algorithm on an active data source.*			
Test ID	Description	Expected Results	Actual Results
10a	SUCCESS : SET QUERY = SELECT id, description FROM News WHERE id NOT IN (SELECT aid FROM Frequency)	1	1
10b	SUCCESS : RESULT NOT NULL	1	1
10c	SUCCESS : STEMMER CREATED	1	1
10d	SUCCESS : STRIPPER CREATED	1	1
10e	* SUCCESS : AID = ...	1	1
10f	* SUCCESS : DESC = ...	1	1
10g	* SUCCESS : STRIPPED = ...	1	1
10h	* SUCCESS : STEMMED = ...	1	1
10i	* SUCCESS : WORD SPLIT - POPULATED ARRAY	1	1

Appendix C

Final Gantt Chart

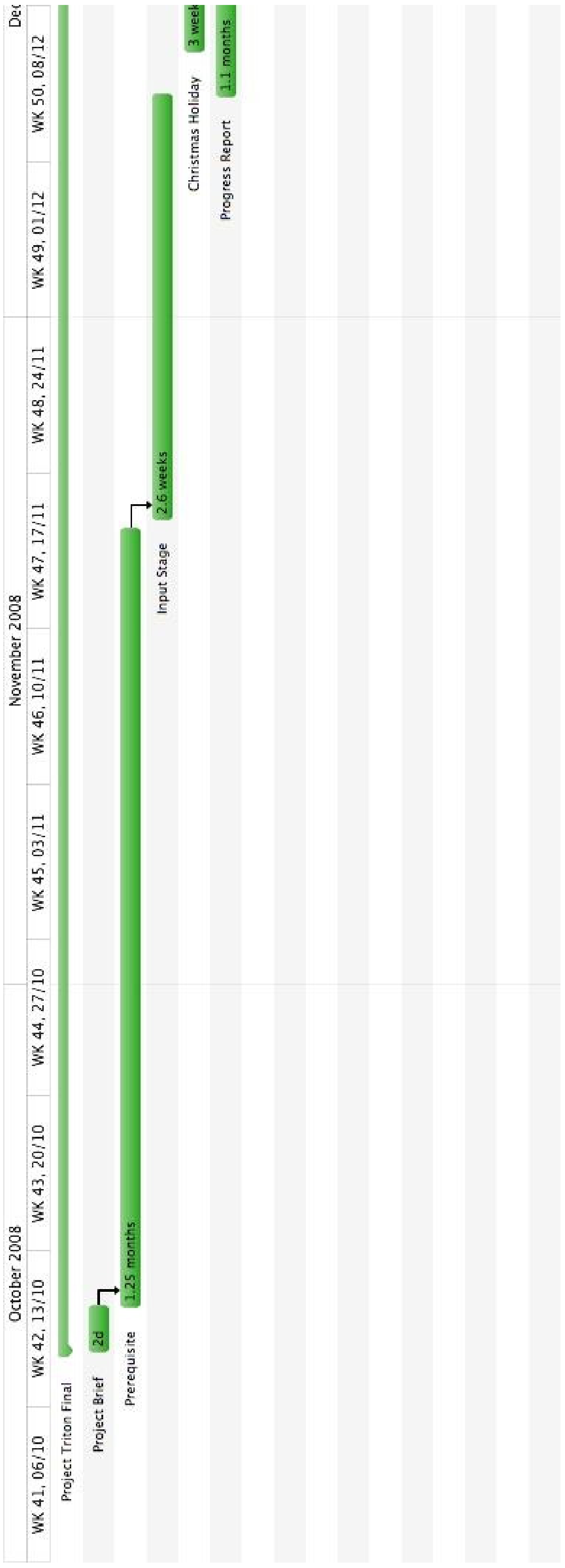


Figure C.1: The Final Gantt Chart of Project Triton - Part 1

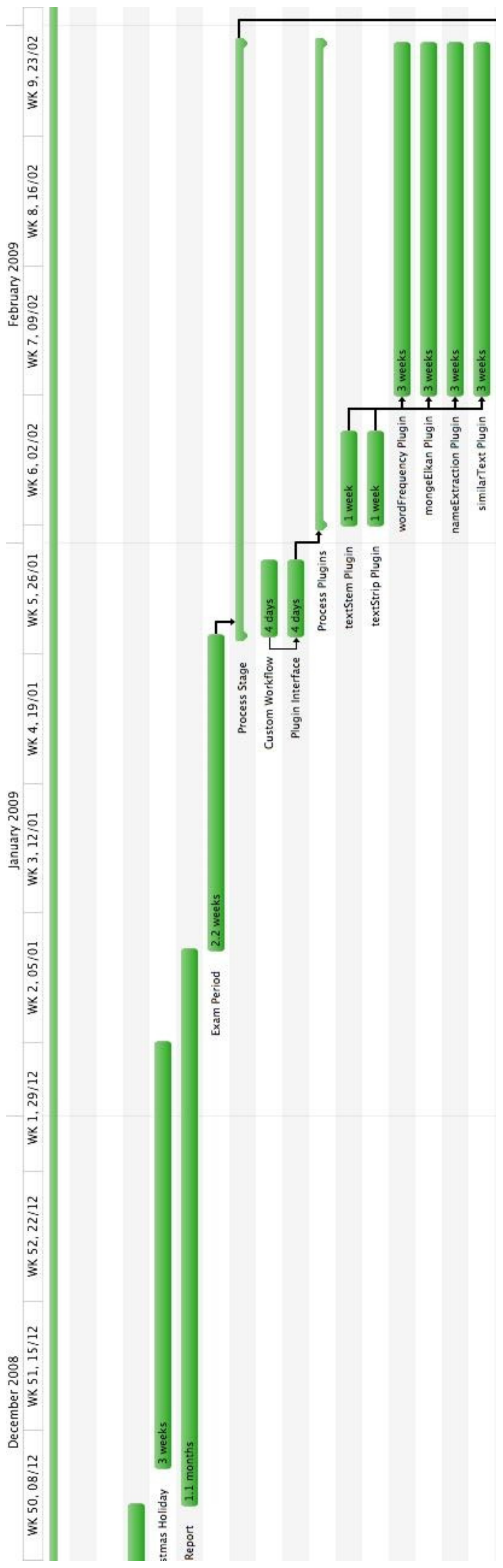


Figure C.2: The Final Gantt Chart of Project Triton - Part 2

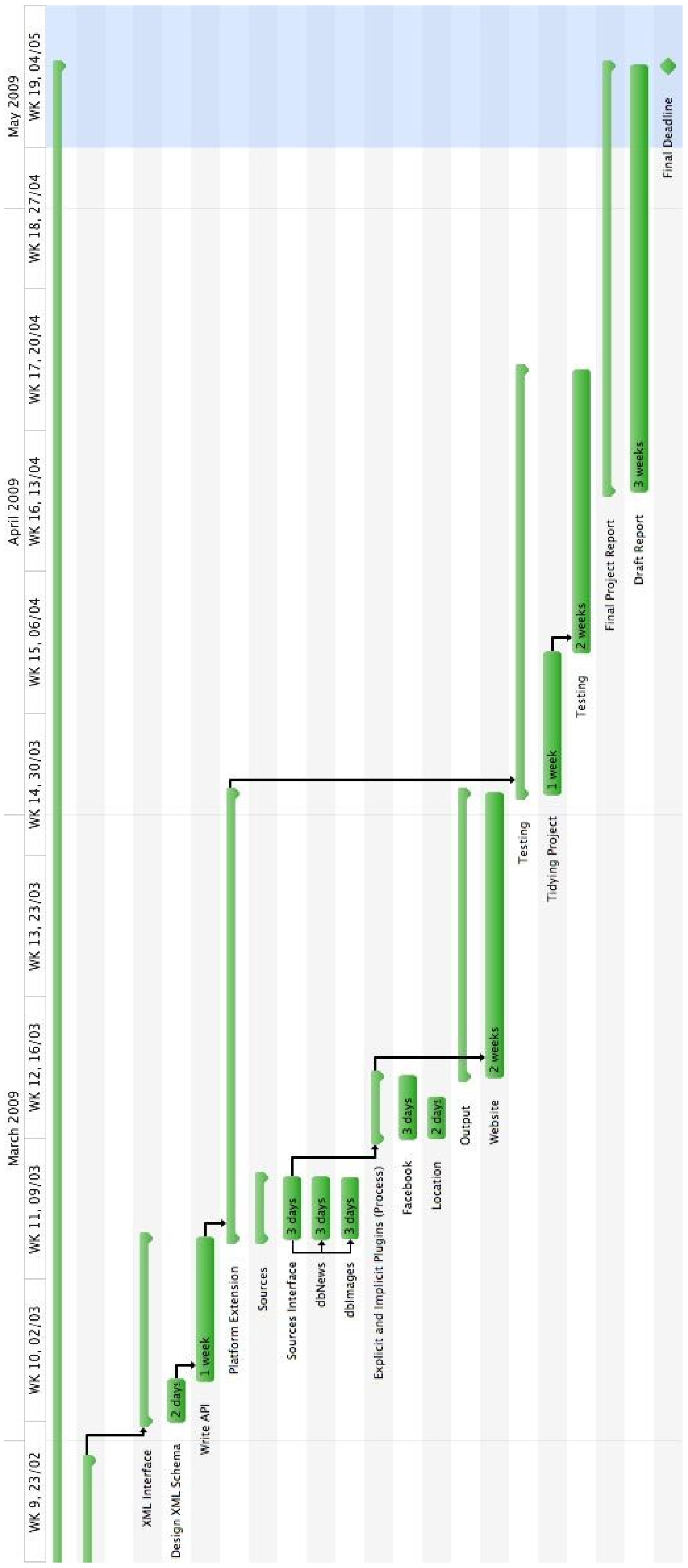


FIGURE C.3: The Final Gantt Chart of Project Triton - Part 3