Department of Computer Science

Technical Report

Undergraduate Dissertation: Information Retrieval and Gathering: An Experimental Prototype for Mac OS X

Victor Gonzalez - George

Technical Report 2006-16 **December 2006**

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

Information Retrieval and Gathering: An Experimental Prototype for Mac OS X

Victor Gonzalez - George
BSc (Hons) Computer Information Systems

2006

Information Retrieval and Gathering: An Experimental Prototype for Mac OS X

submitted by Victor Gonzalez - George

COPYRIGHT

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Batchelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed .....

This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed .....

Abstract

Gathertron is a user centred Information Retrieval and Gathering prototype for Mac OS X. Theoretical analysis of exploratory search techniques combined with users studies identified a set of eight design principles and system requirements which characterise a user oriented information retrieval and gathering system. These design principles guided the design and implementation of Gathertron, from a simple GUI prototype to a fully functioning application. Users were involved throughout the design, implementation and evaluation processes to refine Gathertron and produce a final set of requirements and principles which characterise a generic information retrieval and gathering system. It is concluded that Gathertron actively supports the information retrieval and gathering process of the user and provides a basis for future work in the domain of user oriented information gathering.

Acknowledgements

First and foremost I would like to thank Dr Leon Watts for his advice and guidance throughout the project and for keeping me entertained with endless discussion about the benefits of being an Apple Mac user! I would also like to thank Dewi for her help proof reading my dissertation, without her I would never have finished. Finally I would like to acknowledge Galvatron for giving me the inspiration for the name of my final system.

Contents

Chapter 1

Introduction

The central problem for this dissertation relates to how people find information to satisfy their information needs. Information Retrieval is a field of computer science research which is broadly concerned with the access of information. There is much confusion and debate regarding the true nature of information retrieval, ranging from mathematical modelling and algorithms for retrieval quality to the 'representation, storage, organisation of, and access to information items' [Yates and Ribeiro-Neto, 1999].

These two contrasting definitions of information retrieval introduce problems when designing information retrieval systems. Mulhem and Nigay [1996] identified a common problem with information retrieval systems is that they do not adopt a user centred approach and the design emphasis is placed on the technical efficiency of the system, instead of usability. This indicates that there is a gulf between human computer interaction and information retrieval systems.

Anna Aula, a Finnish researcher, is trying to trying to bridge the gap between information retrieval and human computer interaction. Aula has carried out numerous studies investigating human information retrieval processes and how a system can support these processes. In one study aimed at discovering web based patterns for advanced users, Aula identified a key characteristic of IR is the need to re-access information.

In addition to finding information for their current needs, people require methods for re-accessing information they have found earlier' [Aula et al., 2005].

This demonstrates the importance of developing an information retrieval system by studying users and then involving them during the design process, as this feature of IR may not have been identified without undertaking user studies.

When people search for information, there is a purpose associated with that search and this is where the role or purpose of the information retrieval system is crucial. An information retrieval system,

'does not inform the user on the subject of their inquiry, it merely informs on the existence and whereabouts of documents relating to their request' [Lancaster, 1968].

This definition of an information retrieval system suggests that information retrieval is the process of determining relevance of and locating documents which relate to the person's information need. An information retrieval system is not concerned with obtaining specific answers to well formed questions, rather, the collection of links to information pertaining to the person's information need. This is where the concept of information gathering enters the equation.

In its purest, most simplistic form, information gathering is the 'act of collecting information' [Miller]. Information gathering is an ongoing, recursive process of collecting relevant information for immediate or future use, thus supports Lancaster's definition of information retrieval. In the current home computer environment, there are no information retrieval systems which actively support the information gathering process. Therefore, this project aims to investigate the occurrence and characteristics of information gathering during the information retrieval process and develop an information retrieval and gathering prototype which supports the needs of the user.

1.1 Aims

The aims of the work reported in this dissertation have been divided into two categories, the project aims and personal aims. The project aims detail the high level aims of the project process, whereas, the personal aims describe the technical and professional skills that the author set out to acquire.

1.1.1 Project Aims

- To investigate the role of information gathering during the information retrieval process
- To define the characteristics of a person's information gathering process based on empirical research
- To define a set of principles for a user centred information retrieval and gathering system
- To follow a user centred design methodology to create a native Mac OS X 10.4 information retrieval and gathering prototype
- To provide a basis for future work into information gathering.

1.1.2 Personal Aims

- To investigate technology available on the Mac OS X platform
- To learn new development techniques and programming languages on the Mac OS X platform

1.2 Methodology

To achieve the project's objectives, it is necessary to combine research and engineering methods. A variation of the evolutionary development model, 'exploratory development' [Sommerville, 2001] has been adopted for this project. Exploratory development, as described by Sommerville, is the process of working with the users of the system to explore requirements and deliver the final system. This differs from the standard waterfall life-cycle as the requirements, design and development stages happen concurrently rather than sequentially. The methodology adopted in this project deviates slightly from the standard evolutionary development model suggested by Sommerville as the initial requirements specification was developed before the concurrent development and validation of the prototypes. It was vital to carry out end-user studies to investigate IR and IG techniques to guide the requirements specification before developing the first prototype. To validate the design principles suggested in the literature review, the first part of the project requires research into the basic nature of information gathering.

The design and developments stages are guided by a three stage evolutionary prototyping technique also suggested by Sommerville. This approach involves the development of three prototypes, with users evaluating each iteration to guide the design of the next prototype. Figure 1.1 illustrates an adapted version of the Evolutionary Development Model presented by Sommervile. The main difference is that the requirements specification is no longer performed alongside the development and validation. The requirements specification, which is based on theoretical analysis in the literature review and empirical studies, provides the basis for the initial system prototype in the design phase. The development produces intermediate working prototypes which are validated and redesigned until the final prototype, an evolution of the intermediate prototypes, is created. Throughout this entire process, the requirements will evolve to produce a final set of requirements or principles which characterise an IG and IR system. An exploratory method was chosen as traditional waterfall models are extremely linear and do not provide much scope for redefinition of requirements and design.

1.3 Structure

- Chapter 2 - Literature Review: Information Retrieval, Browsing, Foraging and Gathering. The literature review introduces the field of exploratory search and discusses exploratory search techniques, such as, Information Retrieval, Foraging and browsing. The review discusses the benefits of adopting user centred design approaches and examines the current interpretation of information gathering. The chapter concludes with a set of principles used to guide the design of an IR and IG system.
- Chapter 3 - Studying Usage of IR and IG. This chapter introduces the Macintosh platform and moves on to the first user study which analyses an existing search tool, Spotlight. The second user study is then introduced, which analyses users search patterns with the aim of identifying the characteristics of information re-

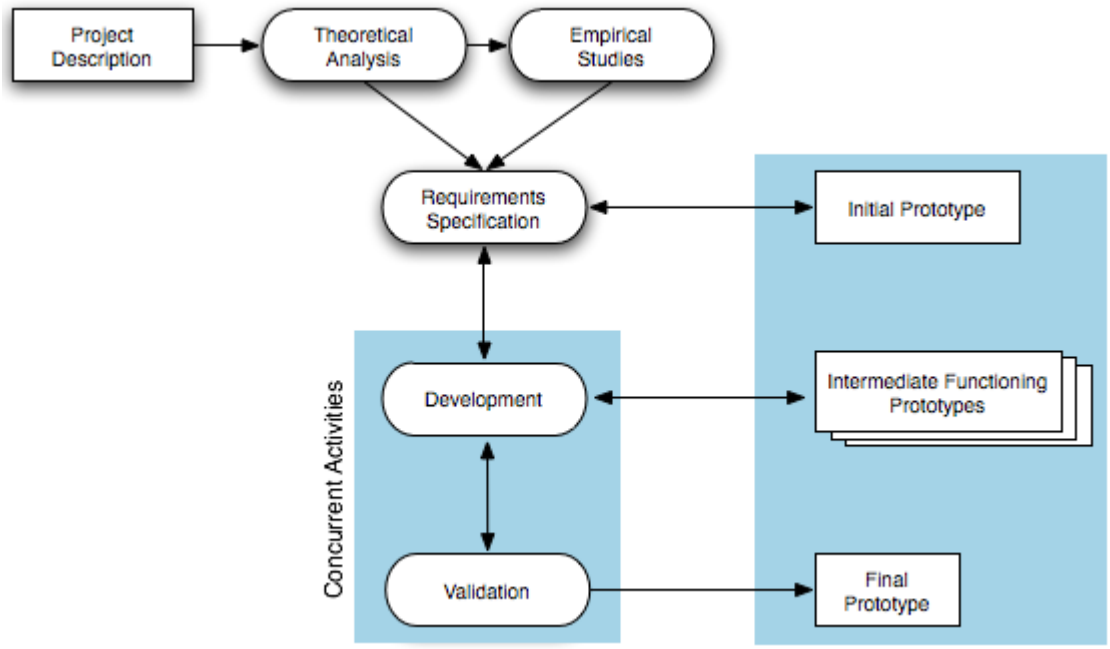

Figure 1.1: Adapted Evolutionary Development Process Model

trieval and gathering. The results of the evaluations are analysed and used in conjunction with the design principles derived from the literature review to create the requirements specification.

- Chapter 4 - Design of 'Searchlight', a Prototype IR and IG System. This chapter details the high level systems architecture derived from the requirements specification and uses the MVC design paradigm to design the model, view and controller objects in the system. The first prototype, 'Searchlight' is developed, which is a sophisticated GUI prototype used for end user evaluation. The evaluation results are detailed and 'Searchlight' is redesigned to implement the suggested changes.
- Chapter 5 - Detailed Design and Implementation of 'Searchlight'. This chapter uses the prototype developed in design chapter and implements the system functionality to create the first working prototype. The MVC paradigm is revisited to guide the implementation of the system and the view layer is redesigned to comply with Apple Interface Design Guidelines.
- Chapter 6 - Evaluation of Searchlight and Generic IR and IG Requirements. This chapter discusses the evaluations for the first fully functioning iteration of Searchlight. The initial evaluation results are presented and the prototype is redesigned to create the final prototype, 'Gathertron'. Gathertron is then evaluated and distributed to twelve Mac OS X 10.4 users and as a method of further evaluation. The final system is then evaluated against the design principles identified in the literature review.
- Chapter 7 - Conclusions - The conclusion offers an overview of the entire project

and moves on to a critical analysis of the main stages. Information Gathering is redefined with reference to the design principles and the future work is discussed.

Chapter 2

Literature Review: Information Retrieval, Browsing, Foraging and Gathering

2.1 Introduction

There are several ways of conceptualising the problem of how people find information to meet their needs. Each conceptualisation suggests a conceptual model for the interaction a person may have with a system which has been designed to furnish them with relevant information. This literature review aims to investigate Information Retrieval (IR), Information Foraging (IF), Information Browsing and Information Gathering (IG) to produce a set of principles which characterise a person's interaction with an IR and IG system. These principles are used to create a requirements specification which is used throughout the project to guide and evaluate the final system.

The review commences with a discussion of exploratory search techniques and focusses on an in depth analysis of IR. User centred Information Retrieval System (IRS) design methodologies and principles are introduced and an example a system, TAIPRI, which has been designed to support user information needs at the query formulation stage of the information retrieval process is discussed. The main stages of the IR process, from query formulation to information clustering and categorisation are analysed, with reference to systems such as 'Findex', 'Session Highlights' and 'FIRE'.

By exploring IF and browsing, this review attempts to discover the alternatives to query based IRSs. Information Gathering is then introduced with particular reference to studies carried out by [Schwartz and Pu, 1994]. The review concludes by examining the principles which characterise an IR and IG system.

2.2 Exploratory Search

Data retrieval systems such as search engines, digital libraries and databases are ideal for supporting the retrieval of data if the person has a well defined query. Data retrieval systems support well formed, question and answer style queries, where the person is aiming to retrieve a particular item of data, such as cinema times or sports results. However, data retrieval systems do not support poorly formed information problems which require exploration of the conceptual information space. If a person has limited knowledge in a particular domain, for example, aquatic animals, how would they formulate a query to retrieve information to improve their knowledge on the subject? When we attempt to address a poorly defined information problem, we submit a tentative query and explore the retrieved information 'selectively seeking and passively obtaining cues about where the next step lies' [White et al., 2006]. Exploratory search aims to blend the individual conceptualisations of information search, such as IR, IF, IG and information browsing to create highly interactive systems which allow people to search the information domain to solve poorly defined information problems.

Figure 2.1 illustrates three search activities, look-up, learn and investigate [Marchionini, 2006]. Look-up is the most basic and direct form of search and typically involves direct question and answer style searches, or well structured information problems, for example, 'what are the train times for the next train to London?'. Learning and investigative searches are activities which require more cognitive processing over longer periods of time to extract knowledge from information. For example, where a look-up search may involve retrieving data which answers a single instance of a problem, learning and investigative searches involve the retrieval of information and the assessment of its relevance and validity for use at that moment of for future reference.

2.3 Information Retrieval Systems

As stated in the introduction an IRS 'does not inform the user on the subject of their inquiry, it merely informs on the existence and whereabouts of documents relating to their request', therefore, this definition excludes data retrieval systems. Data retrieval systems have very different characteristics to IRSs, for example, when people use IRSs they are aiming to retrieve relevant information which represents a partial or best match to their queries. With data retrieval systems, people aim to retrieve exact data which exactly matches their query. For example, a person may use an IRS, such as Google, to search for information pertaining to the domain of 'wild boars', but they may use a data retrieval system to retrieve the television schedule for 'BBC1 on Tuesday 14th June'. This distinction is important because data retrieval systems support people with well defined information goals, whereas, IRSs are more exploratory and aim to address poorly formed information problems.

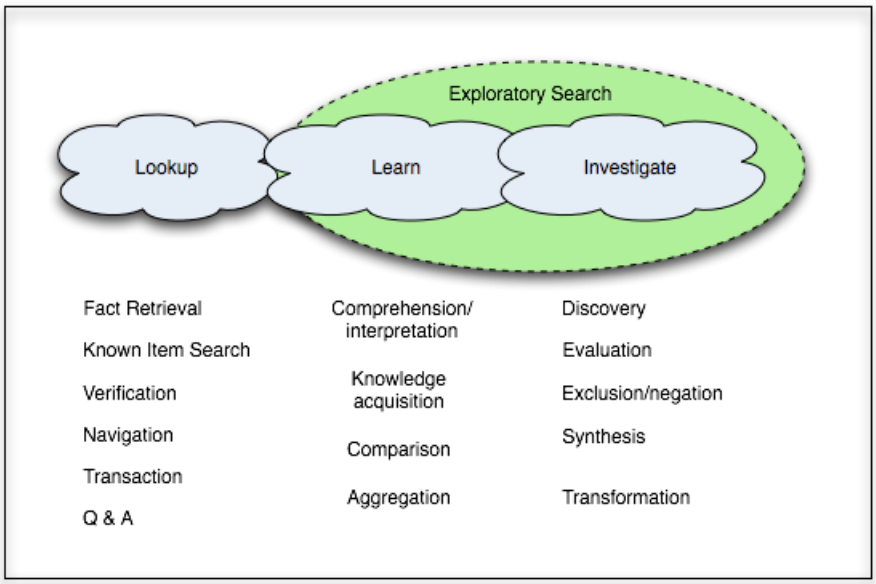

Figure 2.1: Search Activities

2.3.1 User Oriented IRS design

A common problem with IRSs is that they do not adopt a user centred approach [Mulhem and Nigay, 1996]. It is suggested that all experiments conducted to discover the efficacy of an IRS use metrics to assess retrieval speeds and the number of correctly matched query terms. These may be valuable metrics to analyse the technical efficiency of the system, but they do not assess the usability or account for user interaction. Parallel investigations into HCI for IR and interactive system design have suggested the following design principles: [Mulhem and Nigay, 1996]

- Conceptual Design: the system is built around the conceptual information needs of people
- Experimental Design: software design implemented and then tested on people. This is an iterative process

A combination of the two design methods would be the most effective technique of creating user-oriented systems. An IRS should support human information needs; combining conceptual and experimental design allows the designer to identify these needs and work with people to develop a system which matches their requirements. This human approach to creating an IRS is reinforced by the definition of IR as 'human problem management' [Mulhem and Nigay, 1996].

Feedback and Support

There are three main entities to take into account for any instance of IR: [Brajnik et al., 1996]

- The user of the information
- The IRS itself
- An intermediary - in the form of a person or the mechanism between the user and the information system; for example, an interface).

Feedback and support in any system is vital to ensure people of all abilities are able to use it effectively. Even the most intuitive systems need to offer some degree of guidance for both novices and people with more experience. Support can be classified in the following ways: [Brajnik et al., 1996]

- Technical Support - to enable correct use of interface or system. For example, a system such as Google simply requires the person to enter a search term and then select the 'Search' button. Technical support would inform the person of this process
- Conceptual Support - to support the model implemented by an IRS. Conceptual help can be classified as 'terminological and strategic support' [Brajnik et al., 1996].

Terminological support improves the person's query formulation vocabulary. Most IRS's support more advanced query formulation. For example, the use of speech marks to force exact string matches, or the use of boolean operators such as AND, OR. Strategic support suggests advice on how to improve the number of results returned. For example, removing superfluous terms such as 'the' and 'of' to increase the number of relevant results [Brajnik et al., 1996].

The difficulty of system support is finding a method of providing support effectively, but, support should not be completely excluded from a system. If a person understands how to formulate queries and grasps the concept of IR but cannot use the system, the system has failed. The same can be said for providing technical support and no conceptual support. The person may understand how to use the system or interact with the interface, but may not understand how to formulate effective queries, thus never returning correct results.

Most systems will offer support in the form of help documents or online support, but people often do not have the time or the inclination to read support documents. [Brajnik et al., 1996] identify six methods of providing support:

- Contextual - support for the person at the current stage in the IR process.
- Generic - general system support, such as an FAQ.
- Unprompted - support is automatically displayed on the screen when a perceived problem arises.

- Prompted - the person must explicitly ask for support.
- User Controlled - the person is able to browse through related support entries as desired.
- System Controlled - system provides a list of support entries and the user must select an option and move on.

They carried out an experiment using FIRE, an intelligent search interface, which attempted to emulate a human intermediary by essentially interacting directly with a person and supporting query formulation. FIRE was a prototype which investigated preferential support techniques for query formulation and information navigation. FIRE also allowed the person to assess and classify retrieved documents and categorise them based on relevance.

The results from the experiment demonstrated that:

- Explicit terminological support is important in contextual form.
- The automatic reformulation of queries using FIRE does not improve performance
- User-focussed, strategic conceptual support should be provided and unprompted as it improves user query formulation
- Technical support improves system usability - should be both prompted and unprompted
- Systems should enable user-controlled interaction

The experiment demonstrates key areas which must be considered when designing an IR system. Nevertheless, there are constraints with the experiment. The people used in the experiment had to be trained to use the system. This is not representative of a real system because people are not normally trained before they use a search system. Although, in live environments, mediators may be used to give people a quick demonstration of how to use a system. For example, a friend or colleague could introduce a new IR system and explain how to use it. Novices were not included in the experiment, third and fourth year computer scientists were used. This does not reflect the computer skill of the average person. Advanced computer science students already have a solid foundation in query formulation and this is likely to limit the generality of Brajnik et al's findings.

2.3.2 Query Formulation

It could be argued that many IRSs are built to produce fast and verified results and rarely take peoples needs into account. A system called TAIPRI [Mulhem and Nigay, 1996] was developed to support the information needs of a person when attempting to translate their information need into computer query form. The input to the system consisted of a query field where the person entered search terms and their approximate weightings. For example, if they wanted to search for cats and dogs, but wished to find more about cats, they would place a heavier weighting on the term 'cat'. Mulhem

and Nigay suggested that it is beneficial to present the person with a list of search terms to use when formulating their queries. TAIPRI employed a browsable list of search terms which the person could use to formulate their query. If the appropriate keyword was not in the list, then the person was advised to use a synonymous term; the example given was, substituting 'vehicle' for 'car', 'lorry' etc. This could be viewed as a contradiction to user-oriented system design because the person is being forced to change their information needs based on the system design. The idea of weighting a query also conforms to this because the person may not have a clear conceptual idea of exactly what they are trying to achieve. The system did provide limited conceptual feedback in the form of a circle which divided the search terms in to proportions. The concept of reformulation of queries based on the output of a query was suggested, but, the impact of this was not assessed.

The main constraint of the TAIPRI project was that at the time of publishing, Mulhem and Nigay had not performed usability tests, therefore, the concepts they propose have yet to be empirically validated. The most important notion to take from this work is the idea of user oriented design approaches and to treat IR as a human process. An IRS should support peoples' needs and help produce meaningful results as opposed to providing the fastest query response.

2.3.3 Query Language Analysis

Most web-based IRS's are query language based. Anne Aula, Natalie Jhaveri, and Mika Kaki conducted investigations to discover web-based information search patterns for advanced users. Even though the investigation was web-based, the concepts are applicable to the desktop searching environment. The results demonstrate how the system a person uses can support or inhibit the IR process. Re-access to information, discussed in section 2.6, also features heavily because,

'In addition to finding information for their current needs, people require methods for re-accessing information they have found earlier' [Aula et al., 2005].

Aula, Jhaveri and Kaki sent questionnaires to a group of 226 people with advanced web skills. Each person was asked a series of questions and their responses and comments were recorded. Each person was instructed to use their preferred search system and answer the questions accordingly. Over 99% of the applicants used Google as their preferred search system. The main finding were as follows: Over 50% of the people questioned did not understand the system's representation of a simple, operator-free query, for example assuming the default boolean operator was 'OR'. This demonstrates that a lack of feedback in a system can lead to incorrect usage, even for people with greater web experience and skills.

The most common operator respondents reported using was speech marks. Boolean operators such as 'OR' were rarely used. An operator which was used frequently was the site operator which limits the sites included in the search. This could indicate that people assume limiting a search field will return fewer unwanted results. The other most

prevalent operator was 'NEAR' which indicates that people do not always have a clear understanding of exactly what they are looking for.

An interesting comment was that people with greater web experience often formed similar queries to those with less experience. The queries may have taken the same form, for example using the same operators, but the major difference was the phrases or terms included. People with greater web experience chose search terms more carefully to produce more accurate results. This indicates that people are content to use simple queries and trust the system to produce results that are consistent with the content of the search strings.

It was also discovered that people seem to trust the algorithmic efficiency of Google, as many stated that they rarely use results past the second page as they, 'don't care for thousands of queries, because they usually look at first few anyway' [Aula et al., 2005].

One concern with the experiment is that the results are based on questionnaires as opposed to observation of behaviour. It is not possible to verify the answers from a self-selecting and self-reporting group of respondents. A larger sample may help to reinforce the results.

Another drawback with the experiment is that it was targeted at advanced web users and there is no proof or no evidence, to suggest that desktop searching is the same as web searching. The basic searching principles are transferrable, but the information needs of the person may change when trying to retrieve a file from their hard drive.

Summary

The investigation demonstrated that people submit tentative, non-exact queries which aim to retrieve matches close to the query string. People also prefer to limit the scope of the search to retrieve a higher concentration of relevant information.

2.3.4 Multi-Stage Query Formulation

Query formulation can be a difficult process for those with poorly formed information goals [Smeaton and Kelledy, 1998]. Careful query formulation can yield high quality results if the information goal or need of the person is well defined [Smeaton and Kelledy, 1998]. If the information goal of the person is unclear, then they may benefit from a more interactive or exploratory method of IR.

The idea of making IRSs more interactive can lead to 'multi stage query formulation' [Smeaton and Kelledy, 1998]. This is where the person can create an initial query and expand this query using search results and relevant terms. If the system uses a more exploratory method, the person will be able to browse through a list of relevant topics which they could then use to expand the initial query. This may be more effective for those with unclear information needs, however, as systems such as Google have

demonstrated, people may not be prepared to spend time formulating queries when they can enter simple search terms and browse a larger list of relevant information.

An experiment to discover the benefits of interactive query formulation was carried out on two people, one computer scientist with advanced search skills and a novice who had not even used a web based IRS [Smeaton and Kelledy, 1998]. They were both asked to construct queries using single words and phrases. The phrases were predefined and they had to select them from a list and drag them into their queries. The results showed that the computer scientist benefited from a combination of predefined phrases and their own words to produce accurate results. Conversely, the novice was drastically hampered when using multi stage, phrased queries. This indicates that people with more IRS experience have a better understanding of which search phrases must be used to yield accurate results. Intriguingly, the results showed that the single word queries used by the novice were just as effective as the phrase based queries used by the computer scientist. This suggests that novices benefit from IRS's such as Google which encourage the use of single word queries to find information. Research suggests that people favour 'the path of the least cognitive resistance' [Smeaton and Kelledy, 1998], which indicates that people prefer to use simple search terms and browse through results to retrieve information. This could be due to the perceived complexity of query formulation or the difficulty people have changing an information need into a succinct query.

Summary

The investigation above suggests that people spend little time formulating complex queries and prefer to use simple search terms to explore a list of results.

2.3.5 Information Clustering and Categorisation

[Hearst, 2006] suggests two main methods of categorising search results, clustering and faceted categorisation. Clustering is the grouping of search results based on similarity between results, for example, if the term 'Puma' was entered, the results could be categorised as 'Ford Puma' and 'Apple Puma'. Clustering is fully automated and can use implicit information to cluster information based on relevance matching. One of the main benefits of clustering is that it supports poorly formed information problems, as the clustering of results allows the user to view the dominant or most relevant themes relating to a query, rather than a list of grouped results. A major drawback with clustering is that it is dependent on the efficiency of the algorithm to ensure that results are effectively clustered. As a system cannot determine the semantic meaning of a query, it is difficult to group results in a way that is meaningful to the user. This is a common problem with information clustering as,

'usability results show that users do not like disorderly groupings, preferring understandable hierarchies in which categories are presented at uniform levels of granularity' [Pratt et al., 1999].

Faceted categories are usually manually defined categories which order information into meaningful groups. This differs from clustering as categorisation uses fixed categories to group information which is consistent with each search. For example, a faceted category would be a music category containing only music files and a cluster could group information relevant to an artist, genre, etc. The benefit of using faceted categories is that they are consistent and predictable with each use of the system.

Summary

Categorisation or clustering represents the principle of subdivision of an information patch into more meaningful segments. Clustering supports more exploratory search if implemented effectively. Categorisation provides a predictable structure for subdivision of the information space.

2.4 Information Foraging

Information foraging uses the metaphor of wild animals rooting for food to analyse how humans collect information online. This theoretical framework can be used to critique web design and improve user interaction [Card and Pirolli]. There are three major components of IF:

- Information scent - this can be information clues which lead to the final information destination. The information scent will strongly influence the information path a person takes
- Diet selection - what information is relevant and how the system provides this relevant information.
- Patch selection - where to search for information. The person decides which system or patch to use and must also decide when to change patches.

The above metaphors illustrate that IF has strong biological foundations and models the person's interaction with a biological entity within a constrained environment. This application of biological characteristics can help to produce a system which closely matches the needs of a person. The idea of information in the 'wild' is structured in patches and people must search these patches to harvest information. Information Foraging suggests that people must minimise the time spent searching between patches and maximise the time spent searching within each patch. This is the process of enrichment [Pirolli and Card, 1998]. The 'Optimal Foraging Theory' assesses the amount of time expended searching between patches. For instance, if a person is foraging for information in one patch, as the information diminishes, they will face a decision of whether to remain in that patch or move to a new patch to continue foraging.

The strong metaphor in IF provides lots of contributions of the IRS designer, but relies heavily on the assumption that people have a relatively structured idea about what they are looking for (food) within sets of information (patches).

Summary

IF introduces the principle of subdivision of information space into patches and the resulting successive exploration of these patches. Foraging is a forgetful process, as users will exhaust a patch and move onto another, using and discarding information from each patch, but never collecting or gathering information from patch to patch.

2.4.1 Collaborative Foraging

IF does not account for user collaboration within information tasks and assumes that people have well defined information problems. It is suggested that, for people who do not know exactly what they are looking for, keyword oriented search systems are less effective, therefore, collaborative foraging supports people with unclear information goals. Collaborative Foraging is an amalgamation of IF and collaborative filtering which models the human information foraging process on the co-operative processes of other creatures, such as bees [Schultze, 2002].

Collaborative filtering identifies information which may be relevant to one person based on the information needs of a group of people. Collaborative filtering uses networks to obtain explicit and implicit information to build an information network [Schultze, 2002]. Collaborative Foraging uses information gleaned from numerous foraging agents to shorten the 'between patch yield' and increase the information scent.

Web Waggle is a system which was developed to investigate the impact of IF and collaborative foraging [Schultze, 2002]. Web Waggle worked by people identifying information which may be relevant to other people and creating a collaborative network of shared information. Web Waggle was tested on a group of computer scientists, however, the aims and objectives of the experiment seem unclear, thus producing inconclusive results. Web Waggle seemed to rely heavily on the use of user derived explicit data. This could be a disadvantage because people have to put in more effort to retrieve information and this effort will not yield immediate goals; it will simply go towards an end goal of fortifying a collaborative network of information. The use of implicit data is a more favourable method and is used by many retail systems to offer recommendations based on buying patterns of others. The concept of collaborative information foraging is interesting because it provides a structure for information based on the visibility of retrieval patterns of other people. However, any system which relies on voluntary contributions from people, may not be effective due to inaccuracies and lack of participation. This is reinforced by [Marchionini, 2006] who states that people are often unwilling to 'provide feedback when the search is the classic turn-taking model'.

Summary

Collaborative foraging identifies the principle of using explicit or implicit shared information in a collaborative environment to increase the information scent and provide traces between information patches.

2.5 Information Browsing

2.5.1 Information maps

As stated in Section 2.4.1, search term queries are not as effective when the person has a vague understanding of the information problem they are trying to address. Browsing allows people to explore the information domain and capitalises upon the fact that recognition is more powerful than free recall to find an item. One major disadvantage of browsing large systems is that people can become conceptually lost. This 'lost in concept space' problem can be linked to the Internet and file system browsing when someone loses sense of where they are in an information space. This could be due to navigating through hypertext links or navigating through nested folders with little semantic meaning [Korn, 1996].

A method of improving the development of a person's cognitive map is to add visual clues, which help the user to map the information space [Korn, 1996]. The idea of maps in non computer related tasks is to help people plan routes or navigate large spaces to reach an end goal. This approach can be transferred into the IR process. Visual clues could take the form of colour co-ordination of items to portray the idea of landmarks [Korn, 1996]. Browsing is suitable for certain search tasks, for example, hierarchical subject browsing. If someone has an interest in cars, they may commence by searching for 'engines' and traverse through to related areas, such as, radiator, engine block then to cylinders and on to pistons. A method of improving conceptual, spatial awareness could be to indicate the number of nodes a parent node contains. This gives the sense of depth to the person [Korn, 1996].

2.5.2 Document Browsers

The underlying data structure of a file system is extremely important when designing document browsers. When a person traverses a conventional file system, each stage of the process requires a decision, i.e. which path should the person take. The person must base this decision either on recall or recognition. If the decision is based on recognition, then they need some way of identifying that route as the correct route [Korn, 1996]. There are problems with graphically visualising a large file system. Complex file systems mean that a multi-dimensional conceptual space must be mapped onto a two or three dimensional space [Korn, 1996]. Browsers such as Treemaps use full screen space and show structure of the file system as well as text based information and visual clues [Korn, 1996]. This improves information scent as details of each node are displayed. The disadvantage with using such interactive browsers is that they can be disorientating and there is a possibility of information overload.

Hyperbolic Tree Browsers

Pirolli discusses the use of hyperbolic tree (Figure 2.2) browsers to improve IR and foraging techniques. Hyperbolic tree browsers display information using a 'focus and

context visualisation' [Pirolli et al., 2001]. The root of the tree is presented in the centre of the visualisation and then child nodes are arranged thematically from this. Experiments were carried out which identified that hyperbolic browsers allow users to find information more quickly than traditional browsers [Pirolli et al., 2001]. However, for hyperbolic browsers to be effective, the information scent must be high. When information scent is low, hyperbolic browsers are slightly more effective at finding information, but in both cases, more nodes are selected to retrieve information than in generic browsers [Pirolli et al., 2001]. This demonstrates the importance of information scent with respect to IR. IRS's such as Google, improve information scent by adding textual descriptions below each link. Hyperbolic browsers also help to counteract the 'lost in conceptual space' theory as users are presented with an overview of the file system or information field.

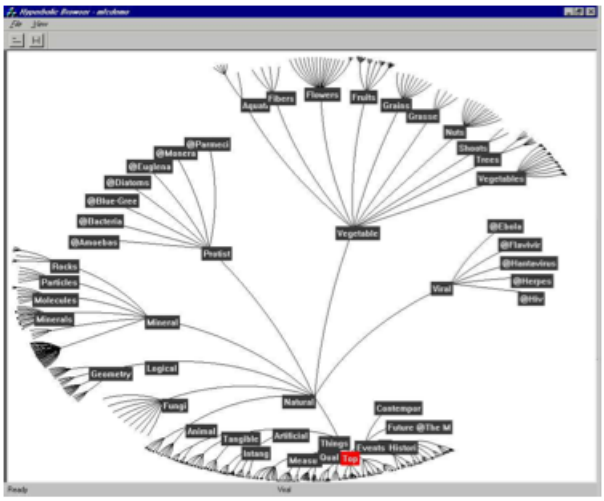

Figure 2.2: Hyperbolic Tree Browser

Goldleaf Browser

'One of the greatest challenges of UID is to simultaneously provide the user with context as well as detail' [Faichney and Gonzalez, 2001].

Goldleaf (Figure 2.3) is a document browser developed by Faichney and Gonzalez. Goldleaf differs from traditional browsers by visualising six sub folders of a tree hierarchy in a radial fashion around the screen [Faichney and Gonzalez, 2001]. Each main sub folder also displays six child folders. One of the major features of Goldleaf is the ability to use large thumbnails to visualise the file content. People can quickly identify the content of each file on the screen and if they cannot locate a file they require, they navigate through to a sub folder using a single mouse click. Faichney and Gonzalez carried out experiments on five computer scientists to discover the usability of the Goldleaf browser. They performed a number of search tasks and asked each user to locate a specific document using Goldleaf and then Microsoft Windows Explorer. They found that

the number of clicks required to find the document was far smaller using the Goldleaf browser. This reduced number of clicks could be attributed to the way Goldleaf uses the entire screen space to display three tiers of the file system [Faichney and Gonzalez, 2001]. Windows Explorer only shows one level of folders and therefore when a path branches off a person must take more time making a decision on which path to take, or use more clicks by traversing up and down the file system [Faichney and Gonzalez, 2001].

There are constraints with Goldleaf, it cannot display large numbers of sub-folders efficiently. The size of each folder and document thumbnail decreases the further down the tree it is. The modern emphasis on multimedia technology and the exponential growth in the digital music market now means that users will have many sub-folders for music and video. Goldleaf would not be able to display all the sub-folders of a user's music directory if they had, for example, one hundred artists.

The test also indicates the importance of document details and preview for IR, in the case of Goldleaf, document thumbnails proved to be vital in the user's recognition process during file selection. As with all browsing and search tasks in the experimental environment, people are always told what they need to locate. In practise, this is not the case as many people have an information goal and no clear document in mind.

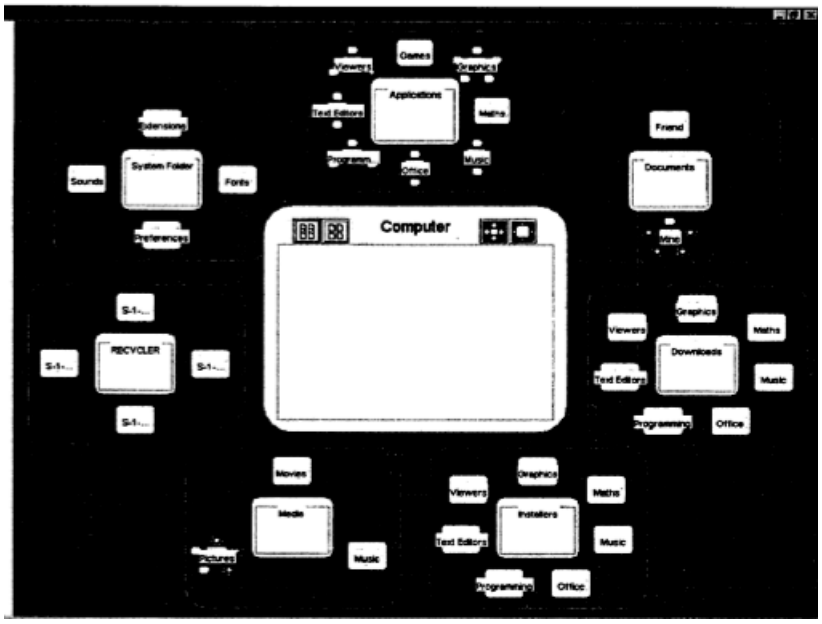

Figure 2.3: Goldleaf Browser

Summary

Information browsing identifies the principle of information exploration and recognisable information structure. The use of visual clues and appropriate file information can

improve the recognition of location within an information structure and improve the information scent.

2.6 Information Visualisation and Re-access

The study by Aula, Jhaveri and Kaki, introduced in section 2.3.3 identified that re-access of information was an important feature of the search practises of web users. For example, a person may formulate and execute a query to find information and then need to find the same information at a later date. The results showed that people find it very difficult to remember the exact query entered to return the same results. The use of browser history was identified as ineffectual because history items often have page titles with little semantic meaning. The use of bookmarks is difficult because users tended to have such a large number, that it was difficult to locate a particular item.

'Findex and Session Highlights' [Aula et al., 2005] are two systems, which aimed to solve the problems raised above. Session Highlights took a visual approach to the problem by allowing people to save their queries to a large clipboard. Bookmarks were saved as large thumbnails because 'people remember visual information well and can easily use visual clues such as thumbnails' [Aula et al., 2005]. Another interesting point raised is that 'recognising is typically much easier than recalling' [Aula et al., 2005]. This suggests that people are more likely to locate a particular item by acknowledging the thumbnail as opposed to remembering the location of the item itself. Findex used Google search results and represented these results in a different format. It used a clustering technique to organise the information into groups as stated above.

Summary

Aula et al identified the importance of query re-access and recognition over recall. The user needs a method of storing previously created queries or links to information, whilst ensuring that these saved objects provide enough semantic meaning to be re-used at a later date. Visual clues such as file previews and labels must be used to aid the recognition process.

2.7 Information Gathering

Information gathering is a term which is loosely used in computer science to refer to the act of gathering or collecting information. IG has no definitive meaning, in contrast, IF and information browsing are defined domains of exploratory search with relatively stable definitions. IG is most commonly discussed as a function of a system which retrieves and gathers information in an automated fashion.

[Schwartz and Pu, 1994] developed a tool, Netfind, which locates email addresses and additional information about internet users. They applied an IG architecture to Netfind

to gather this information from a widely distributed network. Below is a framework they developed to detail the stages of IG.

1. Examine previously gathered information. This stage involves examining previously gathered data and applying rules to decide which sources should be used for the next IG activity.
2. Collect information from a selected information source. This is the actual act of gathering information, which is usually an automated process determined by a set of rules
3. Apply source specific data to eliminate unusable information. This involves filtering invalid information based on given criteria
4. Combine records from multiple sources. This stage combines information gathered from multiple sources to solve heterogeneity problems, for example, differences in file formats for gathered data.

The framework adopts an algorithmic approach to gathering data based on rules and criteria, which demonstrates the automated, system oriented nature of IG. As there is no framework which details the user's IG process, user studies must be undertaken to investigate the human IG process.

2.8 Conclusion

This chapter examined the research literature on exploratory search to determine a set of good design principles for a user centred IR and IG system. These principles are detailed below.

2.8.1 IR and IG Design Principles

1. Query Formulation. Users submit non-exact, tentative queries to return information relevant to the query rather than exact matches. Users also submit simple queries to produce quick results and browse through the results, often redefining queries. Users also limit the search to locations of possible relevance, so that the results returned contain fewer irrelevant results. The system must support the formulation and re-formulation of non-exact queries to return relevant information which the user can browse through. The scope of the query must also be limited to reduce the amount of unnecessarily irrelevant information returned.
2. Query Information Visualisation. Users utilise file previews and specific file attributes, such as name and location, to make a judgement on the relevance of information. The visualisation of information must concentrate on supporting the user's recognition process and not rely on recollection.
3. Query Information Clustering and Categorisation. Users need a method of dividing patches into more manageable sizes. Clustering or faceted categorisation must

be used to divide information patches into more meaningful segments.

4. Information Browsing. Information browsing involves navigating through an information space, looking for relevant information whilst creating a mental model of that space. Visual clues, such as document thumbnails and file attributes must be used to provide the user with sufficient feedback on their location within the information space.

5. Information Foraging. Users move from patch to patch exhausting information in each patch before moving on to the next. The system must support this patch exploration process.

6. Collaborative Foraging. In a networked environment, users explicitly and implicitly share information to build a collaborative network of relevant information, to increase information scent and reduce time expended between patches.

7. Information Re-access. Users frequently need to re-access previously executed queries to find relevant information. Users must be able to save and attach meaning to these queries to aid the recognition process when attempting to re-access saved queries.

8. Information Gathering. Users need to be able to collect information during patch searches to counteract the effect of 'forgetful foraging'. Users must also be able to gather information during IR and information browsing and attach meaning to this information to aid the recognition process when attempting to retrieve previously gathered information.

In the next chapter, these principles will be re-examined and refined through user studies.

Chapter 3

Studying Usage of IR and IG

3.1 Introduction

The research into exploratory search conceptualisation in Chapter 2 provided a set of principles which define the interaction with an IR and IG system. To produce a full requirements specification, analysis of Spotlight and empirical studies to determine how people search on the Macintosh platform, were undertaken. This chapter introduces the Macintosh platform and describes the two user studies which were carried out to validate the principles identified in Chapter 2.

3.2 Searching and Finding Information on The Macintosh Platform

One of the project aims is 'to create a native Mac OS 10.4 information retrieval and gathering prototype', however, there is no justification or explanation regarding this decision. The Macintosh platform was chosen to develop on for three main reasons.

- From a developmental perspective the development tools available as standard on the Macintosh platform are ideal for supporting rapid application development. These tools are discussed in section 5.3.1.
- From a learning and personal development perspective, the final year project provides a massive learning opportunity for students and for this reason, it provides a great platform to investigate different operating systems and development techniques. The Macintosh platform has always been of personal interest and this project provides the basis for such investigation.
- Arguably the most important factor is the search technology, Spotlight, which has been integrated into the latest version of Mac OS X. Spotlight, as described below, allows the user to search file content and metadata of all file stored on the file

system. No other mainstream operating system provides this functionality which is integrated system wide.

The importance of Spotlight technology is not only that it is a new and interesting platform for user research, but as it is integrated into the file system, developers are able to work with the technology to incorporate it into their applications. Before Spotlight is discussed, it is worth mentioning the role of the Finder application in the OS. From the perspective of the user, Finder is a file system browser which is similar in function to more common file system browsers such as 'Windows Explorer'. Finder is the default application which manages all files in the file system, for example, Finder manages common file tasks such as creation and deletion of files.

3.2.1 Spotlight Technology

Spotlight appears in four forms:

- Standard menu bar implementation. In the standard Spotlight implementation, the user selects the magnifying glass icon in the top right hand corner of the screen and a search field is displayed. The user types in a search string and a results list appears, populating as the user continues to type. The results are represented in a list format with different categories to represent the file type returned. Figure 3.1 displays the standard Spotlight search bar.

Figure 3.1: Standard Menu Bar Implementation

- Advanced floating view. This is an extension of the standard menu bar implementation which is displayed when the user selects the 'see all' results button in the standard menu view. This view is not owned by any application and is simply an extension to the standard implementation which allows the user to sort the results according to date, kind etc.
- Smart Folder View. Smart folders are an extension of Spotlight as they are virtual folders which use a user defined query to create dynamic folders. Smart folders are created using the Finder-Spotlight search field, where the user enters their query and the Finder window is then populated with search results.
- Command Line. As Spotlight is integrated system wide, searches can be executed from a UNIX terminal by using the `mdfind` (metadata find) command followed by a string.

High Level Overview

Figure 3.2 illustrates the relationship between the main components of the Spotlight implementation.

When a file is modified, created or deleted, the kernel notifies the Spotlight engine that a change has taken place to a file. It is the then job of the Spotlight engine to update the system store with the new metadata attributes of a file, these include, filename, size, date created. It does this by using Launch Services to determine the file type and use an appropriate metadata importer to import the metadata attributes of the file and construct a dictionary. This dictionary is then passed back to the Spotlight engine which updates the system store.

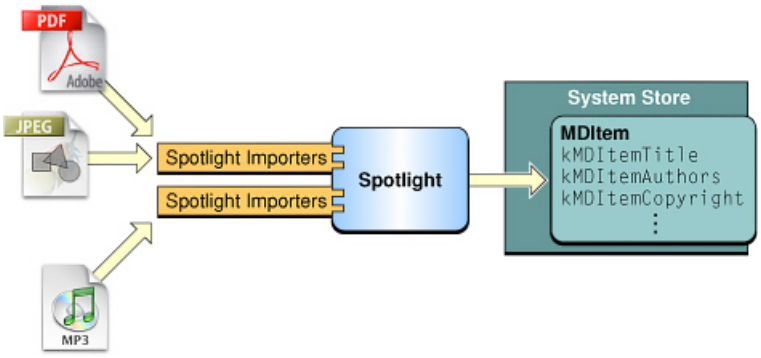

Figure 3.2: Spotlight Overview

This demonstrates the basic method of indexing metadata of a file, but it does not take into account the user's interaction when searching. When a user enters a query into the spotlight window, the application owner creates the query expression and executes the query. This query is passed to the spotlight engine which searches the system store to match the metadata attributes and return the results to the application. This is only a high level overview of how Spotlight works to provide a basic understanding of the technology. Section 5.4 contains a more detailed analysis of exactly how Spotlight works in relation to system design.

3.3 Requirements Sources

Figure 3.3 illustrates the sources which were used to create the preliminary requirements specification in appendix A.2. The literature review provided a list of principles which define the users interaction with the system. These principles characterise the main features of an IRS and IG system, however, they do not provide enough information to create a user centred requirements specification.

As the final prototype will be a native Mac OS X 10.4 application, the audience for the prototype is effectively all Mac OS X 10.4 users. Empirical studies with a sample of Mac OS X 10.4 users provided the main source of data for the requirements specification.

The empirical studies were divided into two sections, Spotlight analysis and co-operative evaluations analysing how people address specific information problems. The studies are discussed below.

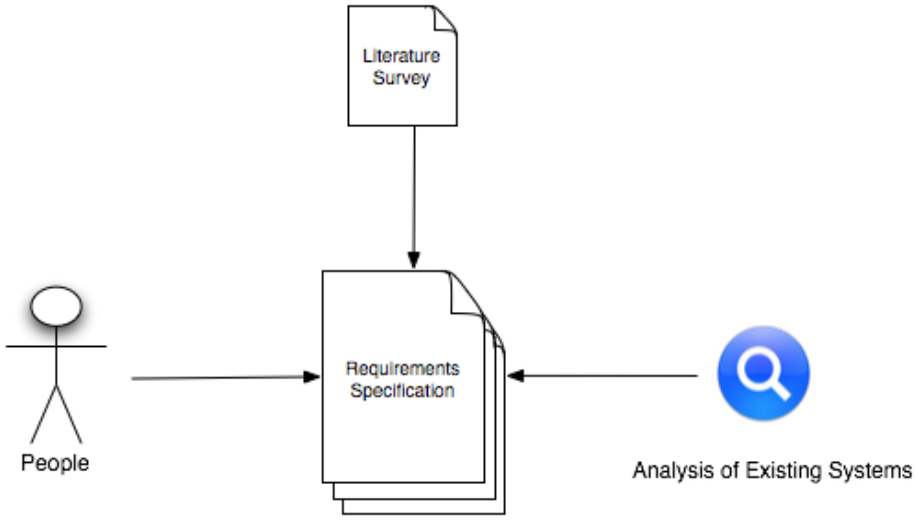

Figure 3.3: Requirements Sources

3.4 Study 1: A Think-aloud Study of Spotlight Usage

As Spotlight is still a relatively new technology, there are obvious constraints with the system. The high level overview of Spotlight in section 3.2.1 described how the system functions, but did not discuss any constraints with the technology. To identify the constraints and benefits of Spotlight, a series of structured interviews were carried out with 10 Mac OS X 10.4 users. The term interview should be used loosely as it was a think-aloud evaluation style interview, where the participant discussed their opinions of Spotlight whilst running through typical activities on their computers. Participants were briefed on the aims of the study and the kinds of activity they would be performing. They were informed that they could leave at any time and asked to consent to participate if they were happy to do so on these terms.

The aims of the informal interviews were as follows:

- To identify the major constraints with Spotlight
- To identify the main strengths with Spotlight
- To identify areas of improvement with Spotlight technology
- To identify the common tasks users perform with search results

The interviews were kept as informal as possible. This decision was made to help the participant feel more relaxed and more comfortable offering their opinions. The interviews were conducted in informal environments, such as, a group study area, a lecture room or the participant's house. The interviewer adopted an informal approach, working through typical Spotlight scenarios with the user to encourage the participant

to offer their opinions. Where possible, the participant was asked to bring their own machine to use as a collaboration and memory jogging device. It was not as easy for a user to think hypothetically about constraints with Spotlight without actually using the Spotlight, therefore using their own machine acted as a memory prompt and an 'icebreaker' between the interviewer and interviewee. The interviews were not recorded, however, the interviewer took notes throughout the interviews. Where 1:1 interviews were not possible, remote interviews were used by holding Audio conferences over iChat AV.

The key results were taken from the interview notes and placed into the table found in Appendix A.1. The table contains the participants quote, the meaning of the quote, the domain the quote relates to and a suggested requirement of the system. This structure was adopted so that the quote and the activity associated with that comment can be linked directly to the domain of interest and a possible requirement. This aids the requirements validation process as the requirements can be directly linked back to the participants comments and activities.

3.4.1 Results

The results show that most of the constraints with Spotlight are focused around the visualisation and categorisation of results. The results grouping does not match the users expectation of what type of file they expect to see in each category and the inconsistency between categorisation across applications has a confusing effect on users. During the interviews, the participants were asked what file types they expected to find in each category, the results can be found in table A.1

The main benefit of Spotlight, as identified by the participants, was the speed of result retrieval without much effort on the part of the user. For example, the user does not have to remember the name of the document, just something about that document and Spotlight will return relevant results. This reinforces the query formulation design principle from Section 2.8.1. During the interviews, users were also asked what actions they typically performed on the search results. The most common actions were:

- Open the file. This was the most common action performed on search results. Opening the file was performed for two main reasons, to retrieve information from the file itself and to verify its relevance.
- Go to the location of the file, also know as 'reveal in Finder'. This was performed when users needed to access the file itself, for tasks such as, file transfer, or accessing a particular file location to view similar items

3.5 Study 2: Co-operative Evaluations

The main content of the requirements specification was derived from end user evaluations to determine how people address specific information problems. To gain a full

and accurate requirements specification, it is not acceptable to leave the all of the requirements and design decisions to the developer. Users need to be involved throughout the process to ensure that the requirements and the final system meet user expectations.

3.5.1 D E C I D E Framework

To plan the user evaluations, the D E C I D E Framework [Basili and Rombach, 1994] was adopted. This framework was chosen because it provides a clear structure to the evaluation and as Basili states 'well planned evaluations are driven by clear goals and appropriate questions'. The main disadvantage with this approach is the uneven weighting for different sections of the framework. There is no stage to specifically design the evaluation itself, therefore, all of the evaluation planning and technique evaluation took place in the third step, 'Choice of evaluation paradigm and techniques'. The D E C I D E framework is an acronym for the following six stages.

Determine Goals

High level goals of the evaluation:

- To investigate how people address a variety of information problems
- To identify and analyse how the purpose of the information problem affects the method adopted
- To investigate if people display information gathering tendencies

  If people do gather information, how do they do this?
- To identify additional practical constraints with Spotlight
- To produce a full and accurate requirements specification to support information gathering.
- To validate the design principles in Section 2.8.1.

Explore Questions

In order to achieve the goals of the evaluation, specific questions were asked which intended to guide the design of the evaluation and the analysis of the results.

- What is the participant's individual interpretation of the problem?
- What technique does the participant adopt to solve their problem?
- What system does the participant use?
- Is the process recursive?

  If so, how many stages does the participant adopt?

What leads the participant to feel they are not satisfied with the results and to continue the search?

What leads them to feel satisfied with the results?

- Does the participant exhibit gathering tendencies?

  How does the user gather relevant information?

  What does the participant do with the relevant information?

  How does the user discard the irrelevant information?

- What does the person do with the information once it has been retrieved?
- What does the participant identify as constraints with Spotlight?
- What does the participant identify as the benefits with Spotlight?
- Does the lack of familiarity with a system influence the methods of information gathering?

Choice of Evaluation Paradigm and Techniques

Questionnaires were not used at any point during the requirements gathering process as they offered little benefit to the overall investigation. Questionnaires are ideal for collecting a large amount of data from a large source of people. However, to design a successful questionnaire, the majority of the questions must have a finite set of answers, which in some cases will lead the participants answers. It is important when conducting user studies, not to lead the participant into answers that suit the developer. Questionnaires are most useful when people have a conscious opinion or definite answer to a question, the nature of this investigation is such that users are often not aware of processes and steps they take when retrieving and gathering information. The questionnaire would return very little useful data and would results in unnecessary time consumption.

Interviews were appropriate for the investigation in section 3.4 it was purposely exploratory. However, they were not appropriate to identify activity patterns during IR and IG processes. Specific user activities needed to be identified and this information is only obtained through observation and not question and answers sessions. Even the most self-reflecting people would not be able to identify the particular stages they follow when gathering information.

Experimental design was not chosen for requirements gathering as it does not support the aims of the evaluations. Experimental design is ideal for comparative user interface testing, where data collected compares the usability of different systems under different conditions. The evaluation style in this investigation places emphasis on observing and analysing humans performing their natural processes. There are no independent variables which can be manipulated to provide meaningful data. It is important to analyse how and why people perform particular actions and how these can be supported, not what affect manipulating a particular variable has on the person.

The analysis of experimental design methods indicated that a more observational approach was appropriate to support the aims of the investigation. In an ideal scenario, ethnographic studies would have been adopted to analyse users in their natural environment. This would enable the evaluator to spend time in the field with the end users, analysing exactly how they address their information problems in their natural environment. These studies would have to take place over a long period of time with many different users, therefore they are clearly not applicable for an eight month project.

Therefore, the decision was made to conduct a formal evaluation in an open-ended interactive style. A variant of the Think-aloud technique, Co-operative Evaluation, allows the evaluator to collect large amounts of data about what the participant is thinking, whilst working with the participant to produce a more informal, comfortable atmosphere.

The Think-aloud technique is intended to ascertain what the user is thinking whilst they are using a system. Most common uses of Think-aloud involve the user running through a series of tasks whilst verbalising their thoughts. Think-aloud is extremely effective if evaluations are recorded as there will be large amounts of data to analyse. One of the constraints with Think-aloud in the investigations was that it was hard to keep the participant talking and occasionally, the self reflection would cause the participant to perform actions they may not have performed otherwise. This constraint with Think-aloud was counteracted by allowing the evaluator to adopt an interactive, co-operative style, so that the participant always felt at ease.
Co-operative evaluation is a variant of Think-aloud where the evaluator works with the participant to solve particular tasks. The evaluator is able to ask the participant questions about the system they are using and discover the user's opinions.

Participants

10 students from the University of Bath between ages 20-23 were used in the evaluations. The evaluations took place in a specially equipped HCI laboratory at the University of Bath. Each participant was a Mac OS X 10.4 user.

Procedure

The co-operative evaluations involved a set of problem-tasks which the participant and the evaluator worked through until the task was completed. Each participant worked through 7 tasks specifically designed to answers the questions raised above. Each session lasted approximately 30 minutes and screen outputs and audio were recorded for post evaluation analysis. All problems were lead by the participant and they were encouraged to address the tasks in their own way. It was the participant's decision to identify when they felt they had completed the task and attempt to explain why they thought that was an appropriate end point.

The evaluator acted as a prompt for the participant to articulate their thoughts and offer help on how to use particular systems. It was important that the interactor did

not clarify problems set because it was the participants decision on how to address the information problem that was being analysed. The interactor was able to ask questions such as 'what do you think about this interface?' or, 'what do you think about this results view?'. It was important to ensure that any questions did not lead the user. During the evaluation, the evaluator concentrated more on collaborating with the user than note taking.

The evaluations were initially tested with a test participant to practise the tasks and suggest any improvements. The trial run identified a few key areas of improvement in the evaluation, such as, techniques to encourage participants to keep articulating their thoughts, appropriate questions to ask and restructuring of data content on the test machine.

Materials

The test machine in use was a 15 inch 1.25ghz Powerbook, running Mac OS X 10.4, connected to an external display, keyboard and mouse. The machine was configured with a new user account and the hard drive was populated with information relating to the problems set. Participants had access to the internet and any DTP packages they required. The test machine was configured with a large amount of source data taken from various online sources and additional miscellaneous data from the interactors machine.

Identify practical issues

A major design issue with the evaluations was attempting to re-create the participant's natural information environment. As stated before, it was not possible to go to the participant's environment to spend time with them and record their activities. The evaluation method, coupled with the scenario based problem-tasks tried to emulate a typical information problem in an informal setting and the results demonstrate that it was a successful technique.

Participant availability and time constraints were major practical issues to overcome. There was difficulty finding a large enough Mac user base for evaluations and then finding users who were willing to participate. A total of 10 Mac users were recruited for the evaluations, this was a large enough user base to extract meaningful data, but a larger group would have yielded a wider range of results

Equipment and storage devices were major practical issues. The HCI laboratory is shared between numerous students, so there were major time constraints on using the equipment. Audio-visual equipment in the laboratories was excellent, however, data transfer from DV tapes to storage devices proved to be a problem as the cables used to transfer the data were shared between numerous students.

Storage space was an unconsidered practical issue. Each 30 minute session equated to approximately 5GB storage space and each miniDV tape only stored 60mins of film.

This turned out to be a fortunate constraint as it meant that analysis of each session had to be carried out soon after the evaluation so that the data could be overwritten.

Decide how to deal with ethical issues

It was stressed at the beginning of each task that it was a very informal session and participants were free to ask questions and most importantly, every answer was the right answer. Participants were given an extremely high level objective of the evaluation, 'investigation into searching'. The detailed objectives of the experiment were not communicated to the participants to ensure that actions remained impartial. Participants were free to stop the experiment at any time and were offered anonymity in the video and audio recordings. All participants signed a consent form before each evaluation to inform them that no personal information would be used. Below the text content of the consent form

By signing this consent form, I agree to take part in a controlled research experiment to investigate user-centered search systems. I agree that any private information has been legally obtained and used for appropriate purposes in a controlled environment. I acknowledge that my actions will be recorded and analysed for research purposes.

Evaluate, interpret and present data

It was important during the analysis of the data to separate the participants' verbal reasoning between solving the problem and the use of the tool. Usability problems with the system had to be separated from problems participants experienced when trying to address their information problem. This division was important because it allows the developer to identify the difference between usability problems to address with a system and the conceptual problems of IR and IG. Usability problems with Spotlight can be addressed as additional features of the system, which do not necessarily relate to the concept of IR and IG.

The analysis of how participants interpret the task was important because the way people express their information need or problem relates directly to the task. For example, typing specific keywords into a search engine effectively defines the problem they are trying to address.

Statistical data was used to evaluate the rate of occurrence of a particular activity, however this was used to reinforce key concepts and not to be interpreted in a statistical manner.

Before the analysis of each session could begin, the video data needed to be re-watched to transcribe all of the key activities and quotes. When all of the data had been transcribed, the key activities and quotes were analysed in conjunction with any evaluator notes to ascertain activity patterns exhibited by the participants. These activity patterns were summarised and the full requirement specification in Appendix A.1 was produced. Video data are a very rich resource for an investigation, admitting many possible analyses.

They are typically used for extensive evaluation of the content and process of an activity, not for assessing its outcome.

A common problem with design decisions made in projects is that requirements often seem to 'appear' without any relation to the context of the problem they are trying to solve. This often leads to questions such as 'why did you design this feature?' or 'why is this feature necessary?'. Therefore, to attempt to counteract this problem, each requirement is accompanied with a description and the principle or source from which that requirement was derived . The data from the evaluations was originally placed into a table to analyse the key activity patterns and identify possible requirements. However, as the amount of data obtained was so large, this was not practical, therefore, the results were summarised in Section 3.6

3.5.2 Problem-Task Analysis

The table below lists the problem tasks participants completed during the evaluations. Each task is accompanied with its respective domain and additional notes. Tasks 1-4 adopt a scenario based approach to try give the participant an objective for the information problem. By accompanying each problem with a background scenario, the participants were able to understand the relevance of the task and interpret it in their own way. Tasks 5-7 adopt a more structured task based approach and do not include problem scenarios.

Problem 1	
Scenario	You have a cockroach infestation in your home. There are holes in the walls, cockroaches scuttling across the kitchen floor, and bugs in your bath tub. You do not know what to do to solve your cockroach problem.
Task	Find information which will help you solve your cockroach problem
Domain	IF, IR, IG, Browsing
Notes	Poorly defined information problem. Most open task, note how user approaches first open task. Recursive task, no clear stopping point, monitor how users identify relevant information

Problem 2	
Scenario	You are writing a document about mice and remember that you have a few documents saved somewhere on your computer, which contain information about house mice, but you arent sure how relevant they are.
Task	Find information about mice to compile your document
Domain	Suggested IR, IG, IF, Browsing
Notes	Poorly defined information goal, suggesting there is something useful on computer, 'mice probably keyword, hinting towards document retrieval. Will the participant store the results as they search? Do they bookmark / save searches? Still recursive, with less clear stopping point, as how much information do they need?

Problem 3	
Scenario	You have just bought a new car and the bumper has fallen off. You are not allowed to drive with your bumper hanging off so you need to find a way to fix it. You do not have Internet access for this task.

Task	Without using the Internet, find any information which may help you solve your bumper problem.
Domain	IR, IG and browsing for bumper related material. Data retrieval for repair services etc
Notes	More focussed task. Forces participant to use desktop. Less likely to gather information, as could be a quick answer. Possible data retrieval for number for mechanic

Problem 4

Scenario	A friend has recently become interested in 'badger watching and asks you for any information or tips about badgers in general. You are sure that you have a document on your computer specifically about badgers, but cannot remember any details about it. You are sure that it was created in 2006 because you became interested in badgers on your holiday to Bognor Regis.
Task	Find any badger related information to give to your friend.
Domain	Query Formulation, IG, IR, IF and Browsing
Notes	More structure, encourages participants to search for Bognor or 2006. General search for a known document should encourage gathering of information. Search within different patches, how do they move from one patch to another?

Problem 5

Scenario	N/A
Task	What is a major cause of deforestation in the Amazon Rainforest?
Domain	IG and data retrieval
Notes	Ideal task for IG and data retrieval, encourages question and answer based searching, gathering of information during task

Problem 6	
Scenario	N/A
Task	Who was the second president of the United States?
Domain	Data retrieval
Notes	Less recursive than all previous task, likely no IG, simple data retrieval

Problem 7	
Scenario	N/A
Task	Find ONLY IMAGES AND PDF files on your computer relating to cheese
Domain	QF
Notes	Find out if people know how to create advanced queries and their relevance. Searching for file container, not file itself

Tasks 1 and 2 are very non linear and were aimed at assessing the key areas of IR and IG. The problems had no clear stopping point and were designed to encourage the participants to gather information, reformulate queries and save searches. Tasks 3 and 4 became more focussed and moved towards data retrieval and gathering. This was because, as the tasks became more focussed, the problem structure became more question and answer based. It was important to assess whether information gathering exists in the domain of data retrieval in addition to IR. Task 4 focussed on advanced query formulation and how participants addressed information problems with multiple criterion. Task 5 was aimed at discovering how participants gather information when they have a direct question with many possible answers. Task 6 was a direct data retrieval exercise and assessed whether participants gather information when they address a less recursive problem. Task 7 was a unique question as it was an extremely complex task which focussed on assessing how participants create advanced queries using Spotlight. The aim was to analyse how participants use advanced features of the system and whether they offer any benefits to the IR and IG process.

3.6 Evaluation Results

The key results for evaluations are summarised and detailed below, to provide the basis for the requirements specification.

Problem-Task 1

Most participants chose to use Google to search for information about cockroaches. Queries varied from a generalised 'cockroaches' where the participant wanted 'information on cockroaches' to specific query strings such as "cockroach" + "extermination" + "bath". The participants who did not use Google, went for a more direct problem solving route by searching 'Yell.com' for 'pest control' and then specifying a postal address. Participants who adopted a data retrieval approach, i.e. looking for a specific number to call, stated that they would 'write the number down', or 'bookmark' the page. This demonstrates that some participants would either gather the specific information or, gather a link to that information. In both cases, the participants required to re-access the information at a later date. This verifies the information re-access and gathering principles from Section 2.8.1.

All of the Google searches were recursive, none of the participants stopped after one search, but as they searched, they identified possible relevant sources of information. Participants with experience in using the Safari browser, opened multiple tabs to keep relevant information open. Rather than gathering small amounts of information as the participant moved through the task, they identified pages which contained information to return to. Participants placed more importance on remembering the location or source of information, than exact information from within that source.

Most participants were satisfied with the conclusion of the task when they felt they had found one or more sources which identified 'how to get rid of cockroaches'. One participant halted the task prematurely when they realised that they already knew how to get rid of cockroaches. Participants identified that knowledge of the location of relevant information was a suitable ending point to the task as they had effectively created a mental model of where they would find the information at a later date to continue the task. This can be linked to information browsing, gathering and re-access principles as the participants attempt to remember the locations of the relevant files to build up their knowledge of the information space and to re-use this information in the future.

Problem-Task 2

Nearly all of the participants interpreted the problem to be 'looking for information about mice'. Most participants chose to use Spotlight to search the test computer for information about mice, as the problem hinted at the existence of relevant documents on the machine. All of the participants who used Spotlight, entered 'mice' as the search term. Several users identified that they just want to 'pull up all information about mice'. This can be linked to the query formulation principle in Section 2.8.1 as users submit a simple query and browse through the results.

When the search list populated, most participants chose the top hit in the list 'task document', regardless of the file type of filename. When the top hit was not a relevant file, as it was invariably the actual task document, participants returned to the populated

results list and moved straight to the pictures or PDF category. This demonstrates one of the constraints with the clustering principle in Section 2.8.1, as results clustered depending on their 'relevance' are not necessarily the most relevant results. Not one participant used the 'Document' category, simply because it contained header files and non relevant information.

The PDF files listed did not have relevant file names, however, participants still selected these. Several participants displayed gathering tendencies whilst searching for information about 'mice'. They would open one PDF in the list, read the document and minimise the document if it was deemed 'relevant'. They would then return to the results list and try the next file in the list. This activity was repeated until they had opened all of the files in the results view, or they had 'found enough relevant information' to be satisfied with the end of the task. This demonstrates the principle of IF from Section 2.8.1 as participants exhaustively searched an information patch for food before ending the task or moving onto another patch.

One participant hovered over a PDF document in the results list and found the location of a PDF containing information about mice and ended their search. They explained that as they had found the location for one relevant piece of information, they assumed the location would yield more relevant results.

Problem-Task 3

This task proved to be more troublesome for most participants. The problem prevented participants from using the Internet as a resource and possibly did not encourage participants to search their computer for information, as several mentioned that they would not use a computer for this task.

The participants who did conduct a search, all typed 'bumper' into Spotlight. As a point of interest, three participants, even those who use Spotlight regularly pressed 'return' to activate the search, even though Spotlight searches on each key stroke.

For this task, participants navigated straight to the PDF category in the results view. Two participants interpreted the filename 'bmpworkshop' to mean 'bumper workshop' and opened that file. Several participants identified the auto-search feature of PDFs to be a distraction. If Spotlight returns a PDF file, the auto-search feature automatically searches through the open PDF for that query and identifies the matches in the PDF drawer view. Several participants identified this feature as a constraint when trying to browse through a file for information as the navigation view is obscured with query results as opposed to page thumbnails. The auto-search feature is ideal for data retrieval as it locates the relevant item instantly. Conversely, when used in conjunction with IR and IG, it is an obstruction for users as they wish to browse information which is relevant to the query, not necessarily the query term itself. This can be linked to query formulation and information browsing principles in Section 2.8.1, as participants constructed tentative queries and wished to browse relevant information, not the exact match.

Many of the participants in this task, stopped if the first search did not yield relevant results. They felt that they had satisfied their information need (almost proved to themselves that they did not have any information on their computer) and resigned to calling a mechanic out.

The participants who did continue, searched thoroughly for repair tips, reformulating the query with terms such as 'car' or 'car repair'. Participants collected bumper repair information as they continued to search by leaving open relevant documents and closing non-relevant documents. One participant decided to look up the location of a relevant file in the Spotlight results list and navigate to that directory to see if there was anything else of interest there. When they did not find anything in that directory, they browsed through the 'Documents' folder, looking for something related to cars. This reinforces the information browsing principle and the use of location attributes in results visualisation to provide users with a direct path to the location of the result.

One user finished browsing for information and wanted to return to a document they identified as relevant, however they had previously quit the application and could not remember what the filename was. Therefore, they recompiled the search and checked each file until they recognised the relevant one. This reinforces the principle of query information visualisation and in Section 2.8.1 as participants could not remember the name of the file and were not given enough information to be able to recognise it. It also demonstrates the principle of IF as the participant exhaustively searched the results list until they located the correct result.

Problem-Task 4

All participants began this problem by using Spotlight with the aim of locating 'a file' or 'any information on badgers'. All participants used either 'badger' or 'badgers' as their initial queries. This mainly returned pictures, which participants opened, but did not use. Again, participants moved straight to the PDF category in Spotlight view because they had become accustomed to irrelevant information being in the documents category. Most users did not find any relevant information after opening the first or second item in the list and resorted to other methods.

Some participants noticed the 'Bognor 2006' information in the task and tried to construct a more complex query and assumed that the advanced Spotlight view would be ideal for this. With the exception of one participant who tried to use advanced view to order results by date, participants returned to the Spotlight menu bar view when they decided that they could not construct a complex query in the advanced view. Only one participant entered 'bognor 2006' which returned one file containing information about badgers. Participants who could not find any information using Spotlight, used Google and searched for 'badgers' or 'badger'.

Participants stopped their search when they decided that there was no information about badgers on the computer because Spotlight did not return it. Participants who used Google, stopped when they found 'reputable' sites such as 'badgers.org'.

Problem-Task 5

All participants used Google to search for information on 'deforestation'. Queries varied from the open terms such as, 'deforestation' to the more specific, 'deforestation Amazon'. Participants typically used the first three results in Google to continue their searches. Participants who had entered general queries such as 'deforestation', reformulated their queries to make it more specific, for example 'Amazonian deforestation'. This reinforces the query formulation principle in Section 2.8.1 as participants initially submitted a tentative query, browsed through the results and reformulated the original query.

Participants exhibited gathering tendencies by searching multiple sources and identifying relevant items of information to use to solve the problem. Most participants identified a major cause of deforestation and used another web-site as verification of what they had originally discovered. Several participants searched within the web-page to narrow down the search even more. For example, one participant searched for 'causes' within a web-site about Amazonian deforestation. The demonstrates the importance of subdividing information patches into more meaningful segments.

There was a clear stopping point to this task, however, participants still were unsure when the task was complete. Even though the task was oriented towards data retrieval with an element of IR, participants were unsure how much information they needed. When gathering information from various sources, participants did not seem to be able to identify an end to the task, rather the gathering of information sources was a prerequisite for future work.

Problem-Task 6

This was the shortest task for all participants. Each participant used Google and searched for variations on 'US presidents'. All participants did not use more than one source for this task and there was a clear, right or wrong answer to the task. One user was satisfied with the text description of the Google results view and ended the task at that point. This reinforces the information visualisation principle in Section 2.8.1 as the participant used the information preview to recognise relevant information.

This was the first task in which participants did not gather information or sources of information as they progressed through the task. This suggests that for tasks which are solely data retrieval oriented, IG is not as prevalent.

Problem-Task 7

This task proved to be problematic for all participants. Some participants attempted to create a complex query in the Spotlight field, for example, entering 'cheese kind:pdf or kind:picture'. When this did not work, they tried to reformulate the query but after three different variations, decided that they did not know how to create a complex query.

Most participants could not complete this task as they were not familiar with complex queries. Two users attempted to compile two advanced searches, one for 'cheese' and only leave the PDF cascading row open, and then create another search, leaving only the pictures row open. The general consensus of the task was that 'I would never need to do that'. Therefore, perhaps the task was not relevant enough for participants to attempt. Several participants did comment that they would simply look through a results list of all file types and ignore items they did not need. This can be linked to the IF principle in Section 2.8.1 as a patch is created through the subdivision of the search space based on specific file attributes.

Three users attempted to create a complex query using the smart folder view. They tried to add criteria to imitate the query in the task, but resorted to searching for 'cheese' and minimising the sections which were not relevant.

3.7 Preliminary Requirements Validation and Revision

The results in Section 3.6 represent a summarised account of the evaluation transcriptions. They indicate particular activity patterns exhibited by participants. From these evaluation results, clear definitions of IR and IG were established.

The problem-tasks were designed to encourage the participants to retrieve and gather information and the key finding was that there is a void between users retrieving information and the ability to retain knowledge of the whereabouts of this information. The evaluations demonstrated that participants either consciously or subconsciously gathered retrieved information, but the conceptual problems arose when there was no tool to support the gathering of this information. This shows that Spotlight is not an IG system, rather a combination of an IR and data retrieval system.

The usability problems identified in the evaluations related to the representation and selection of results retrieved from Spotlight searches, for example, the lack of location information about a result or poor categorisation of results. Users did not exhibit any conceptual problems with the standard implementation of Spotlight, however this was not the case with the advanced view. Users did not understand the main function of the advanced view and consequently were not able to use it effectively. Users stated that they assumed it allowed them to create more advanced queries, or allowed them to save searches.

3.7.1 IR and IG

The evaluations demonstrated that for recursive information problems, participants exhibited the tendency to identify relevant sources of information to return to at a later time. This common process was performed in different ways, from minimising windows, to bookmarking web-sites, to simply remembering the location of a relevant file. It is this ongoing collection of pointers to relevant documents which defines the process of IG.

The main difference between IR and IG is that IR is concerned with locating and retrieving relevant information, whereas IG is concerned with collecting pointers to the identified information of relevance. The term 'pointers to information' is important, as an IG system must not gather the document information, rather a link to that information. This is derived from the section 3.6, where participants would identify sources of information to return to, rather than store the information itself. During the evaluations, when users felt they had retrieved enough information, or could not find anything of relevance, the task finished. With IG, their is no clear stopping point, as the process of collecting information is ongoing.

The IG function must not only allow the user to collect information, it must also allow the user to re-access this information. This function was demonstrated in section 3.6 where participants identified relevant information, continued their search and then returned to the relevant information later in the search process.

The evaluations results suggest that IG is a sub-function of another process, such as IR or data retrieval. Participants gathered relevant sources of information after performing a information or data retrieval process. In tasks 1-4, participants would initially retrieve or forage for information using Spotlight and then gather relevant items. On the rare occasions participants chose to browse for information, they would still identify relevant information, therefore, IG supports IR, IF, data retrieval and browsing. The gathering of information may rely on other processes, but the re-access of gathered information can take place at the point the user identifies an information need, therefore, the system must allow the user to re-access gathered information at any point in the process.

One of the key characteristics of IR and IG illustrated by the evaluations was that IR and IG do not change the participants knowledge of an information subject, rather the participant is aware of where they can access the required information to change their knowledge. Tasks 1-4 demonstrated that participants were concerned with finding the sources of information for future work, but not necessarily an answer to a problem which would improve their knowledge of the subject. As tasks 5 and 6 were more data retrieval oriented, they provided participants with answers to the questions which changed their knowledge of the subject area. This demonstrates the key roles of IR and IG. IR is a techniques to find relevant files, and IG is a technique which gathers links to these retrieved files in one location.

The evaluations demonstrated that there is a clear link between IG and IF. IG compliments IF as it transforms IF from a forgetful, exhaustive process, to a persistent process, which allows the user to gather information as search each patch.

The main component of the Information Gathering System (IGS) must be comprised of a customised IRS. The evaluations demonstrated various failings with Spotlight and these must be addressed with the proposed system.

The IR component of the system can be divided into the following stages:

3.7.2 Query formulation

The system must allow the user to retrieve information by matching a user defined query. The evaluations indicated that query formulation was the first stage of the IR process for every task. Participants resorted to browsing or exploring the hard drive, as a secondary process, to ensure that they had not overlooked any information. The evaluations reinforced the query formulation principle in Section 2.8 as users did submit tentative queries which often subdivided the information space to allow them to browse through a smaller set of results. The system must also support the matching of results which are 'like' the query and not exact matches as users rarely remember exact details about a file.

3.7.3 Query Result Representation

The results representation is concerned with displaying the results to the user. The evaluations identified filename and file location as the key attributes to determine information relevance during the IR process, therefore these must be the most prominent attributes in the result representation. The result representation must also include a file preview to improve the result recognition process. This was identified in the query information visualisation design principle in Section 2.8 and the evaluations reinforced this principle as participants placed much importance on the use of file previews to distinguish between file types in the Spotlight results view.

3.7.4 Query Results Categorisation

The evaluations demonstrated the importance participants placed on result categorisation. Participants ignored categories which were not relevant to their information need and mainly concentrated on the same category with each task. Participants identified the constraints with the current results categorisation and suggested categories can be found in table A.1. The results categorisation is a sub-component of results representation as it categorises the results displayed to the user.

The query information clustering and categorisation principle in Section 2.8.1 discusses the use of clustering to group results. However, customised faceted categorisation will be used instead of clustering because participants in the evaluations were extremely reliant on document type categorisation to select search results.

3.7.5 Results Manipulation

The system must support the basic result tasks identified in Section 3.4. Users identified the tasks they commonly perform with search results and explained how the current Spotlight implementation does not clearly support all of these tasks. The system must visibly support the basic result tasks.

3.7.6 Query Re-access

The recursive nature of IR must be supported by allowing users to re-access saved queries. In Section 3.4, users identified one of the constraints with Spotlight is that they are not able to re-access previous queries. This reinforces the information re-access design principle in Section 2.8.1.

3.8 Conclusion: From Forgetful Foraging to Persistent Gathering

This chapter detailed the two user studies which were used, in conjunction with the design principles identified in Section 2.8.1 to produce the full requirements specification in Appendix A.2. The requirement validation above lead to an important redefinition of an IR and IG system, from a forgetful IF system to a persistent IF and IG system. The system will incorporate the key techniques of IF, allowing the user to search within patches, however, they will be able to gather relevant information during their foraging activities to retain information. The requirements specification is used to guide the design of 'Searchlight' a prototype IR and IG system in the next chapter.

Chapter 4

Design of 'Searchlight', a Prototype IR and IG System

4.1 Introduction

This chapter describes the high level system design which has been derived from the requirements specification. Important design decisions are discussed and the adopted design methodology is explored. The high level systems architecture is analysed which provides a basis for the first evolutionary prototype design. The prototype, Searchlight, is then evaluated with users and a redesign is proposed which provides the design structure for the low level implementation.

4.2 Design Decisions

'Choosing appropriate features and devoting the needed resources to implement them correctly can save you time and effort later' [Apple, 1992-2006a].
The primary aim of the project is to investigate the concepts of IG and IR, therefore the design emphasis has been placed on implementing the core components identified in the requirements specification. This means that the application will not be a complete, commercial package ready for distribution. The decision was made to concentrate on these core components areas and push 'aesthetic features' such as user preference stage and help documentation, towards the end of the development. The reason for this is because this added functionality is vital for a complete application, but concentrating on it early in the design process would have meant that the key areas would have suffered and potentially may not have been implemented.

The decision was made to omit user customisation from the prototype design. Most native Macintosh applications use preference panels accessed through the main menu bar where the user can change and save various preferences. This was not implemented in the final prototype as it was necessary to concentrate on creating a standard prototype

design which was used to evaluated the concepts of IR and IG. The requirements, design and testing processes were user oriented to investigate the key features and components of the system which support typical activities, therefore the final prototype should closely support the needs of the majority of the target audience.

4.3 Design Methodology

A user centred evolutionary prototyping technique [Sommerville, 2001] was adopted during the design and implementation phases. As described in section 1.2 the design and implementation were concurrent activities. This is true to a certain extent, as the design of each prototype evolved concurrently with the implementation of each prototype. However, an initial prototype was developed as a conceptual model from the requirements specification before the implementation of the intermediate functioning prototypes were developed. This was because it was important to translate the concepts ascertained during the requirements process, into a physical object to ensure the specification was feasible.

The design and implementation processes were guided by the Model View Controller design paradigm.

4.3.1 Model-View-Controller Paradigm

This design pattern classifies objects into appropriate layers depending on the role of that object. The model objects contain the underlying data, or data structures of the application, which are not visible to the user. For example, the model in this system would be an object which creates the query to search System Store for matching metadata attributes. View objects are objects which make up the Aqua[1] user interface. In this system, the view objects were windows, buttons, views etc. View objects do not interact directly with the model, therefore an intermediate layer is needed, the control layer. The controller acts as a mediator between the two layers, for example, the controller layer would link the query text a user enters into a search field to the query object in the model layer.

The MVC design pattern was chosen for the following reasons

- The layered design allowed the view to be developed before, during and after the implementation of the model and controller layers without out having a negative impact on the entire system.
- The view could be developed before the controller and model layers to expose to users and gain feedback on the design before the system was fully implemented. This saved a lot of time in the design process as it meant a GUI could be quickly developed without having to implement any functionality and feedback could be quickly obtained.

[1]The standard UI environment for Mac OS X 10.4

- The program was more extensible, as once the basic back end was implemented, there was more time to redefine and discover requirements with the users.

The design stage concentrates on the overall systems architecture and discusses the detailed design of the view layer. This is because, as stated above, the initial prototype will be an advanced GUI which will closely match the GUI design of the final prototypes. The initial prototype will be developed in Interface Builder, which is a GUI development package for Mac OS applications. The MVC design pattern is used during the full implementation of the system and the model and controller layers are discussed in Section 5.4.

4.4 Architectural Design

The architectural design is discussed by dividing the main components of the system, as identified in Section 3.7, into their respective MVC components, rather than dividing the entire system architecture into the MVC layers. This is because it is important to understand how the system functions as a whole before going into greater depth to understand the relationship between the individual MVC components.

4.4.1 Overview

Figure 4.1 illustrates the relationship between the individual system components suggested in Section 3.7

The diagram begins by assuming the person has identified an information problem and has chosen to use the system. The query formulation component represents the user formulating and entering a query string into the system. This query is then executed and the system displays the results to the user in the results representation component. These two components have a two-way relationship as the results returned to the user may not be relevant, therefore they may decide to reformulate their query. The result representation component is made up of two sub-components, file information representation and results categorisation.

When the results are returned, they are sorted into categories which the user can select to display results of particular types. When the user has selected a desired category, they will make their result selection, this is guided by the file information representation component which controls the information displayed for each result. If the user decides that the results are not relevant and does not wish to reformulate the query, or has exhausted their search, they may decide to end the process. If the user has selected a result they consider to be relevant, they can manipulate the results in various ways, for example, open the file, or gather the result. After result manipulation, the user may wish to end the process, gather the relevant result, or reformulate the query. If the user decides to gather the relevant result, they may then end the process, reformulate the query to retrieve more results, manipulate another result or manipulate the gathered

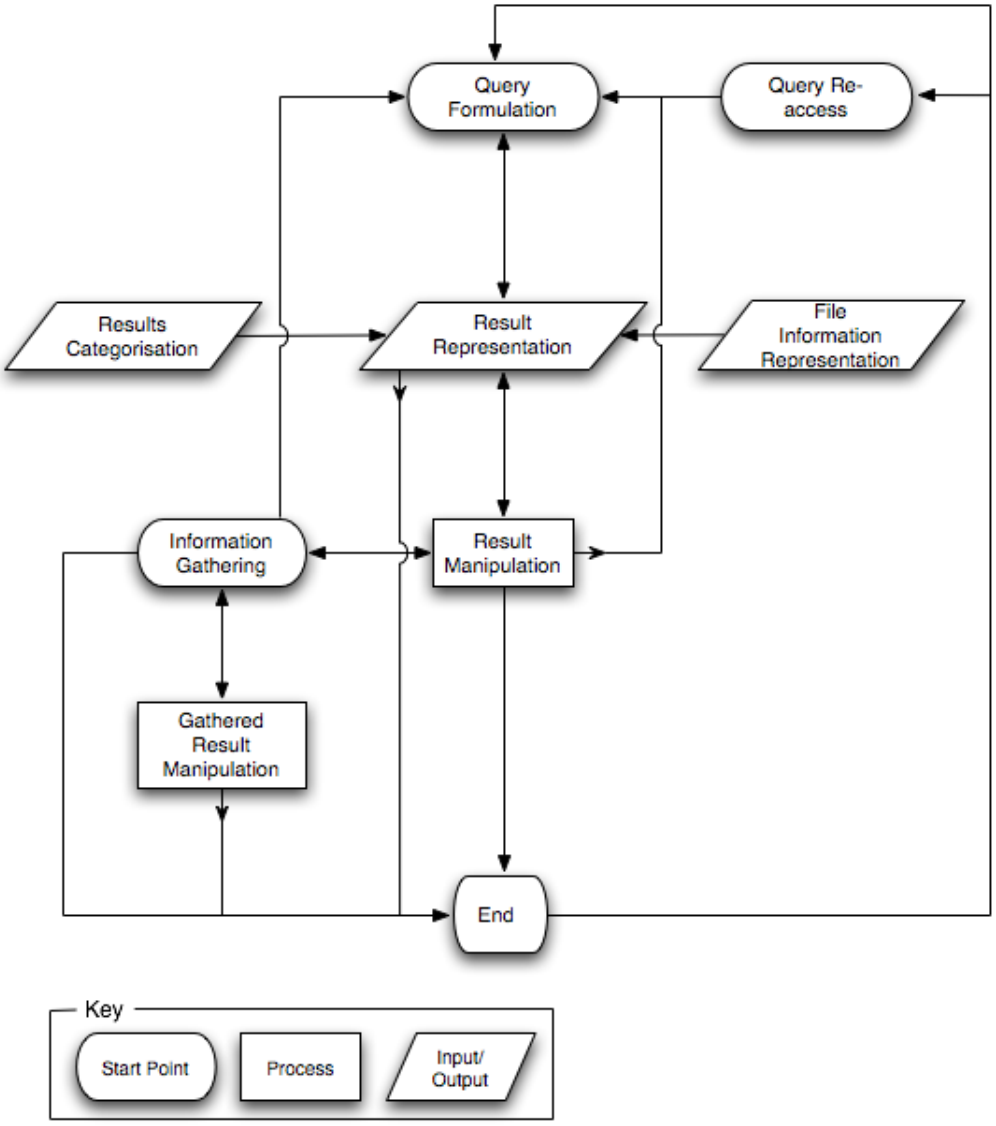

Figure 4.1: System Architecture Diagram

result itself. If they manipulate the gathered results, they may then return to the IG component or end the process.

Query Formulation, Query re-access and Information Gathering components have been identified as possible starting points in the system, as the process could begin at any one of these points. As functional requirement 14 states, the user may wish to re-access a saved query to retrieve information which they had previously accessed. Alternatively, as functional requirement 17 states, the user may wish to re-access gathered information, therefore, they would only use the IG component and disregard the rest of the system.

The diagram helps to emphasise the recursive nature of IR and IG and demonstrates that there are many paths through the system. The 'end' point has been added into the diagram to demonstrate the finishing point for interaction with the system. However, 'end' of the IR and IG process may be the start of a task process, or may lead directly back to the start of the IR and IG process itself.

Figure 4.2 illustrates how all of the individual layer objects of the system components are related.

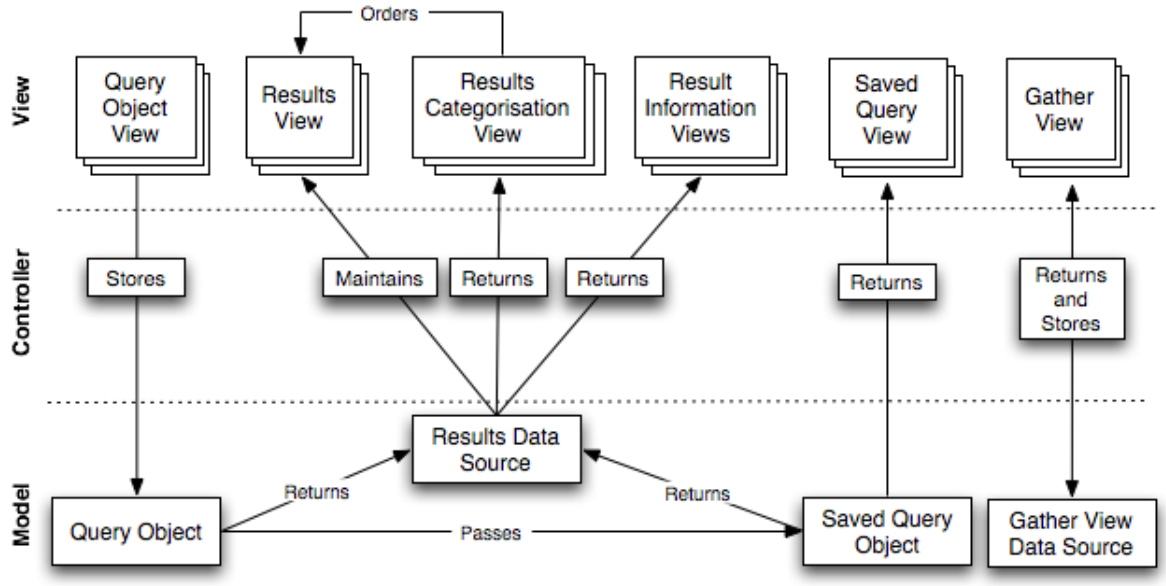

Figure 4.2: MVC Design Model

4.4.2 Query Formulation

The query formulation requirements discuss the need for a query formulation interface which allows the user to create a custom query to search their hard drive for information.

The model will be a query object which uses the query string entered by the user in the

query view to define the search. The query object will interact with the system store to return metadata items which match the user's query.

The view consists of the query object view, which is the primary input to the system. The query view is an editable input field where the user enters a query string to retrieve information from their hard drive.

In a simple IR scenario, the query view acts as the second stage of the IR process. The first stage is the identification of the information problem, the second stage is translating this problem into a query. In a more complex IG scenario, the query view will act as a starting point and intermediary in the process. The user may commence the search using the query view, gather relevant information and then return to the query view to redefine the query.

Figure 4.1 demonstrates that when items are returned, they are passed to the results view in the form of search results. It is important to acknowledge that the search results can be used to reformulate the query entered in the query view and this is demonstrated by the use of double arrows between the query view and the results view. This functionality was derived from functional requirement 3, 'Users must be able to reformulate existing queries'.

The controller will link the query object and the query object view together. Figure 4.2 shows that the controller 'stores' a value from the query object view in the query object. This means that the controller must store the query string from the query view in a variable, to be used by the query object when required. This is discussed in detail in section 5.4.4.

4.4.3 Result Representation

The result representation manages the retrieval and organisation of query results.

The model depends on the implementation of the results view. The model must provide the results view with the data to be displayed in the UI. Figure 4.2 describes the model as the 'results data source', which would typically be a dictionary or an array which stores all of the returned results.

The view is made up of the results view and the result information view. The results view is the core view of the results representation component. The results view is a collection of all of the results which match the users query. The user utilises the results view to select particular results and identify whether or not they are relevant. The results view is the centre point of the IR and IG process, as it is from here the user decides which files they wish to manipulate. The result view leads the user into a selection process, by providing specific result information to allow them to choose a particular result. The results view incorporates the file information representation and result categorisation modules to perform this task.

The result information view displays relevant information for each result in the result view, for example, filename and file type. The information displayed for each result

guides the user's result selection process and the information displayed must include the required file attributes identified in functional requirement 5. The information view will also include a file preview which gives the user a visual representation of the file type to aid the result selection process, this was derived from functional requirement 7.

Figure 4.2 shows that the controller for the results view and information view have different functions. The controller for the results view must maintain the flow of data from the results data source to the results view. Whereas, the controller for the information view returns the data from the data source to the view object. This is discussed in detail in section 5.4.4.

4.4.4 Result Categorisation

The results categorisation component organises the returned results into categories defined in Appendix A.1.

The model for the results categorisation component is the results data source as it stores or manages all of the matched query results.

The results categorisation view is vital for the management of large amounts of information in the results view. The results categorisation view allows the user to select different categories to order the results in the results view, this was derived from functional requirement 9.

Figure 4.2 shows that the controller returns data from the model to the results categorisation view. The controller must return the results from the results data source to the results categorisation view in their respective categories. This view then orders the results in the results view based on category selection.

4.4.5 Result Manipulation

The result manipulation component encompasses the tasks users perform on query results. This common set of tasks was identified in the requirements gathering phase. They are, open the file, show the location of the file and reveal more information about that file.

The results manipulation component is not included in figure 4.2 as it does not conform to the MVC design paradigm. The results manipulation component essentially consists of actions the user can perform on results, therefore most of the functionality will be implemented in the controller layer. This is because the controller will provide the functionality for the actions and link this to the selected result.

4.4.6 Query Re-access

The query re-access component is concerned with saving and re-accessing previously saved queries, this was derived from functional requirement 14. The model will consist of a saved search object which stores the saved query for future use.

The view consists of the saved query view which could have many different implementations, ranging from a list of recent queries, to a source list of saved queries, to a query history. The saved query view provides a secondary input into the system as the user can access a saved query at any time during the process.

Figure 4.2 demonstrates that the query object passes the query to the saved query object. It is then the role of the controller to return the saved query to saved query view to display to the user.

4.4.7 Information Gathering

The IG component acts as a secondary input and a result manipulation tool in the system. As demonstrated in figure 4.2 the model for the IG component is the gather view data source. This must store a pointer to each gathered item of information, which could be in the form of an object for each information item, or in a container such as an array or a dictionary.

The main view object is the gather view which acts as a storage area for links to gathered results. Users may use the results returned in the results view to store a subset of these results in the gather view, this was derived from functional requirement 15.

Figure 4.2 demonstrates there is a two way relationship between the gather view and the gather view data source. The controller must perform two functions, it must store the items added to the gather view whilst returning these items from the data source to be displayed by the gather view.

4.5 Prototype 1: GUI Prototype

The primary aim of the initial prototype was to obtain feedback to guide the design of the UI and the key features of the system. The emphasis was placed on quickly obtaining feedback on the initial system to verify that the initial requirements were accurate and could be translated into something the user could understand. For this reason, a GUI prototype was implemented and evaluated with users.

The prototype consisted of a sophisticated interface which was created using an application called Interface Builder. This is a drag and drop development environment which requires no code implementation to create complete interfaces which closely match the final look and feel of the system.

The system architecture details the functionality and the flow of data between components, but it does not describe the individual interface components and interaction

design. When creating a dedicated Mac OS X 10.4 application, Apple suggest guidelines to ensure that the application complies to interface and interaction standards. Two different sets of guidelines are suggested, Apple Human Interface Design Principles [Apple, 1992-2006a] and Apple Human Interface Guidelines [Apple, 1992-2006b].

The Human Interface Design Principles detail key principles which are vital for designing intuitive interfaces. They are more general principles about Human Computer Interaction and how to design your application to be 'usable'. The Apple Human Interface Guidelines are specific guidelines to be followed for the design of particular elements of the interface, for example, the appropriate button type for a particular action. The human interface design principles were used to create the GUI prototype. The reason the specific design guidelines were not used in the creation of the GUI prototype was because they are extremely detailed and more time would have been spent configuring the look and feel of the system than addressing the key concepts. These detailed guidelines were used in the intermediate prototypes.

As none of the back end of the system was implemented, the users were not able to see the results of their actions, therefore, during the user evaluations, the evaluator had to run through the functions of the system with the user. This approach was adopted because to implement any of the back-end of the system would result in a long development time to learn the required language and implement these features.

4.5.1 Detailed Design

This section discusses the detailed design of the view layer of the system. Figure 4.3 illustrates a GUI design template which will be used to guide the design process

Window Types

The first stage of the GUI design is to decide what form the main application window will take. The system will be a single, main windowed application with optional sub windows and drawers. To design the system for modelessness, the gather view will be created in a separate window, accessed through the main application window, or application menu bar.

The application will not allow for multiple main windows as it does not conform to the Apple multiple windowed paradigm which document based applications adopt. Adopting a multi-windowed design which enables users to perform multiple searches simultaneously would cause system bottlenecks when performing searches. One of the benefits of Spotlight, as identified by users, is the speed of result retrieval, if multiple searches were being performed at the same time, the response would be unacceptably slow and the system would not satisfy functional requirement 9, 'The retrieval speed of the system must be as fast, or faster than the current Spotlight implementation'.

The main window is optimised for a 1024 x 768 display to conform to Apple design principle, 'unless you know that your users will be using a specific display size, it is best

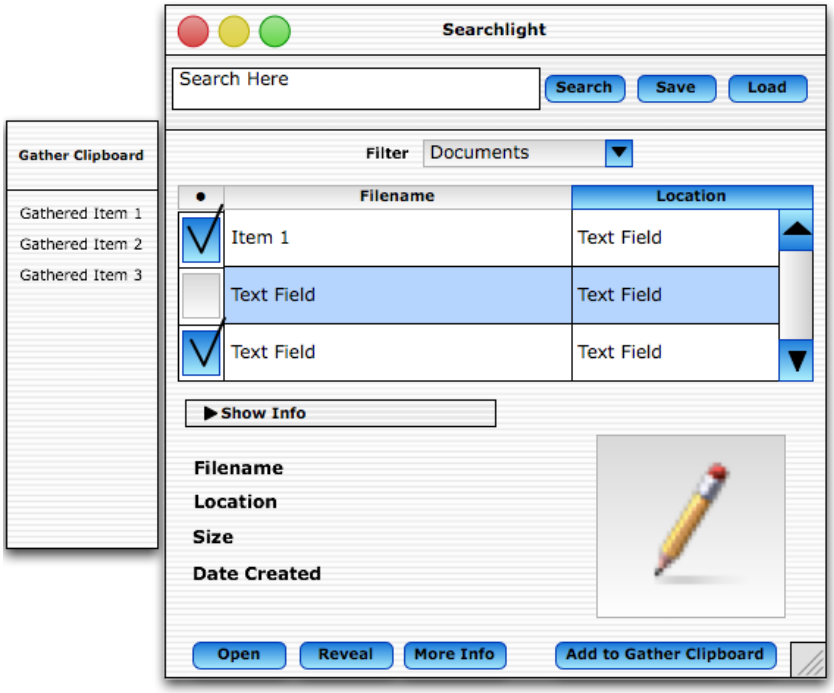

Figure 4.3: GUI Design Template for GUI prototype

to optimise your applications for display at 1024 x 768 pixels'.

The only sub window of the application is the gather view, which is a floating window used to store gathered items.

Query View

The query view consists of a single search field. The search field conforms to the consistency principle as it is the same search field used in Finder and the advanced Spotlight view.

Results View

The results view consists of a table view which has three columns. Each column represents an attribute of the returned result and each row represents a result. The first column is a check box which indicates whether or not the user has already viewed the result, this conforms to functional requirement 8. A check box was chosen so that if a user has seen a result but wishes to mark it as unseen again, they are able to. This was preferred to changing the colour of the text when selected as it could introduce too many colours to the interface.

The second column contains the name of the file and the third column contains the

location of the file. These attributes were identified to be the most important file attributes for recognition in the requirements gathering stage. A table view was chosen over a scroll view or a non list layout as they are ideal for storing large amounts of similar data and have built in sort descriptors. The results view columns are interchangeable and resizable and the vertical and horizontal scrollers are provided if necessary.

Result Information View

The information view consists of a custom view which displays file attributes and a file preview. When the user selects a particular result, the attributes for that result update to display the metadata attributes for the selected file. This is a static view which is not resizable or customisable, this is to give the impression of stability and consistency throughout the application, as each result reacts in the same way. The file preview is an image field which represents a file system preview of the document. A large thumbnail view is used to ensure that it is clearly recognisable to aid the user's result recognition process. The whole information view is non-editable and is placed below the result view to give a natural progression down the results list to the information view. The information view was originally positioned in the table view, e.g. a column for each information attribute, however, this is a poor use of space and scrollers have to be used to view all of the information.

Result Categorisation View

The result categorisation view is a filter menu positioned above the results view. The verb 'filter' is used to indicate to the user that they are able to filter the results based on the criteria in the list, this simple naming convention (the use of verbs for actions) adheres to Apple design principles. A filter menu was used instead of a list to save space and because a filter drop down menu is consistent with other OS applications. When the user selects a category, the results view changes to display only the results which match the filter criteria. This method was chosen because users in the evaluations placed so much emphasis on categorisation of search results. It also ensures that the results view is manageable and the user is not overloaded with information.

Result Manipulation

The main information tasks which were identified in requirements specification, are supported through the use of the task buttons. Each task has a separate button at the bottom of the interface, again to show natural progression through the system. The buttons are standard buttons used throughout the OS and they are identical in size, spacing and operation. Each button has a label which identifies the action of the button. The 'open' button opens the selected result, the 'more info' button displays the standard OS 'Get info' inspector window and 'reveal' reveals the file in Finder. The reveal button is used to support the gather view, by displaying the file for the user to drag to the gather view. The application tries to ensure that there are no hidden

features and that complexity is kept to a minimum by only presenting the user with closed options for result manipulation. The application also reduces the occurrence of errors as the only user input into the system are the query and gather views.

Gather View

The gather view is the only other window in the application. The gather view consists of a floating utility window containing a table view with two columns, name and location. A floating window was used instead of a drawer because it supports modelessness by allowing the gather view to be open whilst the main window is closed. A table view was used again as it is ideal for storing and ordering large amounts of similar information. The columns identify the name and location attributes for the gathered information as they were the most important file attributes identified in the user studies. The gather view supports direct manipulation as it allows the user to drag a file onto the tableview, to reinforce the gathering metaphor used in the system. This was chosen instead of including a task button with adds the file to the gather view because it provides the user with more feedback and they can see the results of their actions. A file in the gather view also uses the same task buttons as a file in the results view, this is to keep actions consistent and avoid unnecessary repetition.

Query Re-access

Queries are saved and loaded through two buttons at the top of the interface. When the user selects the save button, a standard save sheet appears to ask the user for the location of the save. A standard loading sheet appears asking the user where they wish to load the saved search from, when the load button is pressed. This method is used so the save and load methods match user expectations and are consistent with other OS applications. When a search is loaded, the saved state overwrites the current state of the system as if the user was continuing their previous search.

Figure 4.4 displays the first GUI design

4.6 Prototype Evaluations

The evaluations for the initial GUI prototype differed from the cooperative evaluations for requirements gathering, this was because the aims of the evaluations were different. The aims of the prototype evaluations were as follows:

- To identify usability problems with the interface
- To identify any components which did not match the users expectations
- To walkthrough the concepts of the system with the user to ensure that the requirements and principles are valid
- To identify positive and negative feature of the interface

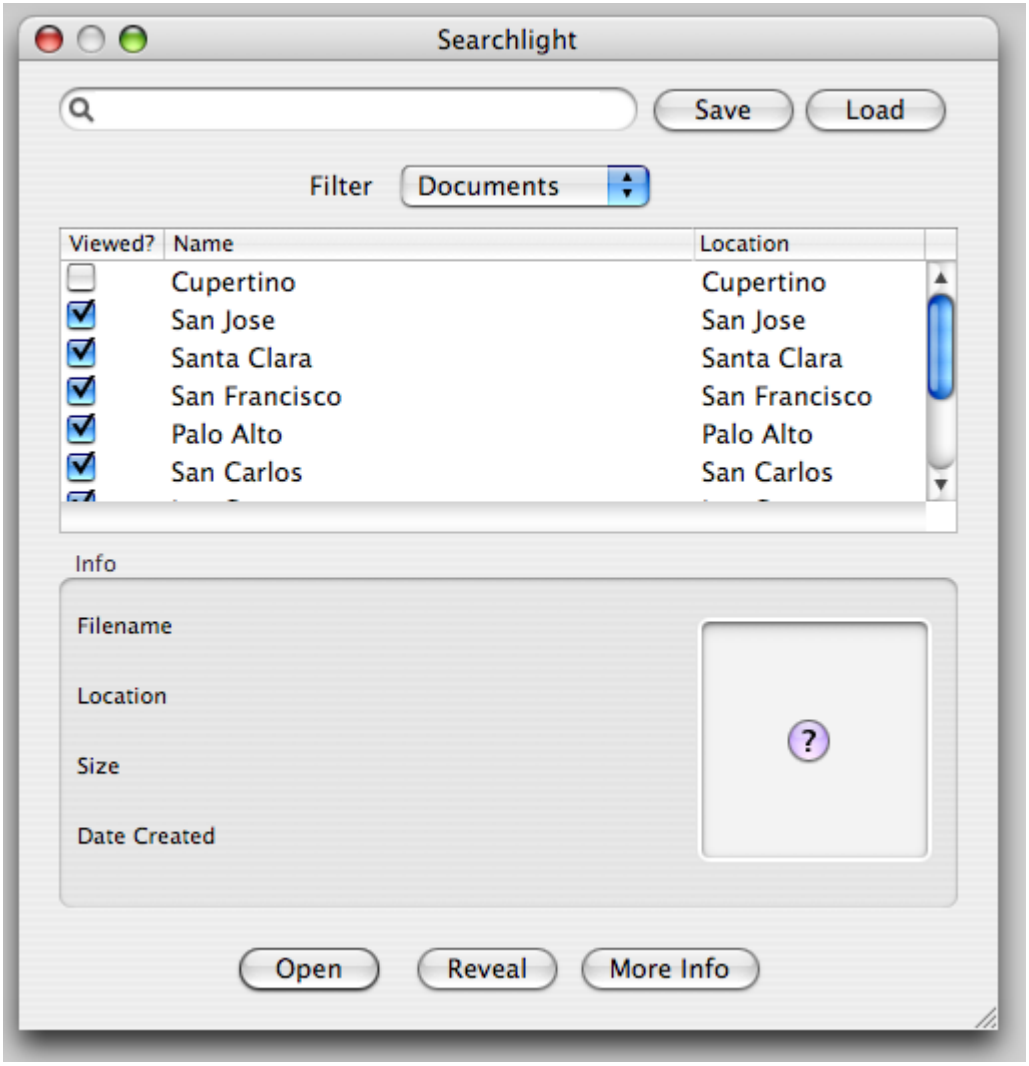

Figure 4.4: Initial GUI design

Five Mac OS users were chosen to walkthrough the interface with an evaluator. The evaluations were extremely informal, and lasted approximately fifteen minutes. Notes were taken during the interviews and the main results are detailed below.

4.6.1 Evaluation Results

The evaluation results suggested that the main problems with the interface are usability problems, rather than conceptual problems. Users understood the key IR and IG concepts and generally expressed positive comments regarding the usefulness of the system.

Users expressed their dislike of the query saving method. It was identified to be 'too much hassle' and most users said that they would not use the function as it was too complex.

Issues regarding where the search would be saved were raised and one user noted it was ironic that they would have to use a search tool to find a saved search. A suggestion was to have a visible list of saved searches that a user can simply click on to recompile the search. There was also a problem between the user expectations and the system actions when a search is saved. A user was not sure if it saved the previous search state or would update as soon as it was loaded.

The 'open' button was deemed unncheck-boxecessary as most users thought it was too inconvenient to have to select the search result and navigate to another button to simply open it. A simple double click use of the return key on a search result was suggested.

The 'filter' button caused a divide in the group as some users liked it because it did not take up space, whereas, others did not like the extra click involved to view the categories. Some users also did not like the idea of hidden categories. Users were happy with the categories used and the suggestion to create a category for 'other files' was made.

The file attributes proved to be the most popular attributes users look for in files and the associated actions represented typical file activities. Users suggested that the 'more info' button should display a selection of attributes instead of the standard 'get info' window as the application already displays most of the information the 'get info' inspector displays.

Users were slightly confused with the function of the check-box used to identify if a result had already been selected. They felt that it was not that important to dedicate a column to a check-box and it looked out of place in the interface.

The gather view also prompted differences of opinion as some people preferred the floating window as they were able to close the main window and keep the gather window open. Others felt that it 'got in the way' and that a drawer would be more applicable. The final design will experiment with both views.

4.6.2 Redesign

Using the feedback from the GUI prototype evaluation, the prototype was redesigned to incorporate the changes. The key changes are as follows:

Result Categorisation View

The result categorisation view changed type from a drop down menu to a table view populated with the same options. This has been placed along side the results view and has been given the heading 'file type'.

Gather View

The decision was made to move the gather view from a floating utility window into a drawer view as users did not like the idea of a floating window in the application. The drawer view also uses less space as it is hidden until activation.

Saved Query View

Saving queries has been completely redesigned. There is now a single save search button which adds the query string to a saved query list in the drawer view above gather view. The user then selects the appropriate saved query and re-runs it. This simply passes the original query to the query view and recompiles the new search.

Result Manipulation

The 'more info' button now opens a custom inspector which contains custom attributes for the selected file. The method of identifying a previously selected result was changed from a check-box to changing the text colour of a link when selected.

Figure 4.5 displays the redesigned GUI.

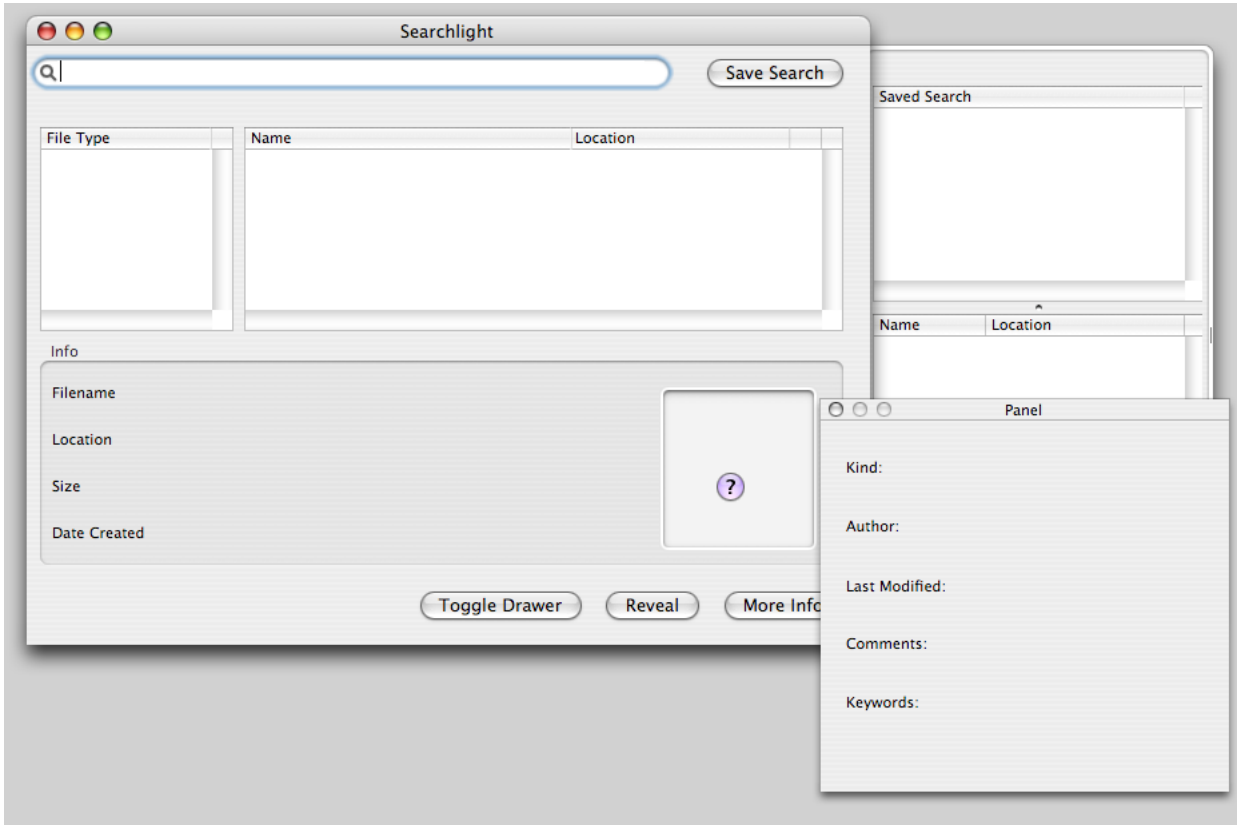

Figure 4.5: Redesigned GUI

4.7 Conclusion

The design process was guided by the functional requirements derived in the requirements gathering stage and the design principles in Section 2.8.1. The design demonstrates that design principle 6 'Collaborative Foraging' has moved out of the scope of the system. The collaborative principle is out of scope for the system as it is based around the sharing collaboration between multiple users to build information networks. The desktop environment is a predominantly individual environment, therefore the sharing of information is not applicable for this system at this moment in time.

The GUI prototype provided some valuable information to guide the design of the fully functioning prototypes. The redesigned prototype will provide the basis for the implementation of the final prototype. It also proved that the key concepts identified in the requirements phase are transferrable to a usable interface. However, as no functionality was implemented, users must be involved in the implementation and evaluation processes to ensure that the system closely matches the user's needs.

Chapter 5

Detailed Design and Implementation of 'Searchlight'

5.1 Introduction

This chapter builds on the high level design of the first GUI prototype to implement the functionality of the system. The chapter begins by discussing the possible implementation languages and moves on to discuss the more complex aspects of the model and controller layers of the system. The chapter presents the first fully functioning prototype, 'Searchlight' for user evaluations.

5.2 Hardware Analysis

A brief analysis of the hardware required to run the final prototype is necessary as not all PC users will be able to run the developed system. Due to the architecture of the operating system which is designed for the IBM PowerPC platform, any native Macintosh application will not run on any other operating system, regardless of its architecture (x86 or otherwise). As the operating system only runs on Apple Macintosh hardware the audience of the application is limited, as Apple computers on represent roughly 3% of the entire PC market.

Apple are currently undergoing a hardware transition from the IBM PowerPC platform to x86 Intel chips. This means any application built to run on the old PowerPC platform will not run on the new Intel machines. Therefore, to ensure longevity and future development of the application, it must be compiled to run on Intel machines. Apple introduced a concept called 'Universal Binaries' which is an application built for both PowerPC and Intel architectures, therefore, the application must created as a Universal Binary. The only hardware requirement would be that the user has a machine which is capable of running Mac OS X 10.4. As Apple machines have a relatively long life, machines dating back to 2002 and earlier are capable of running this operating

system.

Despite developing a dedicated Mac OS X application, it is important to acknowledge that the concepts and ideas developed in this project are transferrable to other platforms. It is the development methods which are not transferrable.

5.3 Implementation Language

As stated in requirements and design chapters, the system is manipulated through a GUI, therefore the development language selected must support easy creation of a sophisticated GUI which adheres to Apple Interface Design guidelines. Additionally, as the interface will undergo a series of changes driven by users, the interface for the system must be easy to update without leading to large development times.

Non-functional requirement 4. states that the application must be a native Macintosh application, therefore this automatically excludes languages such as Visual Basic, which are not available on the Macintosh platform. An object-oriented language was chosen for the implementation of the system as the difficulties of implementing a system in, for example, C or Pascal, far outweigh the benefits. The main disadvantage with implementing the system in a language such as C, would be that there are no specific libraries provided with C to create an interface. The appropriate libraries, for example Swing in Java or AppKit in Objective C, would have to be downloaded, configured and installed.

This lead to the choice of three possible implementation languages, Java, C++ and Cocoa frameworks with Objective C.

One of the benefits of Java is that it has built in libraries for interface development, swing and awt. These would provide all of the necessary tools to create a complete interface, however, the layout managers used in Java, such as 'gridbag', are notoriously intricate and it would be difficult to create an application which has the look and feel of a native Mac OS X application.

Another benefit of using Java is that it is a cross platform programming language, therefore an application created on Mac OS X 10.4 would run on Windows. Conversely, the system is centred around the use of Spotlight technology, therefore creating an application in Java would not make it cross platform as other operating systems do not use Spotlight technology.

Java uses garbage collection as opposed to reference counting for memory management which has benefits and constraints. A major benefit of garbage collection is that it allows the developer to quickly create applications which do not leak memory, thus enabling them to concentrate on other areas of the development. One drawback is that garbage collection is slower than reference counting, therefore an application developed in Java would be slower than a native application.

One of the major benefits of choosing C++ over Java is that it is a native language whereas, Java is an interpreted language. The interpreted nature of Java means that

applications written in Java run much more slowly than native applications created in C++. Both C++ and Java have similar characteristics and were not chosen as the implementation languages for one main reason, Cocoa Frameworks with Objective C is the primary application environment for Mac OS X 10.4 applications.

5.3.1 Cocoa Frameworks with Objective C

Cocoa is a Mac OS X application environment which is effectively divided into two sections, the runtime environment and the development environment. The runtime aspect of the Cocoa environment is the Aqua user interface the application displays to the user, whilst the development aspect is a suite of object-oriented class libraries. These libraries are divided into two core frameworks, the foundation framework and the application kit. The application kit provides all of the appropriate libraries to design user interfaces, whereas the foundation framework provides libraries for all objects which are not exclusively used to support a user interface

Cocoa applications can be developed using C, C++, Objective C and Java, however, using languages other than Objective C or C can lead to problems. Java can be used with Cocoa as Apple have implemented a set of parallel Java classes which 'bridges Java interfaces to their corresponding Objective C implementations' [1]. However, Apple are no longer providing APIs for Java-Cocoa with the next release of the operating system, so suggest that using Java-Cocoa is only for developers with a good Java background who want to learn the basics of Cocoa.

Objective C is an object-oriented extension of ANSI C and is the primary language used for developing Cocoa applications. All of the major OS applications are created using Cocoa and Objective C, therefore it would be ideal to create a solid, efficient application which has the look and feel of a Macintosh application. Objective C uses reference counting for memory management which has advantages and disadvantages. The major advantage is that it is quicker than garbage collection and gives the developer more control over objects in the system. The major disadvantage is that the developer must ensure that the application de-allocates all objects after use otherwise it will have adverse effects on system performance.

The main benefits of using Cocoa with Objective C are that it supports the rapid development of robust applications, fully working Cocoa applications can be created without writing a line of code. The development environment for Cocoa applications consists of XCode and Interface Builder. Interface Builder is an interface development environment for Cocoa applications. It works with the application kit framework to provide a palette of key interface components to create a fully working Cocoa application. Interface Builder is tightly integrated with XCode which is an IDE for common programming languages.

The analysis of the possible implementation languages demonstrated that using Cocoa frameworks with Objective C would be the best choice. All commercial applications for

[1] http://developer.apple.com/documentation/Cocoa/Conceptual/JavaTutorial/chapter01/

the OS are creating in Cocoa, therefore it would be foolish to forego its benefits and use another language which is not so tightly integrated. Cocoa provides a plethora of classes which cover a wide array of components, including the creation of Spotlight queries. The major constraint with using Cocoa and Objective C is the time that must be invested in learning the language and implementation techniques. As these are completely new languages and environments, a substantial part of the development time will be learning these new development approaches.

5.4 Prototype 2: MVC Revisited

The system design in Chapter 4 was guided by the MVC paradigm, however, only the view layer was implemented in the first prototype. The implementation process for prototype 2 is discussed below, with particular concentration on the more complex components of the model and controller layers.

5.4.1 Overview

The system consisted of three classes, `Controller.m`, `Transformers.m` and `Gatherboard ArrayController.m`, one category, `ToolbarCategory.m` and an interface object `main menu.nib`. The `Controller.m` class contained the majority of model objects and was instantiated to create the controller object which was exposed to the `mainmenu.nib` interface object. The `mainmenu.nib` object contained all of the view objects such as the `NSSearchField` and `NSTableView` objects. The `Transformers.m` class contained transformer methods which took a value from a view object, converted it to another value and returned it to the view object. For example, a transformer method was used to transform the size of a file in bytes to kilobytes, megabytes and gigabytes. The `GatherboardArrayController.m` class contained the model objects for the gather view. The `ToolbarCategory.m` category extended the `Controller.m` class to manage the the interface toolbar.

5.4.2 Model Layer

The low level design and implementation techniques for the model objects identified in figure 4.2 are discussed. Figure 4.2 suggests that the model consisted of four objects, however, the actual implementation differed. The results data source object was not implemented, as due to a Cocoa technique called 'bindings', discussed in section 5.4.4 the model object for the results view did not need to be implemented directly. The model layer was divided into the query object, the saved query object, and the gather view data source. As the query object is the largest and most complex component of the system, it is described in the most detail.

Query Object

The main component of the IR function of the system is centred around the use of Spotlight to query the system store for files matching the query string. This is the main core of the application and required the longest development time.

Two APIs are available for querying metadata using Spotlight, they are `MDQuery` provided by the Core Services framework and the `NSMetadataQuery` provided by Cocoa Foundation Framework. The concepts are identical for both APIs, however, the implementation is different and as `NSMetadataQuery` is an Objective C class as opposed to a Core Foundation class. `NSMetadataQuery` was chosen as it is the Cocoa interface to Spotlight.

There are three main stages to querying metadata with Spotlight, query expression definition, query configuration and query execution.

Stage 1: Query Expression Definition

The design of the query expression defines the query the Spotlight engine uses to match metadata attributes in the system store. The standard Finder Spotlight query expressions is defined as follows:

```
(* = "queryString*"wcd || kMDItemTextContent = "queryString*"cd)
&& (kMDItemContentType != com.apple.mail.emlx)
&& (kMDItemContentType != public.vcard)
```

The entire query is called a compound query because it consists of four individual queries, a comparison query, a content query and two exclusion queries. The comparison query, `* = "queryString*"wcd` searches all metadata attributes in the System Store to match the query entered by the user which is represented by `queryString`. The content query, `kMDItemTextContent = "search string*"cd`, searches the text content of each file to match the `queryString`. The exclusion queries, `(kMDItemContentType != com.apple.mail.emlx)` exclude certain file types from being returned by the query. This is the Finder implementation of a Spotlight query and it differs from the standard Spotlight query implementation only in the exclusion queries, as the standard Spotlight implementation returns emails and vcards.

Non-functional requirement 8 states that the system should imitate the standard Spotlight query, however, the `NSMetadataQuery` API, does not use the same query syntax as the Finder implementation of a Spotlight query. Therefore, another object, an `NSPredicate` object, must be used to define the query expression. The `NSPredicate` object was created in the `createQueryPredicate` method of the `Controller.m` class. Five separate `NSPredicate` objects were created to imitate the standard Spotlight query expression. Only the comparison predicate and the exclusion predicates are discussed as they demonstrate the key characteristics of the final predicate.

The comparison predicate, `* = "queryString*"wcd`, as identified in the standard Spotlight query was implemented in the following way:

```
NSExpression *leftExp =  [NSExpression expressionForKeyPath: @"*"];
```

```
NSExpression *rightExp = [NSExpression expressionForConstantValue:
_queryString];

unsigned comparisonPredicateOptions = (NSCaseInsensitivePredicateOption
&& NSDiacriticInsensitivePredicateOption);

comparisonPredicate = [NSComparisonPredicate predicateWithLeftExpression:
leftExp rightExpression: rightExp modifier: NSDirectPredicateModifier type:
NSInPredicateOperatorType options: comparisonPredicateOptions];
```

This demonstrates the difference in construction of the two query expressions. The `NSComparisonPredicate` implementation divides the query into two expressions, a left and right expression which are compared. This means that all metadata attributes, identified by the `*` value are compared with the value stored in the `_queryString` object. One problem encountered using an `NSComparisonPredicate` was trying to configure the metadata attributes which are `like` the `queryString`. This is because the standard Spotlight implementation does not require the user to enter the exact query string. If the user is looking for a document entitled 'some spoons' and they enter a query string as 'spoons' Spotlight will retrieve this file. When using an `NSComparisonPredicate` object, the predicate type must be specified. In the above example, the `NSInPredicateOperatorType` is specified. This is because, the original type was set to `NSLikePredicateOperatorType` which should imitate the functionality of the standard Spotlight implementation. This proved not to be the case, as the system would only return exact query string matches. Therefore, the `NSInPredicateOperatorType` was used, which solved this problem.

Section 3.4 suggested that Spotlight users do not require certain file types to be returned by Spotlight. These excluded file types, such as `.plist` files, were excluded by creating the `typeExclusionPredicate` object detailed below.

```
typeExclusionPredicate = [NSPredicate predicateWithFormat:@"
(kMDItemContentType != 'dyn.ah62d4rv4ge81a5dmsr4a')"];
```

The alphanumeric string in the above example represents the file system classification of file types. This is the exclusion predicate for a Mac OS Preference file. The final predicate was created by combining all of the individual predicates together.

Stage 2: Query Configuration

When the query expression was defined, the query itself was configured. This involved defining the scope of the query and providing a method of sorting the returned data.

The scope of the query defines in which locations the query will search to match metadata attributes. The scope of the query was defined using the `setSearchScope` method. If the search scope is not specified, the system will look through the entire file system to match the query expression. There is no real benefit gained from searching the entire file system as each user's data is confined to their individual user directory. Searching the

entire file system dramatically decreases the performance of the system. Non-functional requirement 9 states that the result retrieval must be as fast as the current Spotlight implementation, therefore, the decision was made to limit the search scope to the user's home directory. This was executed in the following way:

```
[_query setSearchScopes:[NSArray arrayWithObjects:
NSMetadataQueryUserHomeScope, nil]];
```

The constant `NSMetadataQueryUserHomeScope` limits the search to the user's home directory. A n added advantage of setting the scope of the search to the user's home directory is that this also excludes default system folders which typically contain application and OS configuration files. This meant that the number of irrelevant files returned by the system decreased.

The result categorisation was implemented in two stages, grouping and sorting of the results. The grouping of the results was based on the `kMDItemContentType` attribute which defines the type of file returned and was executed using the `setGroupingAttributes` method of the `NSMetadataQuery` class.

```
[_query setGroupingAttributes:[NSArray arrayWithObjects:
(id)kMDItemContentType, nil]];
```

An array containing grouping attributes is passed to the `setGroupingAttributes` method of the query. The `kMDItemContentType` attribute was the only grouping attribute specified, as further grouping of results would make the results view over complex.

The sorting of the results was based on custom categorisation of the grouping attributes. The standard grouping attributes originally returned values such as `com.apple.mail.emlx` which is the `kMDItemContentType` for an email, however, this is meaningless to the user. The `metadataQuery:replacementValueForAttribute:value` method was used to customise the result categories.

```
- (id)metadataQuery:(NSMetadataQuery *)query replacementValueForAttribute:
(NSString *)attrName value:(id)attrValue {

if([attrValue isEqualToString:@"com.apple.safari.bookmark"])
return NSLocalizedString(@"Bookmarks", @"Bookmark category");
}
```

In the above example, if the content type matched the bookmark content type, a localised string was returned which changed the category name displayed to the user from `com. apple. safari. bookmark` to `Bookmark`. One of the benefits of using this categorisation method is that only the categories that contain results are returned, for example, if no music files are matched, the music category is not displayed. This reduces the amount of information on the screen and allows the user to only view particular categories.

Stage 3: Query Execution

When the query object has been created and configured, it has to be executed when

the user performs a search. This was performed using the simple `[_query startQuery]` method, which was called after the `NSPredicate` objects, detailed above, were created based on the user's query string.

Once the query has been executed, it has two phases, the results gathering phase and the live update phase. The results gathering phase returns results from the system store in batches and the live update phase runs after the initial results gathering phase and updates the results if a new file is added to the file system. The live update phase runs indefinitely as a background process, unless the user stops the process using a toolbar button.

When the query has been executed, it is the role of the controller and view layers to display the information in a meaningful way.

Saved Query Object

The saved query object model was implemented using an XML file to store the user's saved query. When the user selected to save a search, the system created an `NSMutable Dictionary` object and stored the current `_queryString` as a value in that dictionary. The system would then read the saved XML file and place the saved value in a table view. This method was chosen as it was the most simplistic method of saving query objects.

Gather View Data Source

The gather view data source stores the name, date, and location of the gathered file. The model was based on sample code taken from a drag and drop table view tutorial found at `http://homepage.mac.com/mmalc/CocoaExamples/controllers.html`. The reason this code was used was because it fitted the needs of the gather view perfectly. The original code was designed to support the collection of Internet based bookmarks, but was adapted for this project to support collection of all file types. One of the key features of the model is that it supports drag and drop direct manipulation. When a file is dragged, the information for that file is stored in a custom `URL Pasteboard` and when the file is dropped, the gather view accesses this pasteboard to display the required information. Much of the work was implemented through the use of bindings, discussed in section 5.4.4.

5.4.3 View Objects

The implementation of the view objects from Figure 4.2 are discussed.

The UI consists of a `mainmenu.nib` file which contains all of the UI components and resources, such as the windows and the main menu. The nib file also contains information about how the UI components are all connected. The design of the interface was based on the redesign of the GUI prototype created in Section 4.6.2. The main layout of the

interface closely followed the layout of the GUI prototype, however, the Apple Interface Design Guidelines were applied at this stage. This is because the final prototype needed to reflect the look and feel of a Macintosh application. If the prototype was left in its original GUI form, it would not be consistent with other applications across the OS. Any important changes from the GUI prototype are discussed.

Window Type

The first consideration was the type of windows in the application. The design guidelines state that there are four types of windows: document windows, application windows, utility windows and dialogue boxes. The system used an application window with a drawer for the main view and a floating utility window for the additional results information view.

The design guidelines offer two main distinctions between window types, brushed metal windows and standard aqua windows. Brushed metal windows are used for applications which provide an interface for a digital peripheral, something which represents a real world device, or a source list based single-windowed application. The application partially falls into the final category, therefore the brushed metal interface could have been used. However, the decision was made to use the standard 'unified' Aqua interface, because the brushed metal interface makes the application appear 'too heavy'. Apple are moving away from brushed metal to a 'dark unified' interface with the latest release of the OS, therefore creating a brushed metal application would appear dated and inconsistent with the rest of the OS.

The overall design of the application changed from a single window with a collection of objects, to a partitioned single window representing different sections. This UI design was chosen, to imitate the look and feel of OS applications such as 'Mail.app' and 'iTunes.app' and to provide the user with a clear, structured layout. The main interface was a border-less window divided into 4 sections, a side bar, a main view, a sub view and a toolbar. Each separate section was separated using a black separator line, also common with standard OS applications, such as Mail.app and iTunes.app. Figure 5.1 represents the redesigned main window.

Query View

The query view object was identical to the redesigned prototype, however, it was moved to the main toolbar along with the result manipulation controls so that all of the application controls were located in one place.

Results View

As shown in figure 5.1 the results view is the main component of the UI and it has been placed in the main view of the application. The results view is identical to the results view object of the redesigned GUI, however, when the view is populated with

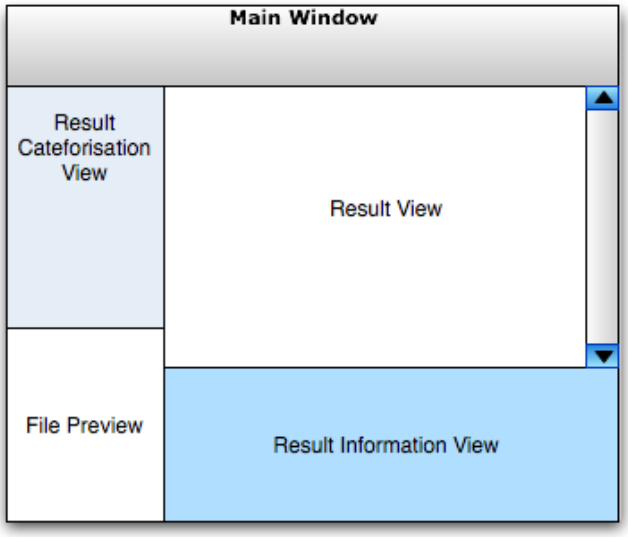

Figure 5.1: Redesigned Main Window

search results, the user is able to sort the results by name or by location, by clicking the column headings. This was derived from functional requirement 13, where the user should be able to apply a degree of custom sorting to results. Additionally, the results view performs one main function, when a result is double clicked, the file itself opens. This functionality was derived from section 4.6 as during the evaluations, users noted that the use of an open button to open a file was awkward and not intuitive. When a result is double clicked, the text colour for that entire row changes from black to deep purple to indicate that the results has already been opened. This idea was originally disregarded as it may introduce too many different colours into the system, however, users identified in section 4.6 that using a check-box in the column was a waste of space and looked out of place with the rest of the interface.

Result Information View

As shown in figure 5.1 the main result information view is positioned below the main results view. There information view is similar to the redesigned prototype, however, the information attributes displayed have been renamed to adhere to the naming conventions used across the OS. For example, instead of using the word 'location' for the file-path of the result, the word 'where' is used to indicate to the user where the result is stored. The file preview has been moved from within the information view, to the sidebar below the result categorisation view. This is because it makes much better use of the space in the sidebar and imitates the iTunes.app sidebar appearance. The additional information view was not changed from the redesigned prototype.

Results Categorisation View

The results categorisation did not change from the redesigned prototype, however, it was moved to the sidebar.

Gather View

The gather view was redesigned to support the drag and drop of any file type from any location. One of the major omission of the GUI prototype was the ability to delete an item from the gather board. The redesign implemented a 'remove item' button which was situated at the bottom of the gather drawer, which simply removed the selected item when pressed.

Saved Query View

The save query view did not change, however, a save search icon was positioned along side the search view in the toolbar.

5.4.4 Controller

The control layer initially proved to be the most problematic layer of the implementation. The controller consisted of a single implementation file, `'Controller.m'` and category, `'ToolbarCategory.m'`. The role of the controller was to link the interface objects of the view layer to the underlying objects of the model layer. Application controllers can have different roles, a mediating controller and a co-ordinating controller. A mediating controller design was adopted for the implementation as it controls the flow of data between model object and view objects instead of controlling the functioning of the entire application. Using a mediating controller, allowed most of the controller functions to be implemented through a new Cocoa technology called 'bindings'.

Bindings

Due to the importance of bindings in this implementation, a discussion of how they work is necessary. The key benefits of using bindings are that they take out much of the code implementation of the control layer and facilitate the communication between model and view objects.

Bindings allow the developer to create relationships between view objects and model objects without having to programmatically create these relationships. For example, if a binding was made between a view object which displays an integer and a model object which stores an integer, the controller would automatically update one object if the other object changed.

Cocoa objects have particular components, which are referred to as `properties`, these properties are divided into two types, `attributes` and `relationships`. Attributes are characteristics of the object itself, for example, an attribute of a `NSMetadataQuery` object would be a query result. The relationships defines the relationship between one object and another, for example, the relationship between the `NSPredicate` object and the `NSMetadataQuery` object. Each property of an object is called a `key` and can be accessed through its `keypath`.

Object Class	Object	Key	Keypath
NSMetadataQuery	_query	resultCount	query.resultCount

Table 5.1: Object keypath

Table 5.4.4 shows the relationship between an object and its keypath. In this example, the `_query` object is the query object itself and the `resultCount` key returns the number of results matched by the query. To access this key through bindings, the key path would be `query.resultCount`. However, this raises a question, how can the key path for the `_query.resultCount` key be accessed through the path `query.resultCount` when the name of the object is `_query`? This leads on the concept of Key-Value Coding.

Key-value Coding

Key-value coding is a technique of accessing properties of a particular object by using strings instead of programmatically invoking the accessor (getter and setter) methods for that object. This is done by adhering to particular naming conventions when creating the accessor methods. When naming the object setter method, the name of the object must be used, without the underscore and prefixed with `set`. For example, the setter method for the `_queryString` object is `setQueryString`. The getter method must be the name of the object without the underscore and the object name itself must be prefixed with an underscore. The `_queryString` object methods are used to give a simplified definition.

```
}
NSString *_queryString;

- (void)setQueryString: (NSString *)value {
if (_queryString !=value) {
[_queryString release];
_queryString = [value copy];
[self createQueryPredicate];
}
}

- (NSString *)queryString {
```

```
return [[_queryString copy] autorelease];
}
```

The above sample displays the three methods used to store the value entered into the query view object, in the query object model. `_queryString` effectively represents the query the user enters into the query object in the view layer. The `setQueryString` method is designed to be passed a string object as an argument and store this value to create the `NSPredicate` object, which in turn defines the query expression and executes the query. This is where the concepts of key-value coding and bindings are crucial.

The query view object is bound to the keypath `queryString` which subsequently passes the query entered in the query view, to the `_queryString` object, where it is stored and the query is ultimately executed. By specifying the key path as `queryString`, key value coding conventions are employed to set the value of the object. The program will initially search for a setter method by the name `setQueryString`, to set the value of the object. If this is not found, the program will look for the getter method `queryString` to return the value of the `_queryString` object and if this is not found, it will return the texttt_queryString object itself. In the above example, the `setQueryString` method was executed and the user's query was stored in the `_queryString` object. These naming conventions answer the above question, 'how can the key path for the `_query.resultCount` key be accessed through the path `query.resultCount` when the name of the object is `_query`?'

It is difficult to understand how this can be implemented without the need for any code implementation, therefore figure 5.2 demonstrates the Interface Builder binding interface for the query object view.

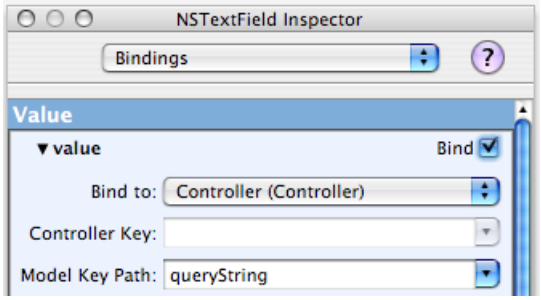

Figure 5.2: Interface Builder Bindings Interface

The above example aims to demonstrate the concept behind bindings and how the technique was applied in the system, however, the main use of bindings in the system was to bind the table columns of the result and result categorisation views to individual metadata attributes. This posed a more difficult problem to solve, as when the `NSMetadataQuery` ran, it returned multiple results which needed to be arranged and displayed to the user. The solution was to use an array controller to manage all of the results returned by the query object.

When an `NSMetadataQuery` finds query matches, it returns batches of `NSMetadataItem` objects, therefore an array controller, named `Query Results`, of class `NSMetadataItem`

was created to handle all of the results returned by the query object. The content of the array was bound to the `query.results` key, where the `results` attribute are `metadataitems`. A benefit of using this it method is that the array controller manages the content of the array without having to implement a single line of code.

The `Query Results` array controller simply returns matched metadataitems, it does not manipulate the results or display any result information. The next stage was to group the results in some sort of meaningful way, therefore another array controller, `Item Kind Grouping` was implemented. Cocoa provides an object which handles grouping of metadataitems, called `NSMetadataQueryResultGroup`, therefore the array controller was set to the `NSMetadataQueryResultGroup` class. The role of this array was to interact with the `NSMetadataItems` to sort them into the correct groups identified by the `setGroupingAttributes` method detailed above. The content array of this array controller was set to the `groupedResults` attribute of the `NSMetadataQueryResultGroup` object. To display the query groups in the result categorisation view, the result categorisation view was bound to the `Item Kind Grouping` controller and the key path was set to `value`. `Value` is an attribute of the `NSMetadataQueryResultGroup`, which returns the value of each individual `kMDItemContentType` configured in the model layer using the `setGroupingAttributes` method, detailed above. The grouping attributes were then customised using the `metadataQuery:replacementValueForAttribute:value` method, also detailed above.

These two array controllers return the query results and sort them into the appropriate custom groupings, however, they do not display the results in the results view. To do this, the final array controller was created, called `MD Item Attributes`. The role of this array controller was to display the desired metadata attributes of all of the results returned from the `Query Results` array controller. As this array controller needed to store and display metadataitem attributes, a mutable dictionary was used as the array controller class. The dictionary stores chosen metadata attributes, such as `kMDItemDisplayName` to display for each metadataitem. Since the metadataitems need to be sorted into groups, the `MD Item Attributes` content is bound to the `Item Kind Grouping` array controller and the key path is bound to `results` which returns the query results from the selected group.

The final stage was to bind each result view column to the `MD Item Attributes` array controller with the key path set to the relevant metadatattribute to be displayed.

Figure 5.3 displays the first fully functioning iteration of Searchlight.

5.5 Conclusion

This chapter detailed the implementation decisions made and methodologies adopted during the implementation of the first iteration of Searchlight. The work reported in this chapter has addressed the personal aims set out in Chapter 1 in a manner that is consistent with the overall IR and IG investigation.

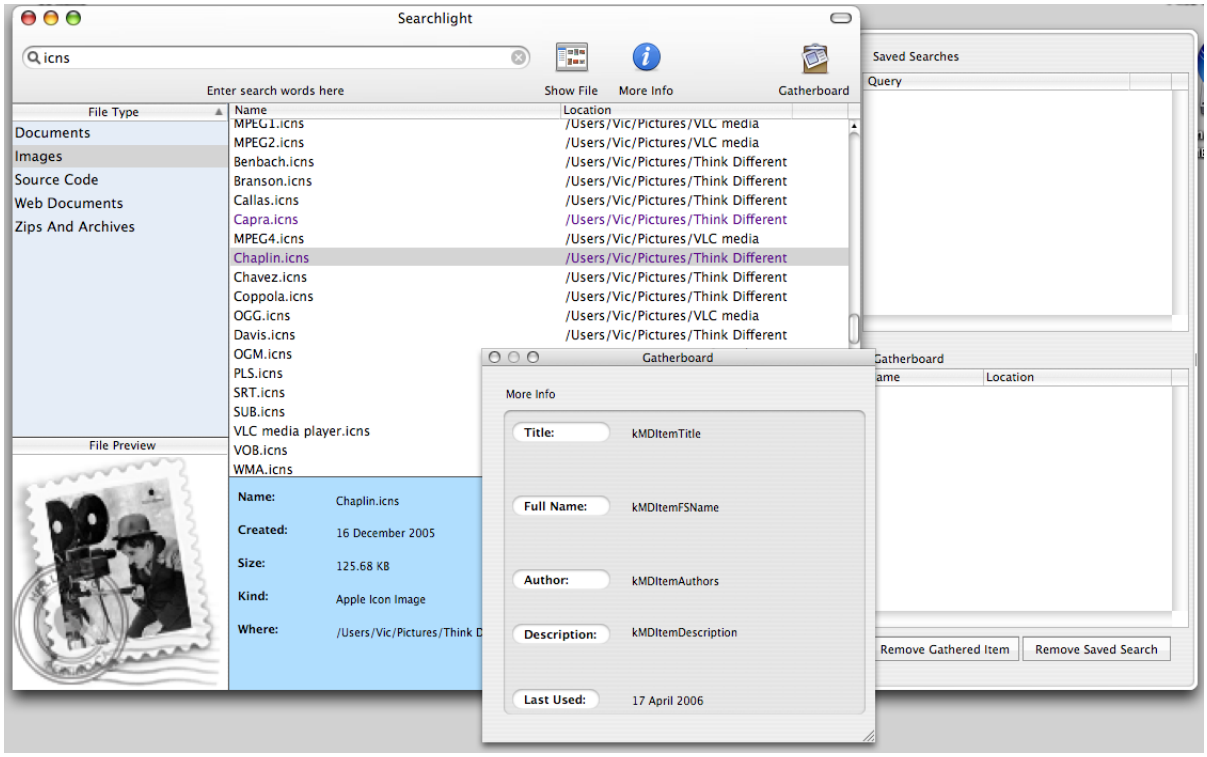

Figure 5.3: Prototype 2: Searchlight

Chapter 6

Evaluation of Searchlight and Generic IR and IG Requirements

6.1 Introduction

Chapter 5 discussed the low level design of the system leading to the development of the Searchlight, the first fully functioning prototype. This chapter discusses the user evaluations which lead to the redesign of the second prototype and the introduction of 'Gathertron', together with reflections on the validity of the IR and IG requirements and principles

6.2 Methodology

Evaluations have been used throughout the project to ensure that at each major stage of creating a user centred IR and IG system, the design matched the needs of the user. The majority of the testing and evaluation is discussed in this chapter. The prototype testing was conducted using the same co-operative evaluation technique used throughout the project. The evaluations focussed on discovering usability and UI design problems, improvements that could be made to the system and any required additional features.

As the Searchlight is a conceptual prototype which concentrates on supporting the core areas of IR and IG, functional and structural testing techniques were not adopted during the testing process. Functional testing, such as black box testing could have been used to verify the system produced the correct outputs based on a set of inputs. However, this would provide data on whether the system functions correctly, but would not offer any information regarding the usability and efficacy of the concepts.

White box testing was carried out at object, unit and integration level progressively during the build. This chapter focusses on the interaction model embedded into Searchlight for IR and IG activities

Figure 6.1 illustrates the testing and evaluation process from the first prototype to the distribution of the final prototype. The term distribution does not mean distribution in the normal sense of distributing a commercial application, rather the distribution of the application to a large Mac OS X 10.4 audience. This was used as a method of functional testing and to gain further feedback from a larger user group.

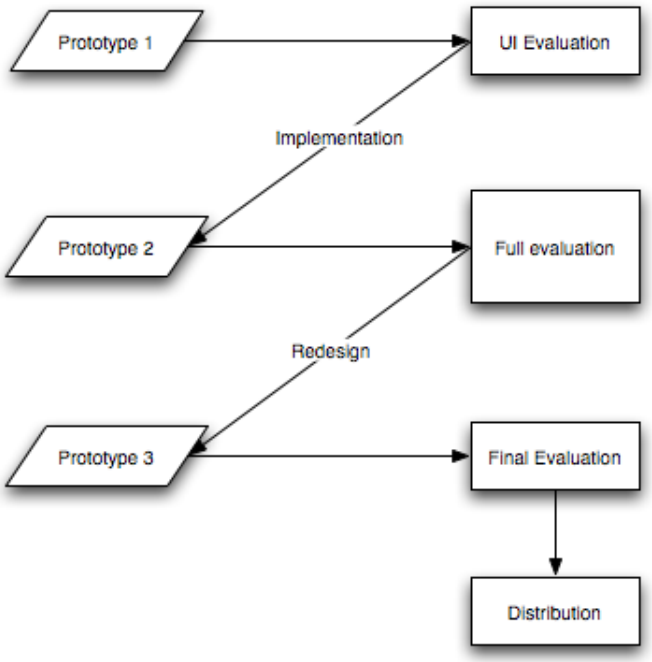

Figure 6.1: Evaluation and Testing Methodology

The full evaluation process in figure 6.1 is depicted as a larger process than the other evaluation stages, as the main body of the prototype evaluations took place in this stage. This was due to time and resource constraints on the project, as it was important to ensure that there was sufficient time to make changes to Searchlight. If the main prototype testing was carried out after implementation of the third prototype, there would not have been sufficient time to implement the suggested improvements.

6.3 Prototype Evaluations

The prototype evaluation adopted the D E C I D E framework as described in detail in Section 3.5.1. Only the key stages are discussed for this evaluation. Many of the same

constraints and ethical issues which exist with the evaluations in the requirements stage also exist with the evaluations during the testing phase.

The goals of the evaluation were as follows:

- Identify usability problems with the prototype
- Identify conceptual problems with the prototype
- Identify additional features or improvements

Five participants were involved in the evaluations, the majority of whom were involved in the evaluation reported in Section 4.6.

The evaluations commenced with the evaluator demonstrating how to use the main features of the system and participants were told that they had to use the system to solve a series of information problems. The problem tasks were all aimed at evaluating the different components of the system and were similar in style to the tasks used in section 4.6. However, this time, participants were all required to use Searchlight to solve the problem. During the evaluations, participants were prompted to offer their opinions and were actively encouraged to identify any constraints with the system.

Table B.1 in Appendix B.1 details each problem task along with their respective domains and aims. Problems 1 and 2 are aimed at assessing whether participants choose to use the IG components of the system without being directly instructed to do so. Problems 3-6 target particular components of the system and aim to test that the user is able to use the key features effectively with little instruction.

As the volume of data obtained from the evaluations was less than in the requirements gathering evaluations, the key points could be placed into a tabular format, identifying usability constraints, conceptual constraints and suggested improvements. Each constraint or improvement is accompanied with a description and suggested design criteria.

6.4 Evaluation Results

Tables 6.1 to 6.3 detail the main results from the evaluations.

Tasks 1 and 2 were successful from a conceptual point of view, as all users attempted to use the IG component to save links to relevant information during their search for information. The IR component of the system also proved to be successful as users automatically chose particular result categories and quickly chose results based on the filename, preview and the location of the file. A surprising constraint with the results view was the negative feedback on the text colour change feature. When the participants selected a result and the colour changed, they were confused as to what this meant, and instead of indicating that a file had already been clicked, it seemed to make the result stand out more.

Table 6.1: Results Part 1, Interface and Usability Issues

#	Component	Description	Design Criteria
1	Query view feedback	Users identified a lack of feedback during query execution and post execution	Provide indication of query in progress, detail number of results found and possibly display the query relating to the results in the results view
2	Gather View	Users were not able to use the gather view effectively as they had to open a drawer to drag files to the view	Place the gather view in a floating utility window
3	Gather View	When the user shows a file in Finder, the gather view window is automatically hidden and the user is not able to drag the file onto the gather view	Ensure gather view always remains visible, unless otherwise instructed by the user
4	Saved Query View	Placing the saved query view in the same drawer as the gather view caused problems. Also, when the user saved a search, there was no feedback, as the query would be added automatically to a list in the drawer	Provide a separate window for saved searches so the user can see when a search is saved and where it is saved to
5	Additional Info View	Users felt that the more info view was not important enough to have a separate window. Also, the window would take the application focus when opened, which meant the user had to close the window before being able to continue using the system	Move the additional info view to a drawer
6	Gather View	When users attempted to reaccess saved information, if the filename of the gathered information offered little semantic meaning, the participant could not remember exactly what the information was about	Provide a method of tagging or labelling the gathered information.

Table 6.2: Results Part 2, Conceptual Constraints

#	Component	Description	Design Criteria
1	Saved Query View	Users were confused with the process of saving searches. The idea of saving the query to a drawer and choosing the saved query from a list re-executing the query without using the main query view was not intuitive	Possibly use a recent searches list in the query view to be consistent with other applications, such as Safari
2	Results View, Result selection Feedback	The changing text colour feature of the results view proved to be inconsistent and confused the user. The text colour would change when a user selected a result, but it did not provide the user with any additional information. Users said that the changing text colour was confusing and did not match their expectations, as the were not sure if it was to indicate that a result was relevant, or the result had already been opened. As this feature caused confusion with users, its inclusion must be considered	Remove text highlighting feature

Table 6.3: Results Part 3, Suggested Improvements

#	Component	Description	Design Criteria
1	Main Window	Several users mentioned that the main window was too wide	Reduce the size of the main window
2	Results view, Information View	The result location attribute was repeated in the results and information view, poor use of space	Remove location attribute from results view
3	Query View	The query view is too large and should be on the right of the main window	Reduce the length of query view and move it to the right side of the window to be consistent with search facilities in other OS applications
4	Results categorisation view	The results view does not contain a category which contains all search results. Users suggested that this may be a useful category	Add a category to the results categorisation view which contains all search results
5	Results view, gather view	Users identified that it could be useful to drag results directly from the results view to the gather view	Enable drag and drop functionality from the results view to the gather view

This can be related to the IF principle in Section 2.8.1 as the aim of changing the text colour was to improve the trace between information patches or items. In theory, the idea of indicating which results have been selected, to decrease time spent searching in exhausted areas, seems effective, however, in practise this was not the case.

Even though tasks 1 and 2 were successful from a conceptual point of view, they were problematic from a usability perspective. Participants experienced trouble using the IG view as the drawer would go out of view when the application was out of focus. Additionally, when re-accessing gathered information, if the filename of the gathered information was ambiguous, the participant could not remember exactly what the document was about. Participants also noted that the use of a split view in the drawer was strange as they were not really sure what the main function of the drawer was. Another constraint with the gather view implementation was the inability to drag results from the results view to the gather view. Two participants attempted to drag results directly from the results view onto the gather view, however, when they were unable to, they opened the file, verified its relevance and then dragged the file itself to the gather view.

The saved query view proved to be extremely unusable, as when the search was saved, the user was not always aware that the search was stored in the drawer and was unsure how to re-access the query, as the query had to be selected from a table view and an associated button had to pressed.

Overall, the participants understood the features of the system, but were not able to use the IG and saved query views, either due to their poor implementations, or insufficient time to learn the application. Users were satisfied with the results categorisation view and did not identify any problems with groupings of the files into particular categories.

6.5 Redesign

The evaluation results identified many areas of the system which required improvements or complete redesign. The redesign focuses on the main components of the system which were redesigned.

6.5.1 Gather View

The gather view underwent a major interface change. The first change was to move the gather view into a separate window which opened through a dedicate toolbar button. This addressed problems 1.2 and 1.3 as the gather view remained visible even when the application was not in focus. An additional column was added to the gather view, which allowed the user to label the gathered item. This addressed problem 1.6 as users could name the gathered items to provide more meaningful definitions to guide the re-access of gathered information. This relates to IG design principle in section 2.8.1, which states

that users must be able to attach semantic meaning to gathered information to aid the recognition process.

Figure 6.2 illustrates the original gather view on the left and the redesigned gather view on the right.

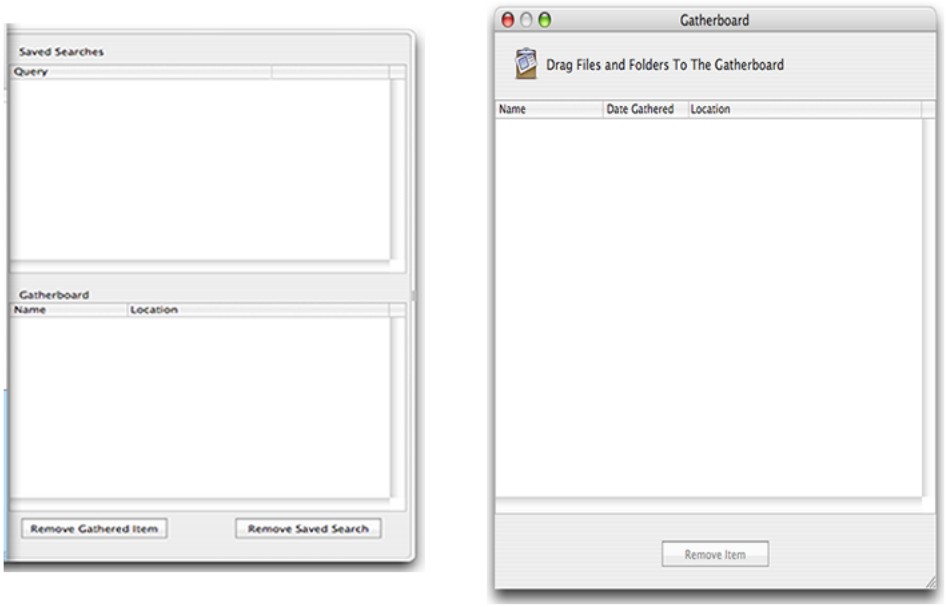

Figure 6.2: Original and Redesigned Gather Views

6.5.2 Saved Query View

The functionality and the UI for the saved query view were completely redesigned. The saved query view was removed and a recent search menu was added to the query view. This stored up to fifty recent searches and provided the user with the option to clear recent searches. When a user executes a query in the query view, the query is automatically added to the drop down list in the search field. This redesign addresses problems 1.4 and 2.1. Using a standard recent query list removes all of the confusion over the function of the saved query view as every query is added to the list, which creates a consistent query saving process. This implementation is common in other applications, therefore the users should not have any problems identifying how to re-access a previous query. The major constraints with this implementation are that the user may wish to access a query which is beyond than the 50 query threshold. Additionally, from a privacy perspective, the user may not wish to save every query. Figure 6.3 shows the original saved query view on the left and the redesigned query view on the right.

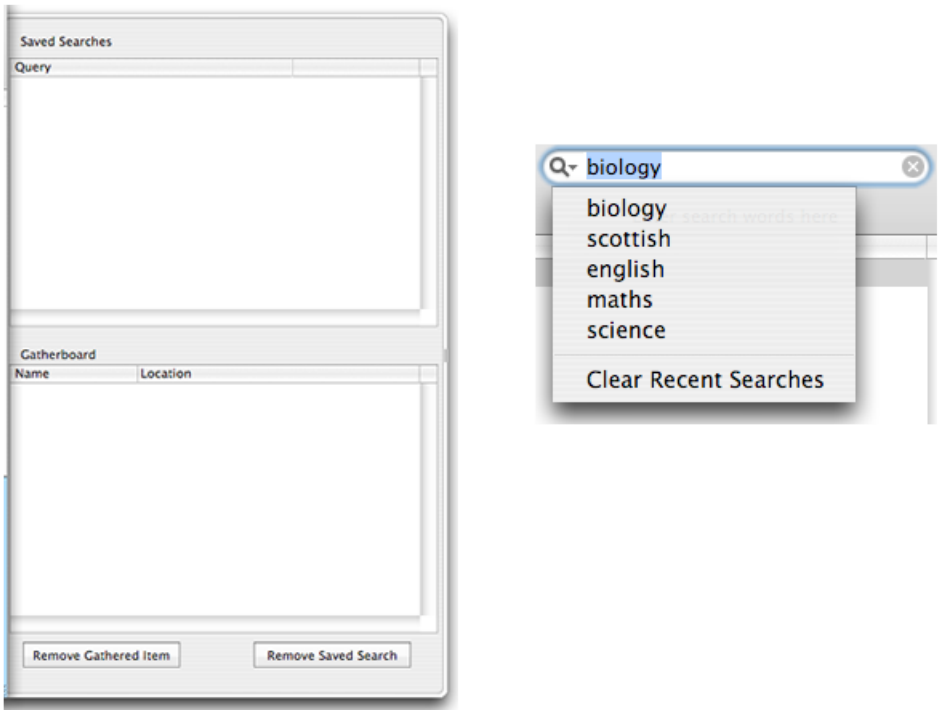

Figure 6.3: Original and Redesigned Saved Query Views

6.5.3 Results View

The location column in the results view was removed as the information was repeated in the information view. This allowed the entire application to be reduced in size, addressing problems 3.1 and 3.2. The text colour changing feature was removed as it offered little benefit to the user's results selection process, this addressed problem 2.2.

An additional feature was added to the results view, a small icon column was added to graphically display the file type for each result. As the location column was removed, the user would need another method of sorting through the results view to select relevant results, therefore the file type icon would allow the user to recognise the file type within each category.

6.5.4 Query Feedback

An additional view was added to the interface, the query feedback view, this was to address problem 1.1. The feedback view used an icon to indicate when the query was in progress, a result count at the end of the query and displayed the query string itself to link the results to the query string. Figure 6.4 shows the query feedback view.

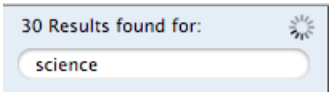

Figure 6.4: Query Feedback View

6.5.5 Aesthetic changes

The additional information view was moved from a floating window to a drawer view, this was to reduce the number of floating windows in the applications. This addressed problem 1.5. The query view in the toolbar was re-arranged and moved to the right, as well as shortened in length. This addressed problem 3.3

One subtle change made in the redesign was the application name change from 'Searchlight' to 'Gathertron'. The icon for the application also changed from a magnifying glass to an image of the Transformer, 'Galvatron' flying towards the screen carrying a basket and a magnifying glass. This change was made because it was important that the application name reflected the concepts the system represented.

Improvements 3.4 and 3.5 were not implemented as the complexity of their respective implementations meant that they could not be implemented in the time available. This is discussed further in section 6.7

Figure 6.5 displays the final prototype

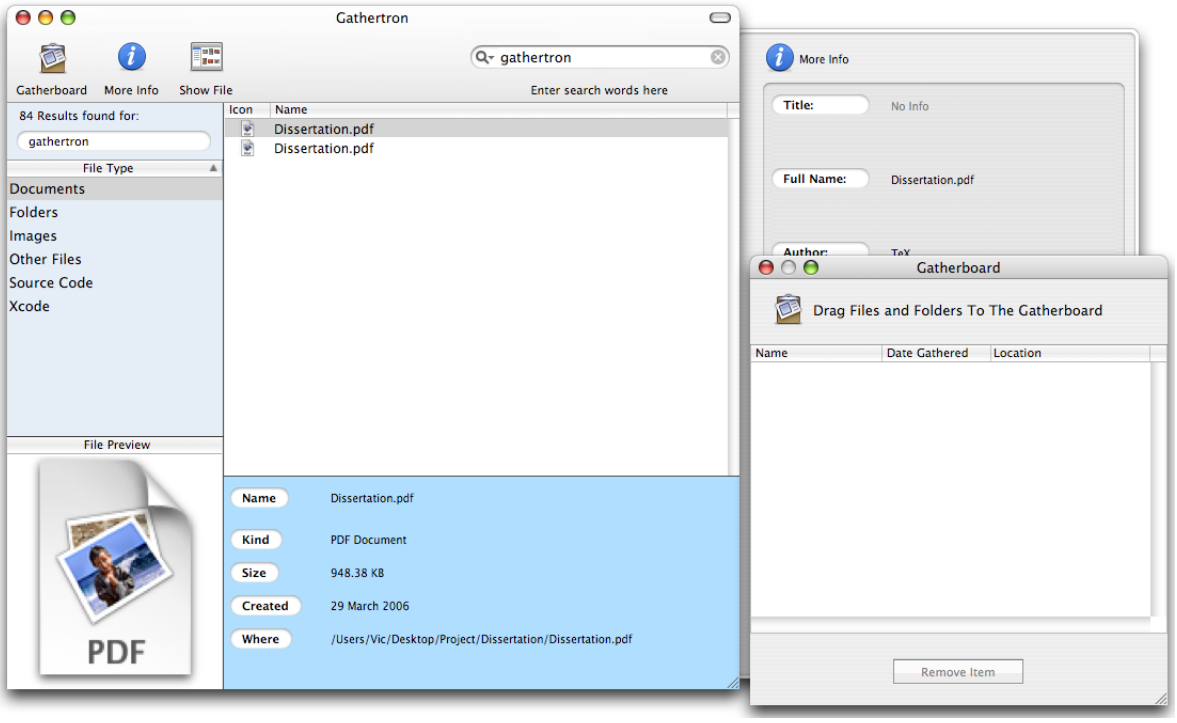

Figure 6.5: Final Prototype

6.6 Redesign Evaluation

Gathertron in figure 6.5 underwent two phases of final testing, informal task based evaluations and distribution testing.

The first stage was to evaluate the changes made to the application to assess whether they solved the problems identified in section 6.4. The evaluations concentrated on the components which underwent the most changes as it was important to ensure that after such a major redesign, the system was usable. Due to time and resource constraints, only three participants were available to carry out the final evaluations. These were the same participants used in the evaluations throughout the design, implementation and evaluation sections. The participants were asked to complete four simple tasks aimed at evaluating the IG and saved query components of the system. The tasks were as follows:

- Gather information about cockroaches
- Gather information about cars or car repairs
- Re-access your gathered information about cockroaches
- Re-access any searches you created about cars

The tasks proved successful at evaluating the redesigned IG and saved query views. The results from the evaluation were very encouraging as participants were able to fully utilise the new implementations. The most important change in the IG view was using a separate window which allowed the participant to close the main window and use the gather view separately to the rest of the system. An unexpected benefit of this was the ability to use the system as an aid for information browsing. One user closed the main window and used the gather view to gather items whilst browsing for information in a particular folder. This functionality was not considered to be inside the scope of an IR and IG system, however, it shows that an IG system is not bound to the domain of IR.

All participants were able to re-access saved queries from the query view without the need for instruction. One participant did instinctively open the gather view to locate the saved query view, however, this was because they had used the previous prototype and assumed it was in the same location.

As the evaluation was aimed at testing the redesigned components, no new problems were raised, however, problems 3.4 and 3.5 from the initial evaluations still existed in the system. These exclusions are discussed in Section 6.7

As a final stage of testing, the application was packaged and distributed to 12 Mac OS X 10.4 users to run over a longer period of time and report any faults with the system. Table 6.6 lists the identified faults.

Table 6.6 Additional System Faults

#	Component	Description	Possible Cause
1	File Preview	When the application is resized, the image in the image view becomes pixelated	Image view is not correctly calculating the correct proportions of the image when resized
2	Query object	The result retrieval time for single letter queries is exponentially longer than any other string.	The query object matches hundreds of files and returns large result batches causing the system response to be slow
3	Query object	Blank query strings can be entered into the query view and the system will return all results which match a blank space character	The query view is functioning correctly as it treats the blank character as a string.
4	Results Categorisation View	If the query is still returning results and the user opens a result, the result category defaults to the first category	The query object is still returning and updating results and defaults to the first category.
5	Query Feedback	If the user deletes the query string in the query view and presses return, the results view stays populated but the feedback view goes blank	The feedback view is bound to the query view, therefore deleting it and pressing return will change the feedback view
6	Information Representation	The size attribute for folders does not display the folder size	Folders are not key value coding compliant for the size metadata attribute
7	Result Categorisation	Not all file types have been accounted for, some files appear in the 'other files' category	Not all file types have been sorted into custom categories

The above problems represent small usability or interface problems with the Gathertron.

As the system is only a prototype, faults of this size are to be expected. None of the faults caused major problems during the evaluations as it was only when users were trying to break the system that they discovered these faults. If Gathertron was to be developed as a commercial application, these faults would have to be addressed.

The distribution testing produced positive results, from stability, usability and aesthetic perspectives. The prototype was also tested on one Intel based Macintosh and no errors were reported

6.7 Conclusion

The implementation techniques, requirements and design principles are evaluated.

6.7.1 Implementation Evaluation

The main criticism with Gathertron was the failure to implement improvements 4 and 5 from table 6.3. Improvement 5, 'Users identified that it could be useful to drag results directly from the results view to the gather view', is discussed first.

During the final prototype evaluations, one of the common user activities was to attempt to drag results directly from the results view to the gather view. Three of the five users involved in the evaluations first attempted to drag results to the gather view and when they realised this was not possible, they looked up the location of the file and dragged it from Finder. This feature was not implemented in the redesigned prototype in section 6.5 as due to the difficulty of the implementation in the time constraints, a fully working solution was not developed.

The main problem with implementing this drag and drop functionality was registering the results view table as a draggable source. When performing drag and drop actions between tables in Cocoa, a drag source and destination must be configured. The drag source was the results view and the destination was the gather view. When dragging a row from a table view, the selected row must be copied to an object called a pasteboard, and the destination accesses this pasteboard to paste the dragged row into the table view. The problem with creating a custom pasteboard for the dragging source was that the source contained metadata items and not standard files, this meant that if the destination source, the gather view, was able to accept this data type, it had to be configured to read this custom pasteboard. This meant that the user would not be able to drag and drop files from any location, as another method would have to be implemented to handle this functionality. This had a definite impact on the usability of the system, as users naturally tried to drag results from the results view as well as dragging files from Finder. This feature would have to be addressed in future work.

Improvement 4, 'the results view does not contain a category which contains all search results', was not implemented as it required a complete redesign of the query view. The reason an 'all results' category was not implemented was because the method which handled the grouping of query results, `setGroupingAttributes`, did not provide a way

of including an 'all results' category. This is because when the method is called, it sorts the results into individual categories based on the file type, and when a result is sorted into one category, it cannot be sorted into another. One possible method of addressing this could be to create two query objects and run them both simultaneously. One query object would return all results and the results would be ungrouped and the other query object would return all grouped results. As stated, this would have meant redesigning the entire query and results views which could cause further errors in the code, thus requiring further retesting. The time taken to develop this feature would outweigh the benefits gained.

This feature suggested demonstrates that users not only require their information space to be subdivided into patches, as the IF and categorisation principles in Section 2.8.1 suggest, they also wish to see the whole collection of results. This could be to gain an overall understanding of the structure of their information space, which ties in with the information browsing principle.

From a functional point of view, the implementation was very solid and no application crashes or memory leaks were experienced. One of the major benefits of the application was the number of relevant results that were returned. The evaluations highlighted that one of the major problems with Spotlight was the number of irrelevant results which impedes the IR process when selecting results. The application limited the number of results by reducing the scope of the search and excluding certain file types. This coupled with the customised categories ensured that the results in each category were meaningful results which the user did not have to constantly ignore. This had a positive impact on the result selection process as users trusted the categorisation of the results to return meaningful data. This relates to the query formulation principle in Section 2.8.1 as users required a higher density of meaningful results from a limited search scope.

The query object of the system was also extremely effective at matching the same results as the standard Spotlight implementation. Non-Requirement 8 states that the system should imitate the Spotlight query functionality and the implementation succeeded in doing this. The system was informally tested against a series of inputs to ensure that the outputs produced were similar to those of the standard Spotlight implementation. The system always returned the same results as Spotlight, minus the exceptions and scope limitations of the search.

One major achievement with the prototype was to implement a system which was intuitive and usable for users of all skill levels. The evaluations demonstrated that users understood the concepts of the system and were able to use all of the features with little or no instruction from the evaluator. This was achieved by adhering to the Apple Design Principles during the design and implementation phases. By adopting a design which was common with other applications across the OS, Mac OS X 10.4 users were immediately able to recognise the key functions and actions of the system.

One criticism of the implementation was the inconsistency of the file preview feature. The file preview displayed a preview of each result to the user. For certain file types, for example, .png files, the system would display a preview of the image, rather than

the file type so the user could immediately identify the image. However, for file types, such as PDF, a generic PDF file icon was used to indicate that result was a PDF file. Finder displays a file preview for PDF files, however, this was not implemented in the system due to time constraints. It would have been ideal to have file previews for all file types, so that the user could immediately recognise the content of each file without having to open the file to verify its relevance.

6.7.2 Requirement and Design Principle Evaluation

The final requirements specification in Appendix A.2.1 evolved throughout the design process as requirements were added and removed, however, only minor changes were made from the original specification. The original specification was based on a combination of the design principles in Section 2.8.1 and user studies in Chapter 3.

A major scope change, or a requirement which was not met, was the inclusion of technical support for the user. Section 2.3.1 in the literature review identified the need for support in the system and this was omitted in the design stage. When the prototype was evaluated in Section 6.4, users identified the lack of feedback in the system to be a problem. Therefore functional requirement 19 was introduced to include query feedback into the system to inform the user of the system state. As the prototype itself was very simple, there was little scope for conceptual support as the main actions in the system were very intuitive. A method of accounting for this lack of support was using carefully named icons and titles in the system. For example, each icon in the toolbar was carefully selected to reflect the users mental model of the action performed. The icon name also reflected the nature of the action, such as 'Show File' instead of 'Reveal in Finder'. Potentially the most complex component in the system, the Gatherview, used the title 'drag files and folders to the gatherboard', to indicate the function of the gatherboard itself. Support could have been included in the system in the form of help documentation, however, as stated in Section 4.2 this was decided to be out of the scope of the prototype.

One of the major requirements additions was functional requirement 18 , 'users must be able to drag results directly from the results view to the gather view'. This requirement was derived from the final prototype evaluations as users stated more than once that this functionality is necessary. The benefit of using this method to gather results is that it removes an extra step from the IG process, the user can retrieve results and store a subset of these results in a particular location. The main disadvantage with this method is that users will gather the results before they have opened the file to verify its relevance. This could lead to the collection of gathered information which is not relevant.

Design Principles Revisited

1. Query Formulation. The final prototype demonstrates that the query principle accurately defined the user's interaction with the system. The query formulation

in the final system supported the formulation of tentative queries which could be reformulated and allowed the user to browse through a list of retrieved results. The scope of the query was also limited to the user's home directory to decrease the number of irrelevant results returned.

2. Query Information Visualisation. The results representation component was built around the use of file previews and the visualisation of important file attributes to support the user's file recognition process.

3. Query Information Clustering and Categorisation. The result categorisation component used faceted categories to organise results into meaningful groups. The use of clustering was decided to be out of the scope of the project as users relied on faceted categorisation during the user studies. The use of clustering also requires a sophisticated algorithm to be developed, which would need a project of its own to develop and evaluate successfully.

4. Information Browsing. This principle was not implemented in the conventional sense of creating a browser and allowing the user to navigate through the file system. A browser was not implemented as it was out of the scope of the IR and IG system and there was not sufficient time to develop a file browser. The concept of information browsing was applicable to the browsing of search results within the system and the concepts of using file thumbnails in the results view was implemented.

5. Information Foraging. Users exhibited IF activities during the initial user studies by exhaustively searching information patches. The requirements identified a move from an IF system which does not support the use of IG, to a system which incorporates IF and IG to exhaustively search and retain information.

6. Collaborative Foraging. This principle was out of the scope of the project as IG and IR in the desktop environment is an individual activity.

7. Information Re-access. The saved query view supported the concept of information re-access. The query view saved the exact query string the user entered to retrieve results, therefore this provided enough meaning for the user to be able to re-access more complex or ambiguous information.

8. Information Gathering. The initial definition of IG, has been redefined during this project. IG in the scope of this project does not allow the user to collect information, it allows them to collect links to this information. IG can be performed as a sub-process of IR, IF and information browsing. The gather view in the final prototype supports the metaphor of gathering information into a container by allowing the user to drag and drop files onto the gatherboard. Labelling this gathered information provides the user with a method of recognising information for re-use at a later date.

The evaluations demonstrate that the majority of the design principles identified in the literature review, were used throughout the project to define the final prototype. These design principles define the characteristics of a user oriented IR and IG system and

further studies using these principles could lead to a generic definition for user oriented IG.

Chapter 7

Conclusions

7.1 Overview

The primary focus of the project was to investigate the role and characteristics of information gathering during the information retrieval process and develop an information retrieval and gathering prototype which supports the needs of the user.

The literature review provided the research into exploratory search, particularly focussing on IR, IF, IG and information browsing. This research identified that there was no framework which characterised an IG system. Therefore, eight design principles were developed which aimed to characterise the users interaction with an IR and IG system.

Two user studies were carried out to investigate IR and IG. The first study evaluated an existing system, Spotlight, to determine usability issues, system constraints and benefits. The second study used a co-operative evaluation technique to identify the characteristics of user oriented IR and IG and the results were verified against the design principles derived in the literature review. A definition of various components of an IR and IG system was developed and a requirements specification was produced based on the evaluation results and the eight design principles.

The design process divided the system architecture into the core system components and the MVC design paradigm was used to design the individual layer objects of each architectural component. The design was guided by the requirements specification and design principles identified in the literature review and an initial GUI prototype, was created and evaluated by a group of Mac OS X 10.4 users. The evaluations lead to a redesign of the GUI prototype and the exclusion of collaborative foraging from the scope of an IR and IG system.

The redesigned GUI prototype was used to guide the implementation of the view layer of the first fully functioning prototype, 'Searchlight'. Searchlight was a native Mac OS X 10.4 prototype implemented in Cocoa and Objective C.

Searchlight was evaluated with the end users of the system and a number of usability

issues, constraints and improvements were discovered. The majority of these issues were resolved leading to the final prototype, renamed 'Gathertron'. Gathertron was re-evaluated and distributed to twelve Mac OS X 10.4 to report any further faults.

7.2 Critical Analysis

The exploratory development process proved successful in supporting analysis and design of an IR and IG prototype. The benefits of involving users in the design and evaluation processes far outweigh the constraints placed on time and resources, as the observation of user IR and IG activities and feedback from user evaluations defined design of the final system.

The users studies proved very successful at producing a requirements specification which characterised an IR and IG system and guided the design and implementation processes. The initial requirements specification proved to be accurate and only minor changes to the specification were made during the entire process.

The co-operative evaluation technique was extremely successful at obtaining large amounts of data regarding peoples search activities. A limitation with the user studies is that they were performed on a small scale, therefore, it is difficult to determine whether the results obtained can be applied to a larger user group.

The design and implementation stages of the project were very successful. The design stage focussed on applying the derived design principles and requirements to design a system which met the user's needs. The implementation stage fulfilled the personal development aims of developing a system, whilst using the appropriate tools and techniques to develop a usable system which characterised IG and IR.

The main obstacle to overcome during the implementation was learning how to develop a Cocoa application, this is because the techniques are radically different from most other programming languages. Much of the development time was devoted to learning the basics of the language and development environment and only when enough research and practise (the creation of many, many example applications) was undertaken, could the development of the final system start. The entire development process was a learning process, as each new feature added to the prototype had to be investigated to discover the appropriate implementation techniques. This was one of the major disadvantages with the implementation process, as towards the end of the development process, when prototypes were evaluated and changes needed to be made, time constraints were even more crucial. One key feature of the system, functional requirement 18, 'users must be able to drag results directly from the results view to the gather view' was not implemented due to the difficulty of the implementation in the time available. One of the major improvements would be to start the development process earlier on in the project, for example, learning the implementation language during the literature review stage.

The prototype evaluations were successful at identifying the usability issues and conceptual constraints with the prototype. However, due to the small scale of the evaluations,

only including five people, it is not possible to determine whether the system would meets the needs of a large audience.

7.3 Information Gathering and Retrieval Defined

The project set out to investigate IG with the aim of developing a set of principles which characterise the user's information gathering process. A set of eight principles were derived from the literature review and were used throughout the project to guide the design of the system. The requirements specification was derived from a combination of these eight principles and user studies. As the requirements specification may change depending on different system implementations, it is the principles which really define an IR and IG system. Below is the finalised framework which characterises the key components of an IR and IG system.

1. Query Formulation. The system must support the formulation of in-exact queries to return a list of results which the user can browse through. The query itself must be limited to exclude irrelevant sources of information which will not return relevant results.

2. Query Visualisation. The system must place emphasis on the use of recognition over recall to retrieve results. File previews and essential result attributes, such as filenames and locations, must be used to aid the result selection process

3. Query Information Categorisation. The system must group results into customised, faceted categories. The categories used in Gathertron were file type categories, however, other categories, if relevant to user search activities can be used.

4. Query Result Browsing. The system must allow a degree of exploration of the information space. This can be in the form of browsing result lists or the provision of a file system browser.

5. Query Result Foraging. The system must combine IG and IF to produce the concept of persistent foraging. The system must allow the user to gather information during IF

6. Query Re-access. The system must provide a method of re-accessing previously executed queries. These queries must also have attached semantic meaning to aid the recognition process when re-accessing queries.

7. Information Gathering. The system must support the collection of links to relevant information sources. These links must have attached semantic meaning to aid the recognition process when re-accessing gathered information. The system must support the gathering metaphor through use of direct manipulation to drag and drop results to the gather area. The system must also support the gathering of results directly from the results source as well as through information browsing.

The Collaborative Foraging principle was omitted from the IR and IG design principles as collaborative foraging does not support the individual process of gathering information. Collaborative foraging is best suited to an information rich, networked environment, where many users actively classify relevant information. This is not indicative of the desktop environment at this current moment in time.

What does Gathertron bring to IG?

Gathertron uses the design principles identified to demonstrate a physical implementation of user oriented information gathering. Due to the limited research into IG, Gathertron can be seen as a leading example of a practical implementation of user oriented IG and can be used as a basis for the development of more advanced implementations in the future.

7.4 Future work

Gathertron is a basis for an IR and IG system, therefore, there are many improvements which could be made.

- The most fundamental improvement would be to implement the gathering of results directly from the results view to the gather view. If there was more time available for the project, this feature could be easily implemented.
- Investigation into the utility of a category which allows the user to view 'all search results' could be carried out. This was not implemented in this system due to time constraints, therefore would be assessed in future work.
- Further research into the effect of document previews on the user's recognition process could be undertaken, with the aim of implementing previews for all types for documents. For example, text summaries of text based documents to identify the content of a file could be investigated.
- The clustering of search results based on relevance or specific grouping attributes could be investigated and implemented.
- The system supports basic exploratory search, however, it does not actively support exploratory information browsing. Gathertron could support the inclusion of a file system browser which allowed the user to choose between query based IR and information browsing.
- Research into explicit information tagging could be undertaken with the aim of investigating the efficacy of users 'tagging' information for re-use. This could tie in with collaborative foraging and be used in networked environments
- The principles of an IR and IG system must be refined with the aim of producing a standardised definition for user centred information gathering.

7.5 Personal Reflection

This project was a success from research and developmental perspectives. I was able to use knowledge learnt from HCI modules to plan and execute successful evaluations. I learnt much about evaluation techniques and how to get the most out of users during observations and how to analyse the data obtained to produce relevant design principles.

The main skill learnt from this project was the ability to develop in Cocoa and Objective C on the Macintosh platform. I was able to learn a lot about Cocoa and implement a sophisticated and complex system in a short time frame. This was a real achievement as the development techniques for Cocoa applications are very different to traditional techniques taught in modern computer science.

The project taught me a lot about time management and the importance of sticking to a structured plan. The magnitude of the final year project is far greater than anything I have previously undertaken and without planning, the results of this project would have been unobtainable.

To conclude, this project was successful in developing a set of characteristics of a generic IR and IG system and applying these to create a usable, experimental prototype to act as a basis for future work.

Bibliography

Apple. In *Apple Human Interface Design Principles*, 1992-2006a. URL `http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines`.

Apple. In *Apple Human Interface Guidelines*, 1992-2006b. URL `http://developer.apple.com/documentation/UserExperience/Conceptual/OSXHIGuidelines`.

Anna Aula and Mika Kaki. Less is more in web search interfaces for older adults. 2005.

Anne Aula, Natalie Jhaveri, and Mika Käki. Information search and re-access strategies of experienced web users. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 583–592, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-046-9. doi: http://doi.acm.org/10.1145/1060745.1060831.

Caldiera V. Basili and D.H Rombach. Goal question metric paradigm. In *J. J. Marciniak (ed.) Encyclopedia of Software Engineering*. John Wiley and Sons, 1994.

Giorgio Brajnik, Stefano Mizzaro, and Carlo Tasso. Evaluating user interfaces to information retrieval systems: a case study on user support. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 128–136, New York, NY, USA, 1996. ACM Press. ISBN 0-89791-792-8. doi: http://doi.acm.org/10.1145/243199.243249.

Stuart Card and Peter Pirolli. This term was coined by the palo alto research center (previously xerox parc) by , and colleagues as the result of human-computer interaction research. In *http://www.motive.co.nz/glossary/information-foraging.php*.

Edward Cutrell and Susan T. Dumais. Exploring personal information. In *Communications of the ACM, Vol 49, No. 4*, page 50, 2006.

James Duncan Davidson. In *Learning Cocoa with Objective-C*. O'Reilly, 2004. ISBN 0-596-00301-3.

Jolon Faichney and Ruben Gonzalez. Goldleaf hierarchical document browser. In *AUIC '01: Proceedings of the 2nd Australasian conference on User interface*, pages 13–20, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-0969-X.

II Franklin P. Tamborello and Michael D. Byrne. Information search: the intersection of visual and semantic space. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1821–1824, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-002-7. doi: http://doi.acm.org/10.1145/1056808.1057031.

Simson Garfinkel and Michael Mahoney. In *Building Cocoa Applications: A Step-by-Step Guide*. O'Reilly, 2002. ISBN 0-596-00235-1.

Marti A. Hearst. Clustering versus faceted categories for information exploration. In *Communications of the ACM, Vol 49, No. 4*, page 59, 2006.

Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, and Geri Gay. Accurately interpreting clickthrough data as implicit feedback. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 154–161, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-034-5. doi: http://doi.acm.org/10.1145/1076034.1076063.

Stephen G. Kochan. In *Programming in Objective C*. Sams Publishing, 2004. ISBN 0-672-32586-1.

Flip Korn. A taxonomy of browsing methods: Approaches to the 'lost in concept space' problem. 1996.

F.W. Lancaster. Information retrieval systems: Characteristics, testing and evaluation. 1968.

Gary Marchionini. Exploratory search: From finding to understanding. In *Communications of the ACM, Vol 49, No. 4*, page 41, 2006.

Professor George A. Miller. In *Wordnet: A lexical database for the English language*. URL `http://wordnet.princeton.edu/`.

John C. Mitchell. In *Concepts in Programming Languages*. Cambridge University Press, 2003. ISBN 0-521-78098-5.

P. Mulhem and L. Nigay. Interactive information retrieval systems: from user centered interface design to software design. In *SIGIR '96: Proceedings of the 19th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 326–334, New York, NY, USA, 1996. ACM Press. ISBN 0-89791-792-8. doi: http://doi.acm.org/10.1145/243199.243280.

Peter Pirolli and Stuart K. Card. Information foraging models of browsers for very large document spaces. In *AVI '98: Proceedings of the working conference on Advanced visual interfaces*, pages 83–93, New York, NY, USA, 1998. ACM Press. doi: http://doi.acm.org/10.1145/948496.948509.

Peter Pirolli, Stuart K. Card, and Mija M. Van Der Wege. Visual information foraging in a focus and context visualization. In *CHI '01: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 506–513, New York, NY, USA, 2001. ACM Press. ISBN 1-58113-327-8. doi: http://doi.acm.org/10.1145/365024.365337.

Wanda Pratt, Marti A. Hearst, and Lawrence M. Fagan. A knowledge-based approach to organizing retrieved documents. In *Proceedings of 16th Annual Conference on Artificial Intelligence*, 1999.

Jennifer Preece, Yvonne Rogers, and Helen Sharp. In *Interaction Design: beyond human-computer interaction*. John Wiley & Sons, Inc, 2002. ISBN 0-471-49278-7.

K. Rodden, W. Basalaj, D. Sinclair, and K. Wood. Does organization by similarity assist image browsing? In *Proceedings of SIGCHI 2001*, 2001.

R. Sacks-Davis, P. Wallis, and R. Wilkinson. Using syntactic analysis in a document retrieval system that uses signature files. In *SIGIR '90: Proceedings of the 13th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 179–192, New York, NY, USA, 1990. ACM Press. ISBN 0-89791-408-2. doi: http://doi.acm.org/10.1145/96749.98219.

Stephen J. Schultze. A collaborative foraging approach to web browsing enrichment. In *CHI '02: CHI '02 extended abstracts on Human factors in computing systems*, pages 860–861, New York, NY, USA, 2002. ACM Press. ISBN 1-58113-454-1. doi: http://doi.acm.org/10.1145/506443.506635.

Michael F. Schwartz and Calton Pu. Applying an information gathering architecture to netfind: a white pages tool for a changing and growing internet. pages 426–439, Piscataway, NJ, USA, 1994. IEEE Press. doi: http://dx.doi.org/10.1109/90.336327.

Xuehua Shen, Bin Tan, and ChengXiang Zhai. Context-sensitive information retrieval using implicit feedback. In *SIGIR '05: Proceedings of the 28th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 43–50, New York, NY, USA, 2005. ACM Press. ISBN 1-59593-034-5. doi: http://doi.acm.org/10.1145/1076034.1076045.

A. Smeaton and F. Kelledy. User-chosen phrases in interactive query formulation for information retrieval. 1998. URL `citeseer.ist.psu.edu/smeaton98userchosen.html`.

Ian Sommerville. In *Software Engineering Sixth Edition*, Tottenham Court Road, London, UK, 2001. Addison-Wesley. ISBN 0-201-39815-x.

Ryen White, Bill Kules, Steven M. Drucker, and m.c. schraefel. Supporting exploratory search. In *Communications of the ACM, Vol 49, No. 4*, page 36, 2006.

Baeza Yates and Ribeiro-Neto. Modern information retrieval. Addison Wesley Longman, 1999.

Appendix A

Where possible, the term 'user' will be replaced with a personal pronoun, as 'user' is a rather dispassionate term for labelling a person who uses a computer. People have complex human needs and this can often be overlooked when using a such a generic word.

- IR - Information Retrieval
- IRS - Information Retrieval System
- IS - Information Seeking
- IF - Information Foraging
- IG - Information Gathering
- IGS - Information Gathering System
- OS - Operating System

A.1 Spotlight Analysis Results

The Spotlight Analysis Table lists the main results from the Spotlight analysis interviews

Spotlight Analysis Results

Quote	Description	Domain	Requirement
Id like the ability to do web searches from it	User would like the ability to search pass the search string to Google to perform web searches	SUGGESTION - Web Interface for IR	Provide a method of searching Google from Spotlight

there's that pointless results window	User expresses confusion and disregard for advanced Spotlight view	CONSTRAINT - Spotlight Advanced view - redundant feature?	Differentiate system from advanced Spotlight view
on my mac, as it is slow, it can be annoying to wait for a search result you know exists, but it gives all the other results first	Criticism of speed and relevance matching can have a negative effect if it returns other files even if the user is certain the file is tagged	CONSTRAINT - Speed. CONSTRAINT - Results view, relevance matching	System must be responsive. Provide a consistent, fast method of matching known files
sometimes, searches are saved when you click again, sometimes not	When a user searches for something using Spotlight, the list remains populated for a certain amount of time. When then user clicks back on the Spotlight symbol to create a new search, sometimes the list is populated, sometimes it is not.	CONSTRAINT - Results view inconsistency	System must either retain results in results view, or clear results view for new search
smart folders say source code and spotlight says documents	In the standard Spotlight results view, source code is grouped under Documents, In smart folder view, it says Source Code	CONSTRAINT - Results grouping inconsistency	System must be consistent in results grouping

documents full of stuff that isnt relevant	In the Spotlight results view, the documents category is often more populated with documents such as source code, package files etc	CONSTRAINT - Results grouping	System must provide more user centred results grouping
Ignore preference files, fonts and packages!!!!	Application preference files, fonts and installer packages should not be included in the results	CONSTRAINT - Results grouping	System must exclude preference files, fonts and packages
no way of telling if u have clicked something in list	If user is opening many documents and returning to the same list, there is nothing to indicate a document has already been opened	CONSTRAINT - Results view	System must indicate which files have already been opened from the results view
reveal in Finder doesnt work with top link without using the mouse	Instead of opening each file, often users wish to go to their location. This is not possible using the keyboard with the top hit in the list, the mouse must be use	CONSTRAINT - Results view tasks	System must allow consistent method of accessing the location of a file
Cant save searches from advanced search view	After creating an advanced search, it is not possible to save this	CONSTRAINT - Results re-use	System must allow searches to be saved
if i kick a search off from apple+space, and its taking ages to run, i cant like 'detach' from the search and let it run in the background while i carry on with another task	If Spotlight response is slow, it is not possible to leave it to run in the background and continue with another task	CONSTRAINT - Multitasking	System must support multitasking

umm... it would be nice if it were easier to get to the location of what you have searched for rather than just opening the file you searched for (although i know its possible if you open the bigger spotlight window)	Difficult to reveal a file in Finder	CONSTRAINT - Result Tasks	System must provide an easy method of accessing a file location
i like how fast it is to get stuff	User comments on the overall speed improvement with Spotlight over other systems	BENEFIT - Spotlight speed	System must be as responsive as Spotlight
most of all, its speed, without compromise of accuracy	User comments on how they are able to quickly find files without having to type in the exact name	BENEFIT - Spotlight query construction	System must match or exceed the Spotlight query engine

Table A.1: User Suggested Result Categories

Category	File Type
Documents	PDFs, MS Office documents, rtf, plain text, iWorks documents. Anything text based which isnt source code!
Folders	All folders
Images	All image types
Email and contacts	All email messages
Source Code	All technical documents, e.g. java, .m, .c files etc
Movies	All movie types
Web Documents	Any web related document e.g. html, .php
Music	All music/ files, e.g. mpeg 4

A.2 Requirements Specification

The requirements section is divided into functional and non-functional requirements.

A.2.1 Functional Requirements

Query Formulation Requirements

1	The system must retrieve information from non-exact user queries
DESCRIPTION	The user muse be able to enter a query which they assume to be close to the exact query to retrieve information
SOURCE	Chapter 2, Section 2.8, Design Principle 1
2	The user must be able to browse search result from queries
DESCRIPTION	Users must be able to submit a tentative query and browse the results produced
SOURCE	Chapter 2, Section 2.8, Design Principle 1 and 4
3	Users must be able to reformulate existing queries
DESCRIPTION	The system must support the reformulation of existing queries
SOURCE	Chapter 2, Section 2.8, Design Principle 1. Chapter 3, Section 3.6
4	The system must limit the search scope to the user's home directory
DESCRIPTION	The system must only search in the user's home directory for results to reduce the number of irrelevant results returned
SOURCE	Chapter 2, Section 2.8, Design Principle 1

Query Result Representation Requirements

5	Results must display the file name and file location as a minimum
Description	When search results are returned, attributes for each file must be displayed to aid the user's result selection process. These attributes must be the attributes identified by users during the Spotlight analysis and cooperative evaluations
Source	Chapter 3, Section 3.7.3. User studies identified most popular file attributes
6	File previews must be displayed for each selected result
Description	File previews for certain file types allow users to recognise files without having to read filenames
Source	Chapter 2, Section 2.8, Design Principle 2.
7	Additional file information must be displayed for each selected result
Description	In addition to the simple information displayed with each result, additional information should be available for each selected file
Source	Chapter 3, Section 3.6. During the evaluations, users utilised the advanced Spotlight view to display more file information for results
8	The system shall indicate when a user has already selected a result from the results view
Description	When a user selects a result from the list, the state should change to indicate to the user that this item has already been selected.

Source	Chapter 3, Section 3.6. During the evaluations, several users opened the same file twice, not realising it was the same file.
	Query Result Categorisation Requirements
9	Results must be grouped according to categories identified in the user studies.
Description	The evaluations found that users experienced problems with categorisation of search results. The documents category which users presumed would contain common documents, e.g. PDFs, Word documents etc, were mainly populated with source code files. Groupings can be found in Appendix A.1
Source	Chapter 3, Section 3.4. Chapter 2, Section 2.8, Design Principle 3
10	Results grouping must be consistent for each search
Description	The categories for each search must always be consistent and one file type should not appear in more than one category
Source	Chapter 3, Section 3.4. Chapter 2, Section 2.8, Design Principle 3
11	Categories must exclude file types users identified as not relevant
Description	The search results must exclude file types which users identified as non-relevant
Source	Chapter 3, Section 3.4. The Spotlight analysis identified various file types which users find irrelevant.

Query Result Manipulation Requirements

12	Users must be able to perform basic file tasks
Description	When a search result is returned, the user will wish to perform basic actions with that file
Source	Chapter 3, Section 3.4

13	Users must be able to order their results.
Description	Users may wish to order results alphabetically, or by date etc, to aid browsing through result lists
Source	Chapter 3, Section 3.6. During the evaluations, several users attempted to re-order their search results using the Spotlight Advanced view.

Information Re-access and Gathering Requirements

14	Users must be able to re-access previous searches
Description	Users need to re-access searches that they have created in the past. The system must support re-access of previous searches, whether it is in a recent search list or saving queries
Source	Chapter 2, Section 2.8, Design Principle 7. Chapter 2, Section 3.6

15	Users shall be able to save pointers to files or folders from search results
Description	User shall be able to use the system search results to mark a file as relevant or store a pointer to a relevant file for re-use
Source	Chapter 2, Section 2.8, Design Principle 8. Chapter 2, Section 3.6

16	Users shall be able to re-access gathered information

Description	Users must be able to use the information which has been previously gathered.
Source	Chapter 2, Section 2.8, Design Principle 8. Chapter 2, Section 3.6

17	Users shall be able to gather information through browsing
Description	The system must allow users to gather information by browsing the file system.
Source	Chapter 2, Section 2.8, Design Principle 8

18	Users must be able to drag results directly from the results view to the gather view
Description	The results returned in the results view must be draggable to the gather view, without the need to reveal the location of the file in Finder and drag from there
Source	Chapter 6, Section 6.4

19	Feedback must be provided to inform the user on the state of the system
Description	Users need to be informed of the current state of the system and require feedback such as 'search is running'
Source	Chapter 6, Section 6.4

A.2.2 Non-functional Requirements

Usability and Interface Requirements

1	The system must be designed for users of abilities
Description	The target audience for the system is effectively all Mac OS X 10.4 users, therefore the system must be designed for users of all abilities
Source	Requirements Sources

2	The system must conform to Apple Design Guidelines and Human Interface Principles
Description	When designing Apple application it must comply to basic HI and UI design principles
Source	Apple HI Design Principles

3	The system must be fully usable without the need of support
Description	Most Apple applications are very intuitive and should be usable without having to consult a user guide
Source	The literature review identified the importance of user support in complex systems. This system should be designed to include all necessary features without requiring the user to consult a user guide

Miscellaneous Requirements

4	The system shall be a native Mac OS 10.4 application
Description	The system shall be a dedicated Mac OS 10.4 application, therefore, it must adopt the look and feel of a Mac OS 10.4 application
Source	Project Aims

5	The system must not drain system resources
Description	The system must be able to run for as long as necessary without leaking memory or causing the system to slow down
Source	Apple Design Guidelines.

6	The system must be a Universal Application
Description	The system must be designed to run on both PowerPC and Intel Macs
Source	Hardware Analysis

7	Searches must be able to run in the background
Description	The system must be able to run in the background until the user needs to gather information. It must also be able to run queries whilst the focus is on another application.
Source	The Spotlight analysis identified the constraint of being bound to a search window

8	The system must return the same results as the current Spotlight implementation, excluding those out of the search scope
Description	Users identified the results Spotlight retrieves to be accurate, but there are just too many irrelevant files. The system must return the same set of relevant files, whilst excluding the irrelevant results.

Source	Chapter 3, Section 3.4

9	The retrieval speed of the system must be as fast, or faster than the current Spotlight implementation
Description	Users identified one of the major benefits of Spotlight is the speed of result retrieval, therefore the system must be as fast Spotlights
Source	Spotlight Analysis Results Appendix A.1

Appendix B

B.1 Problem Task Analysis

The table below describes the problems tasks created for the users to address in the prototype evaluations

Evaluation Problem Tasks

1	
Scenario	You are writing a document about mice and remember that you have a few documents saved somwhere on your computer, which contain information about house mice.
Task	Gather information about house mice
Domain	IR and IG components
Aims	To evaluate the efficacy of the IR and IG components. To assess whether users utilise the gather view to store information.

2	
Scenario	A friend has recently become interested with 'badger watching and asks you for any information or tips about badgers in general.
Task	Gather information about badgers
Domain	IR and IG

Aims	To evaluate the efficacy of the IR and IG components. To assess whether users utilise the gather view to store information.
3	
Scenario	You are writing another document about house mice and remember that you have previously gathered information on the subject
Task	Find the information you gathered about house mice
Domain	IG
Aims	To evaluate the retrieval of previously gathered information
4	
Task	Search for the word 'bumper' and save your search.
Domain	Query re-access
Aims	To evaluate whether users understand the save search process
5	
Scenario	You are writing another document about badgers and remember that you have previously gathered information on the subject
Task	Find the information you gathered about badgers
Domain	IG
Aims	To evaluate the retrieval of previously gathered information, but to also assess how the users looks through the gathered information for a particular item of information
6	
Task	Re-access your 'bumper' search

Domain	Query Re-access
Aims	To evaluate whether the users can re-access saved searches effectively

Appendix C

C.1 Gathertron Icon Creation

Figure C.1 illustrates the sources used to create the final Gathertron icon

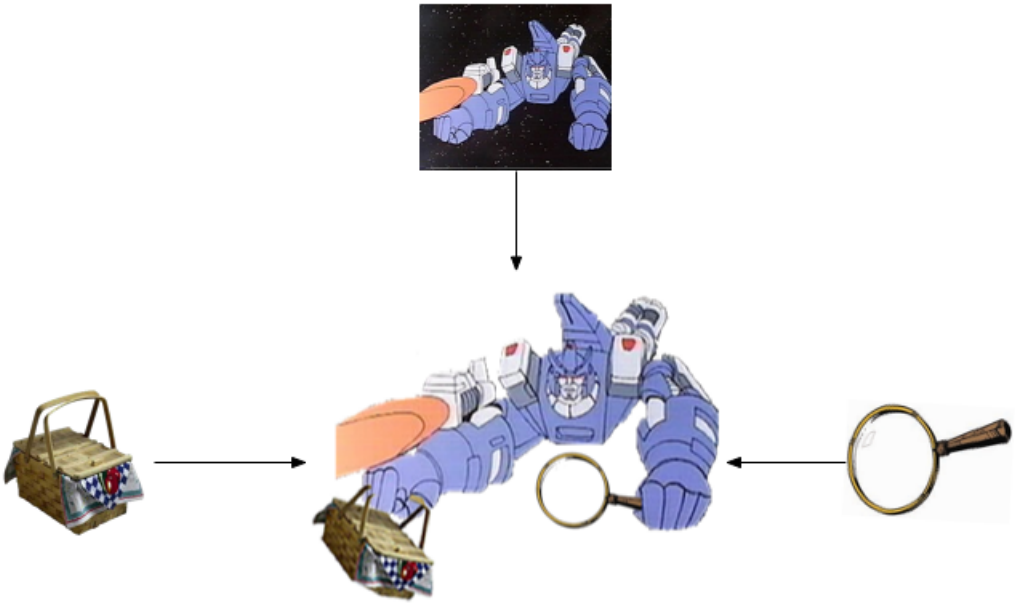

Figure C.1: Gathertron Icon Sources