SOPHIE desktop tools supporting framework development

by

Jiayi Li

A Dissertation submitted to the University of Dublin,

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science (Mobile and Ubiquitous Computing)

University of Dublin, Trinity College

September 2010

Declaration

I declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed: _____

Jiayi Li

11th September 2010

Permission to Lend and/or Copy

I agree that Trinity College Library may lend or copy this dissertation upon request.

Signed: _____

Jiayi Li

11th September 2010

Acknowledgement

I would like to thank my parents for funding my studies and supporting me during all these years.

I would also like to thank my supervisor Stephen Barrett, for guiding me through this project; for his knowledge that makes the project different; for his personality that changes me. Also I would like to thank Xiaobing Xiao, Luca Longo and David Guerin. They've given me a lot of help and support all through my dissertation.

I would like to thank everyone in Ubicomp, I can never do this alone.

In general I would like to thank all the people I love. Without them any of this would make sense.

Jiayi Li

University of Dublin, Trinity College

September 2010

SOPHIE desktop tools supporting framework development

Jiayi Li

University of Dublin, Trinity College, 2010

Supervisor: Stephen Barrett

SOPHIE is a developing novel peer to peer based search system, employing a non-invasive Trust based approach to information analysis and gathering to the problem of delivering more useful web search ranking. The architecture of the system consists of: a peer client, a shared decentralized database and a trust engine DANTE. This project focuses on the functionality and scalability development of the peer clients. One of the key challenges for SOPHIE is to collect user actions implicitly on the peer client.

The current solution to date provides support for the interrogation of user activity only for web browsing, and specifically for later variants of the Firefox Web browser. This project presents a framework solution for the support of arbitrary browser technology that would be specialized within the content of the project for key browsers such as Microsoft Internet Explorer variants, and Apple's Safari amongst others.

Additionally, in the area of enterprise search, it is apparent that other desktop tools (such as word processors, e-mail tools, postscript viewers and etc) are relevant in the access of corporate information. Therefore integration with those desktop tolls is also supported within the same framework.

Table of content

List of Figures

Chapter 1

Introduction

SOPHIE is a developing novel peer to peer based search system, employing a non-invasive Trust based approach to information analysis and gathering to the problem of delivering more useful web search ranking. The architecture of the system consists of: a peer client, a shared decentralized database and a trust engine DANTE. This project focuses on the functionality and scalability development of the peer clients.

1.1 Motivation

This project proposes to develop a desktop framework that is capable of supporting integration with a wide range of desktop tools. This solution is based on the current SOPHIE client side design.

The current solution to date provides support for the interrogation of user activity only for web browsing, and specifically for later variants of the Firefox Web browser. This project propose to develop a framework solution for the support of arbitrary browser technology that would be specialized within the content of the project for key browsers such as Microsoft Internet Explorer variants, and Apple's Safari amongst others. Additionally, in the area of enterprise search, it is apparent that other desktop tools (such as word processors, e-mail tools, postscript viewers and etc) are relevant in the access of corporate information. Therefore it is necessary to support integration with those desktop tolls within the same browsing framework.

1.2 Project aims

As mentioned previously, this project aims to develop a desktop framework that supports integration with a wide range of desktop tools. The aim can be reached by pursuing three different goals.

1. Analyze the core functionalities shared by various desktop tools. SOPHIE desktop tool extensions are supposed to deliver very similar functionalities. The first goal aims to analyze those similar functionalities and encapsulate them into a common interface.
2. Design and construct a framework based on the common interface. The framework designed should be extensible and compatible for varied desktop tools. During the design, hot spots and frozen spots should be clearly defined.
3. Implement several application extensions by applying the framework. Fit in the hot spots and frozen spots using application extensions' specific technologies. In the scope of this project, two browser extensions (IE BHO and Chrome extension) and a non-browser extension (Microsoft Word window service) are proposed to be implemented.

1.3 Project contributions

In social search, client population significantly contributes to the search accuracy. Currently, only a Firefox plugin is implemented to feed SOPHIE core. Apparently, the client population is limited by the user of Firefox. This project aims to implement a desktop tools supporting framework. Web browser extensions should be easily developed by utilizing and reusing the framework. By simplifies the extension development process, more extensions can be developed to feed SOPHIE core, more clients can get involved. Thereby, increase the search accuracy.

The project also provides a framework for non-browser applications such as Outlook, Word and etc. The idea of integrating SOPHIE and non-browser applications will first time come true. It will be used as the data source of analyzing user actions on non-browser applications.

1.4 Dissertation outline

This project is organized as follows:

Chapter 2 displays the background and state of the art related to this project. Peer to peer technology, social search, implicit feedback and framework development are specifically explained. At the end of chapter 2, a overview of the current SOPHIE is also provided.

Chapter 3 clarifies the design of the framework. It starts from design analysis and challenges. Then the structure of the framework is discussed in detail. The hot spot and frozen spot of the framework are specifically defined.

Chapter 4 provides the implementation details. Firstly, it reviews technologies used in the development. Then it goes through each phase of the implementation in detail.

Chapter 5 evaluates the framework from aspects of both framework development and SOPHIE requirements. It also concludes the work done in this dissertation.

In chapter 6, the future work is proposed and discussed in detail.

Chapter 2

State of the art

This chapter introduces the current state of art of the technologies going to be deployed in this project.

2.1 Peer-to-peer technology

This section firstly introduces the definition of peer-to-peer network in our domain. Then the motivation of peer-to-peer over C-S mode is discussed. The classification and structure of peer-to-peer network is also included. Later in this section, sample peer-to-peer applications are introduced.

2.1.1 Introduction

The definition of Peer-to-peer network (P2P network) is understood in many ways. In today's discussions and publications, the understanding of Peer-to-peer network is often different or even opposite. Some like [24] and [25], peer-to-peer network is described as a collection of connected distributed resources. Others [1] describe it more extensible as any distributed network architecture composed of participants that make a portion of their resources (such as processing power, disk storage or network bandwidth) directly available to other network participants, without the need for central coordination instances (such as servers or stable hosts).

From our point of view of this project, peer-to-peer network is much more extensible. It can be described as a network structure that opposite to Client/Server network. As

said in [21]: “peer-to-peer can be defined most easily in terms of what it is not: the client-server model”. In next section we discuss the major distinctions between peer-to-peer network and client-server network.

2.1.2 Motivation of peer-to-peer

The major distinction between peer-to-peer and C-S model is that nodes in peer-to-peer network can perform both as client and server. In another word, the participants share some of their resources (processing power, storage capacity, network link capacity, printers, and etc). Those resource could provide services that offered by the network. Meanwhile, these services are accessible by other participants directly. This idea is different from C-S model which server nodes provide services accessed by client nodes.

The motivation of peer-to-peer generally speaking is that peer-to-peer avoid single point failure as there is no central control in peer-to-peer network. Each participant is able to store/access other participant’s data without affecting other participants. If a node fails, the other nodes are able to re-build the network.
More specific speaking, there are three major advantages:

Scalable: In peer-to-peer network, nodes are not only services consumers but also services providers. The increase of node population effectively improves the network capability while increasing work load. However, in client-server mode, services are constantly provided by centralized servers, more nodes will eventually mean that more resources need to be added at the host.

Stable: Resources and services in peer-to-peer networks are distributed. Therefore, single point failure is avoided. Comparing to C-S model, it is more stable.

Reduce resources cost: Resources including hardware, bandwidth and etc are shared among nodes in peer-to-peer network. Therefore, cost for resources is significantly saved significantly.

However, there are also three disadvantages that may concern. Firstly, decentralization of the system causes of administration difficulties. Secondly, lack of security. Finally, non services provided by the participants are a hundred percent reliable. Peer-to-peer attempts to address these issues from its architecture level. In the following section, we discuss the classification and structure of peer-to-peer network.

2.1.3 Peer-to-peer network structure

Regarding the topology structure of the network overlay, peer-to-peer network can be classified into two groups: structured and unstructured.

For structured peer-to-peer network, connections in the overlay are fixed. Such as Chord[1], CAN[27], Pastry[27] and Tarperstry[28]. They provide a self-organizing substrate for large-scale peer-to-peer applications. For such a system, distributed hash table (DHT) is often used for indexing nodes to ensure that any node can efficiently route a search to some peer that has the desired file, even if the file is extremely rare. Services can be accessed by nodes within a small number of DHT queries. A basic assumption of structured peer-to-peer is that nodes within the network are relatively reliable. In SOPHIE project, reliability of web user can not be guaranteed. Therefore, another group of peer-to-peer is introduced: unstructured peer-to-peer network.

In an unstructured peer-to-peer network, availability of participants is not guaranteed.

A dynamic look up service is often deployed to discover services provided or resources shared within the network. Specifically speaking, unstructured peer-to-peer networks can be grouped into networks with a centralized entity and those with a centralized entity, which are also known as hybrid peer-to-peer network and pure peer-to-peer network. A peer-to-peer network architecture is regard as pure peer-to-peer if any participants or nodes can be removed without causing any services loss of the network. On the other hand, if any participant performs as a central and essential node that could not be removed, those are regarded as hybrid peer-to-peer network. A significant difference between hybrid peer-to-peer networking and client-server networking is that client does not offer any of its resources.

2.2 Social Search and implicit collaboration

Social search or social search engine is a type of search algorithm that ranks search results by collaborating other users' search activities. Social search has become increasingly popular in recent years. Comparing to the traditional search engines such as Google, Yahoo, social search considers Web pages relevance and trust worth from the reader's perspective. Social search requires users' collaboration. "A key open challenge in designing social search systems is to improve the overall information seeking and consuming activities on the web. Reading time, scrolling, bookmarking, save-as, cut-paste are all considered relevant implicit sources of user preferences." [20] This section introduces how search results are ranked in social search, the motivation of social search and approaches of social search.

2.2.1 Introduction

As World Wide Web grows in size, traditional web search algorithms found it difficult

to return relevant results effectively just based on the content of web pages. Google's PageRank algorithm assigns relevance to web pages based on analysis of the link structure. The major weakness of those algorithms is that they rank web pages based on authors' perspective rather than the readers'. Social search is introduced. In social search, web pages are only regarded as relevance and trust worth from readers' perspective. For long time, web search and navigation have been regarded as a solitary activity of a single person using a Web browser. However, a online survey carried out by the Augmented Social Cognition group at the Palo Alto Research Center (PARC) indicate that many web search activities are with social interactions [22]. Web pages found relevance by some users may also relevant to other users with the same purpose. Social search engages web search users with similar purpose to collaborate together thereby increase the performance of search engines.

Web search results are ranked by relevance evidence. Web pages with stronger relevance evidence are considered more relevant that those with weaker or less relevance evidence. Relevance evidence can take many forms in different search algorithms. Traditional text search algorithms use the similarity between query and document and the quantity of the document as relevance evidence. It treats both query and document as a bag of words. Google also published partial of its ranking algorithm which finds relevance evidence from the link structure of web pages. It firstly considers how many pages are linked into this page and how authority they are; then it considers how many pages are linked out of this page and how authority they are. However, the drawback of these algorithms is that they focus on authors' perceptive rather than the readers'.

2.2.2 Motivation of social search

In the discussion of the motivation of social search, study carried out by PARC has to

be mentioned [22]. A survey participated by 150 users are conducted to study whether user search activities could be socialized and whether it could help users through their search process.

The study investigates the impact of social interaction from three aspects: before search, during search and after search. Thereby they construct a model of understanding social search. Figure 1 displays a canonical social model of user activities before, during, and after a search act, including citations from related work in information seeking and sensemaking behavior. [22]

- **Before search**

Context framing

Context framing is the process of defining information needs. The results show that 31.3% of the searches are motivated socially and 68.7% searches are self-motivated.

Requirement refinement

Since the information need is clear, user start to form their query. For example, user types in key words into the search engine. This phase is also known as generation loop [23]. In a generation loop, the initial query is modified based on the search results obtained. The modified query will be set as input into the search engine again. This loop continues until user is satisfied with the search results returned. In this case, the initial query will be influenced by social interactions. The results show that this cycle is marked by social interactions by 42.0% users.

- **During the search**

During this process, investigators study each of the three types of search acts transactional, navigational and informational to seek the social interactions.

Transactional search & Navigational search

With a transactional search, users target a source to perform a transaction. In another words, users navigate to a website to request specific information or to perform a specific task. During a navigational search on the other hand, users perform a series of actions to target on content from an already known location. The content will be easily recognized once discovered. It is often performed when the user knew but couldn't recall the target content. It is obvious that social interaction could hardly impacts these two types of searches as users could perfectly carried out the search individually.

Information search

Information search is the process of searching for information that may or may not be familiar to the user. During this process, query is modified based on feedbacks. Those feedbacks could be obtained from the search results or social interactions. The survey result shows that 59.3% of the users modified their initial query from social interactions. The authors also argue that 39.3% of the users had social experiences prior to search.

- **After search**

After search, users organize and distribute search results obtained. The result shows that 47.5% of the users distributed their search results for verification, feedback, or because they thought others would find it interesting.

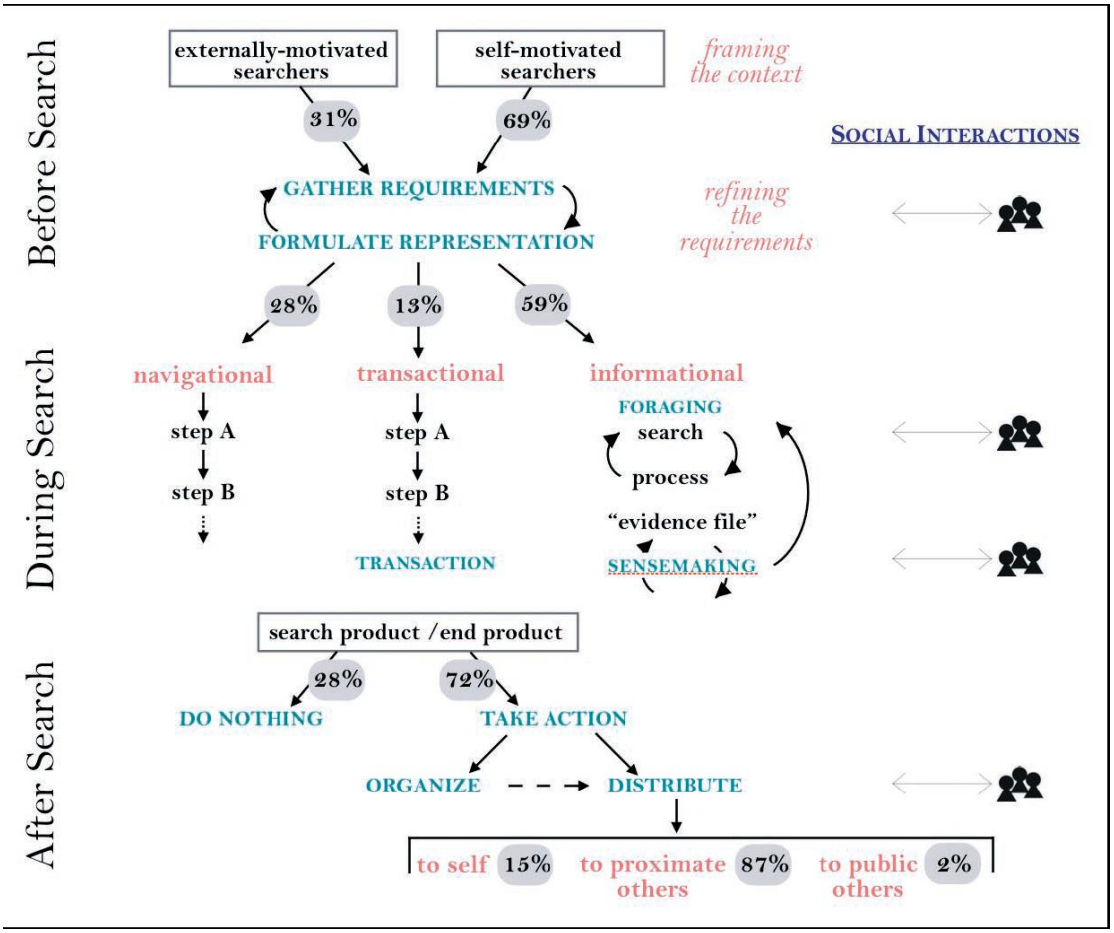

Figure 1 Canonical social model

This research conducted that users have a strong social inclination throughout the search process, interacting with others for reasons ranging from obligation to curiosity.

It is commonly agreed by researchers and practitioners that a search system that engages social interactions or utilizes information from social sources. Under this definition, social search systems can be divided into two general classes: social answering system and social feedback system. The next section introduces the classification of social search systems.

2.2.3 Classification of social search

As discussed earlier, social search systems can be grouped into two classes: social answering system and social feedback system.

Social answering system engages people to answer questions queried by others in a specific domain. Participants are able to query or answer a question or search for a particular question. Participants could come from various levels of social proximity, ranging from friends to greater public. Many social answering systems have been successfully developed and carried out to date, such as Yahoo! Answer, WikiHow and etc. Social answering systems intend to build a knowledge base from questions and answers. Effectiveness of such system depends on the search algorithm and recommendation algorithm to obtain the most relevant answers. Social answering system is not the type of system we are interested in so far in SOPHIE development.

Another type of social search system that we are actually interested in is social feedback system. Social feedback systems utilize social attention data to rank search results or information items. The system obtains social attention data and feedback from users either implicitly or explicitly.[3] Social search systems could collect user feedbacks explicitly in many ways. Popular mechanism such as vote, tagging are widely used. For example the most recent Google's search wiki. Users' votes will directly influence the rankings. In industry, explicit feedback is widely used by video websites (e.g. Youtube). Youtube engages users to vote a video from zero start to five starts. Videos with higher votes are considered to be popular and will be prompt. Meanwhile, votes imply users' interests. User with similar interests could be indentified as their votes are similar. Thereby more relevant recommendations could be made. However, there are two major drawbacks of explicit feedback algorithm. Firstly, votes or tags can not be subjective. People assign relevant or irrelevant

subjectively which might not be accurate enough. Another issue is that people could not be fully motivated to incorporate. Implicit feedback algorithm solves those two problems. Implicit feedback is introduced in the next section.

2.2.4 Implicit feedback

Implicit feedback is inferred from user behavior, such as book mark adding, printing, saving, downloading or even smaller user behaviors such as scrolling, mouse click and etc. This section first introduces the two general approaches to ranking with implicit feedback. Then a simple implicit user feedback model is described.

2.2.4.1 Implicit feedback approaches

The first approach considers implicit feedback as independent evidence. The idea is to re-ranking results obtained from a search engine according to previous user interactions. In this approach, the original ranking\s are ignored. In another word, re-ranking is carried out based on user interaction for the query in previous sessions only. The relevance score for each result is computed based on user actions and their importance. For a given query Q, the relevance score S is computed for each result R based on available user interaction features. Ii is the score of interaction i and Wi is the importance or weight of user interaction i. The score is computed:

$$S(R) = \alpha \sum I_i W_i \qquad (1)$$

The query results are ordered in by decreasing S(R) value to produce the final ranking. In this model, different user actions are assigned with different importance or weights. In this approach, any original search algorithms are ignored.

The second approach is improved from the previous one. It combines implicit feedback with some of those traditional search features. Similar to the previous approach, search results are re-ranked based on previous user interaction sessions. However, in this approach, the re-ranked score for each piece of results are contributed by both implicit feedback and the original features such as content based key words, PageRank and etc. The final is computed as follows:

$$S(R) = S(\text{Implicit feedback}) + S(\text{Original Score}) \quad (2)$$

The limitation of the first approach is that only those query results pair with existing implicit feedback data could be ranked. Comparing to the previous one, the combined approach provides more flexible performance. The reason is that more than 50% of web queries are unique without previous implicit feedback or user interactions [10]. Therefore, this approach is commonly used in social search. In the next section, I will present a simple implicit user feedback model which consist a basic set of user actions.

2.2.4.2 Implicit user feedback model

An implicit user feedback model is a model that encapsulates the process of user action data collection, storage and retrieval. Fig 2 displays a set of features used to represent user actions. Those actions are comprised of both directly observed features and query-specific derived features. It includes traditional browsing features such as cumulative dwell on for this domain, page dwell time for deviation from dwell time on page and query specific features such as match between the query words and the URL, query length and etc. Further more, other observed user actions such as bookmark, printing, save as, download, copy/paste, could also be included. By assigning each of those actions with a reasonable weight, the relevance of a particular query-result pair can be obtained.

Clickthrough features	
Position	Position of the URL in Current ranking
ClickFrequency	Number of clicks for this query, URL pair
ClickProbability	Probability of a click for this query and URL
ClickDeviation	Deviation from expected click probability
IsNextClicked	1 if clicked on next position, 0 otherwise
IsPreviousClicked	1 if clicked on previous position, 0 otherwise
IsClickAbove	1 if there is a click above, 0 otherwise
IsClickBelow	1 if there is click below, 0 otherwise
Browsing features	
TimeOnPage	Page dwell time
CumulativeTimeOnPage	Cumulative time for all subsequent pages after search
TimeOnDomain	Cumulative dwell time for this domain
TimeOnShortUrl	Cumulative time on URL prefix, no parameters
IsFollowedLink	1 if followed link to result, 0 otherwise
IsExactUrlMatch	0 if aggressive normalization used, 1 otherwise
IsRedirected	1 if initial URL same as final URL, 0 otherwise
IsPathFromSearch	1 if only followed links after query, 0 otherwise
ClicksFromSearch	Number of hops to reach page from query
AverageDwellTime	Average time on page for this query
DwellTimeDeviation	Deviation from average dwell time on page
CumulativeDeviation	Deviation from average cumulative dwell time
DomainDeviation	Deviation from average dwell time on domain
Query-text features	
TitleOverlap	Words shared between query and title
SummaryOverlap	Words shared between query and snippet
QueryURLOverlap	Words shared between query and URL
QueryDomainOverlap	Words shared between query and URL domain
QueryLength	Number of tokens in query
QueryNextOverlap	Fraction of words shared with next query

Figure 2 Features considered as implicit feedback

2.2.5 Conclusion

This section describes social search. It starts from the motivation of social search. As World Wide Web grows in size, users find out that it becomes more and more difficult to find interested resources by traditional search solutions. Social search has a obvious advantage over those traditional search that the social search focus on user's

perspective rather than the author's. Then the classification of social search is introduced. In the state of art, I specifically introduce social search with implicit feedback. Implicit feedback inferred from user behavior in a pervasive way. It solves the problem that explicit feedback could not fully motivate people to incorporate. SOPHIE deploys implicit feedback to collect and analysis user actions. Later in the state of the art, I will explain this specifically.

2.3 Framework development

A framework is a generic application abstract that allows the creation of different applications from a specific domain. Applications developed for that domain should be developed by extending the framework. It has been a common complain that the re-usability of object oriented programming is poorly deployed. Most of the software or applications are developed from scratch without reusing any existing component. Framework improves significantly in software reusability: it enables not only some components but also the entire system including their design to be reused. However, applications to be developed in that domain will be various in different aspects. Domain's variability and flexibility introduces a lot of complexities for a framework design. The complexity of framework development is the major challenge even under object oriented programming. This section introduces framework design and describes its procedure.

2.3.1 Introduction

The major purpose of framework design is for flexibility and generality. It focus on requirements and features for a whole domain rather than a particular problem. A variable aspect of an application domain is called a hot spot. [7] Different applications

from a same domain various from one another with regard to some (at least one) of the hot spots. In OOP, hot spot is performed as abstract classes or methods. Applications to extend the framework must customize their own hot spots. For example, JUnit framework provides abstract class TestCase. Each test applications built on JUnit framework must implement their own TestCase for testing purpose. JUnit provides specified API for any test application designer to achieve this. However, some features of a framework are shared by all applications. Those are mostly the core functionalities of the framework. Those features are the frozen spots of the framework. Unlike hot spots, frozen spots are implemented by the framework and can be accessed by extended applications. The kernel will be the constant and always present part of each instance of the framework. Hot spots and frozen spots introduce a major design issue for framework development. In a given domain, framework designer should neither be too specific nor too general. Frozen spots makes the framework too specific thereby limits its flexibility. However, hot spots increasing the level of abstraction of the framework thereby reduce its performance. Therefore, the trade off between flexibility and performance is present. An example of a far too generic framework is a "universal problem solver framework", illustrated in Figure 3.

```
While ( problem_not_solved( ) )
{
        solve_the_problem( );
}
return solution( );
```

Figure 3 Universal Problem Solver Framework

In this universal problem solver framework, problem_not_solved(), solve_the_problem() and solution() are hot spots. This framework could be applied to any problem solving domain. However, it's useless due to generalization. [7]

Framework building is still a hot topic both in industry and academic. The reason of this is not only the complexity of design and implementation but also lacking of standard solutions or methodologies can be followed. Later in this section, I will present a commonly agreed procedure for framework development.

2.3.2 Framework development process

Framework can be viewed as an abstract application. However, framework design process differs from traditional object-oriented application design in many aspects (Figure 4). Firstly, in traditional OOP design, user requirements only focus on the problems and tasks specific for a single application. However, framework design has to analysis the entire domain. Secondly, the output of traditional OOP design is a single application or system. The output of framework design is a framework that can be re-used by applications designed for that domain. Generally speaking, framework development consists of three major phases: domain analysis, framework design and installation.

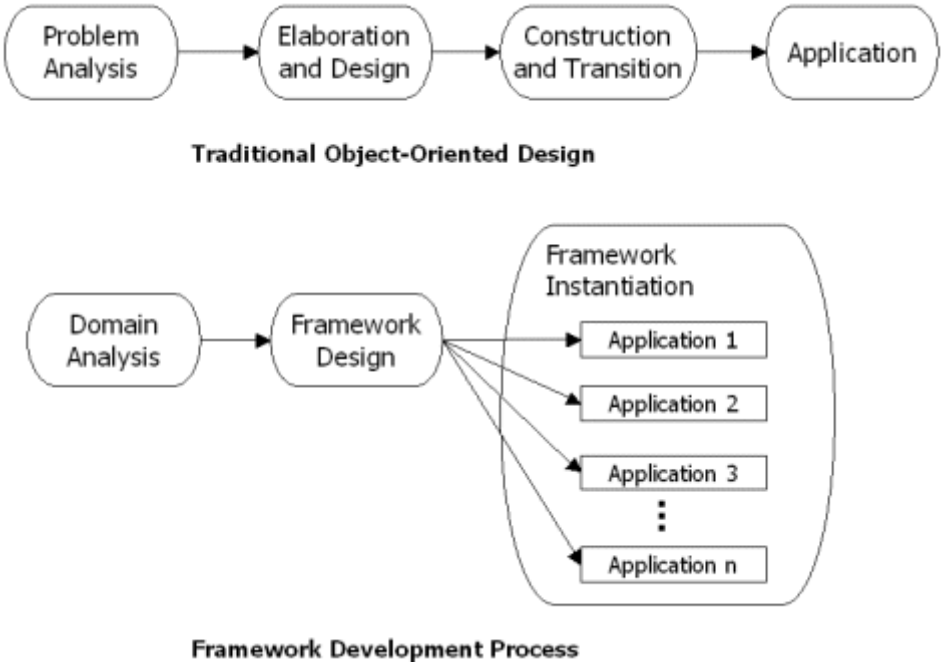

Figure 4 Framework development process

Domain analysis is the process of analyzing current requirements and possible future requirements. During this phase, the size and features of the given domain is defined. Depending on the size of the domain, complexities could be various. Larger domain requires more effort to gather and analysis information and resources. As the complexity increasing, the framework could be more expensive in time, money and other resources. On the other hand, smaller domain may be easier in gathering information and analyzing requirements. However, it reduces framework's applicability.

The next step is framework design. During this step, hot spots and frozen spots of the domain are defined. Abstract classes and concreted classes are defined. Frozen spots are also implemented. Framework API needs to be generated in this phase, therefore application designers could be able to implement hot spots by following the API.

The last step is framework installation. During this phase, hot spots are implemented, thereby, applications or systems are generated. All the systems and applications designed will have the framework's frozen point in common.

2.4 Existing system analysis

SOPHIE is a developing novel peer to peer based search system, employing a non-invasive Trust based approach to information analysis and gathering to the problem of delivering more useful web search ranking. The architecture of the system consists of: a peer client, a shared decentralized database and a trust engine DANTE. This project focuses on the functionality and scalability development of the peer clients.

The overall solution of SOPHIE is illustrated in Figure 5 and 6. Flow chat of the solution consists of five steps:

Step1: Users with SOPHIE clients installed browse web pages

Step2: SOPHIE analysis users' activity and index web pages locally

Step3: Meta data containing the information about web pages indexing is distributed to the peer network

Step4: When other users with SOPHIE clients installed perform key word search, peer network is consulted.

Step5: Peer network re-rank web pages returned to the user according to previous user sessions.

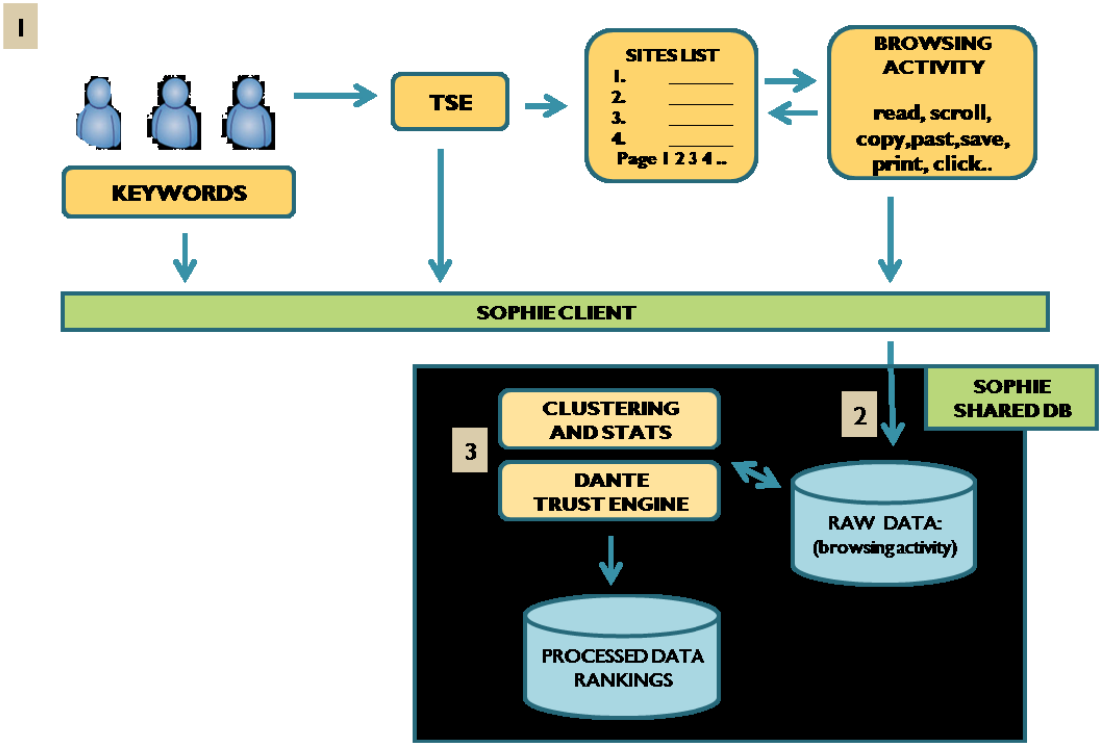

Figure 5 SOPHIE solution flow chat – a [29]

For the peer client side implementations, browser plugins or extensions are deployed to collect user browsing actions and activities. User activities are recorded when a new browser tab is opened. Activities including reading time, scrolling, save as, download and etc are recorded. After the tab is closed, user activities recorded will be saved in a XML file locally. Meanwhile, a copy of the web page is also saved. The two actions above are carried out by the peer engine installed locally. An interest rate or score will be assigned to each web page browsed based on the user activities recorded. Indexing will be carried out by the peer engine regarding to those scores.

This index file will be distributed to the peer network and consulted when other users within the network browsing Internet.

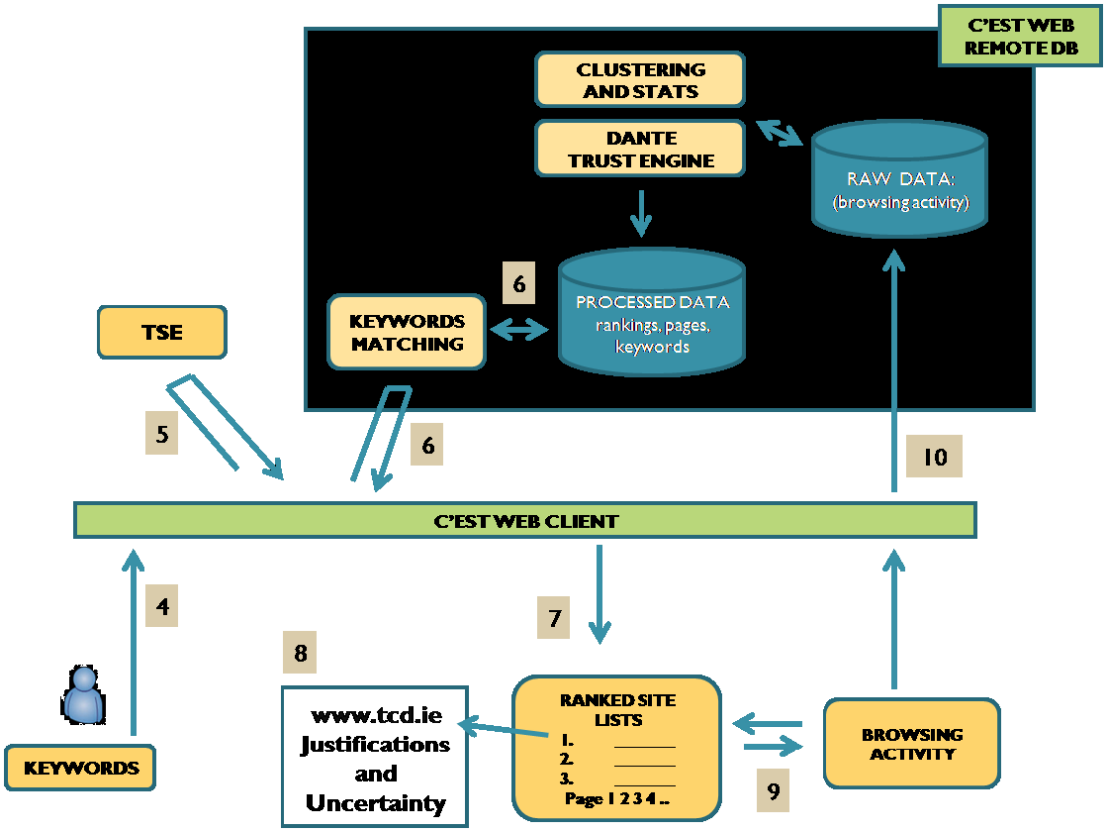

Figure 6 SOPHIE solution flow chat – b [29]

In this section, a brief review of the existing SOPHIE system is provided. SOPHIE project is comprised of a peer client, a shared decentralized database and a trust engine DANTE. Peer client collect user searching information and distributed the meta data over the peer network. When user performs a search from the peer client side, the peer network is consulted. This project focus on the peer client, aims to develop a desktop application support framework that enables existing and new desktop tools to cooperate with SOPHIE's peer network. The proposed work and their challenges are also provided.

2.5 Conclusion

This project aims enhance the performance of SOPHIE's peer client. By introducing a

desktop framework, it potentially increases the user population of SOPHIE. As a social search engine, larger user population improves the performance of SOPHIE. By supporting desktop applications such as word processors, e-mail tools, postscript viewers and etc. within the browsing framework, we can carry out research on enterprise search area and evaluate SOPHIE performance on enterprise search.

In this report, background researches that I have done so far are described. Peer to peer network can be defined as a set of connected nodes which are distributed among the network. Resources and services can be shared between nodes. Comparing to the C-S mode, client and server are not distinguished in P2P network. Each node can either be a service provider or a receiver. Then, social search and implicit feedback are introduced. Social search engines perform search by collaborating other users' search activities. Implicit feedback is the approach SOPHIE deployed to collect user activities. Framework design is also introduced later.

This report also provides a brief summary of the current SOPHIE solution from this project's view. Some issues of the current system are outlined and analyzed. Furthermore, solutions of those issues are displayed in the section of proposed work.

Chapter 3

Design

This chapter outlines the framework design. Three phases took place over the period of design. The first phase is to analysis what the framework should be able to provide. In a other word, it is to analysis user requirements. By doing that, it is important to understand that the end user of this framework is not normal users but SOPHIE core and SOPHIE desktop application end developers. The second phase is to build architecture for the framework. The architecture is supposed to be realistic to implement. Meanwhile, it should full fill the requirements concluded from the first phase. The framework should be extensible enough that other application extensions could access to SOPHIE easily by deploying this framework; on the other hand, the framework must maintain a relative high degree of un-modified framework code to guarantee the control of work flow. The third phase is to verify the design during the implementation. Multiple technologies are involved in the implementation of this framework, those technologies often different from each other in many aspects. Therefore, the design might need to be modified to meet some of those specific technologies.

The first two phases of the design will be explained in this chapter. The last phase will be displayed in implementation chapter.

3.1 Overview

This project aims to develop a desktop framework that is capable of supporting integration with a wide range of desktop tools. Development of a framework is

different from the development of a systems or software libraries; there are two important issues to be concern: extensibility and inversion of control.

- Extensibility: A framework can be extended by the user usually by selective overriding or specialized by user code providing specific functionality. In the scope of this project, extensibility specifies what developers can do by deploying this framework. They can reuse the code (frozen spot) or design pattern (hot spot) provided by the framework to develop their own system (desktop application extension in this case).
- Inversion of control: In a framework, unlike in libraries or normal user applications, the overall program's flow of control is not dictated by the caller, but by the framework. Inversion of control specifies what developers can’t do by deploying this framework. They can’t modify the overall flow of control. In this case, the work flow is strictly limited.

The specific components of the architecture are divided into three parts: application specific adaptor, a common interface and a server side adaptor. Figure 7 displays the general architecture of the design.

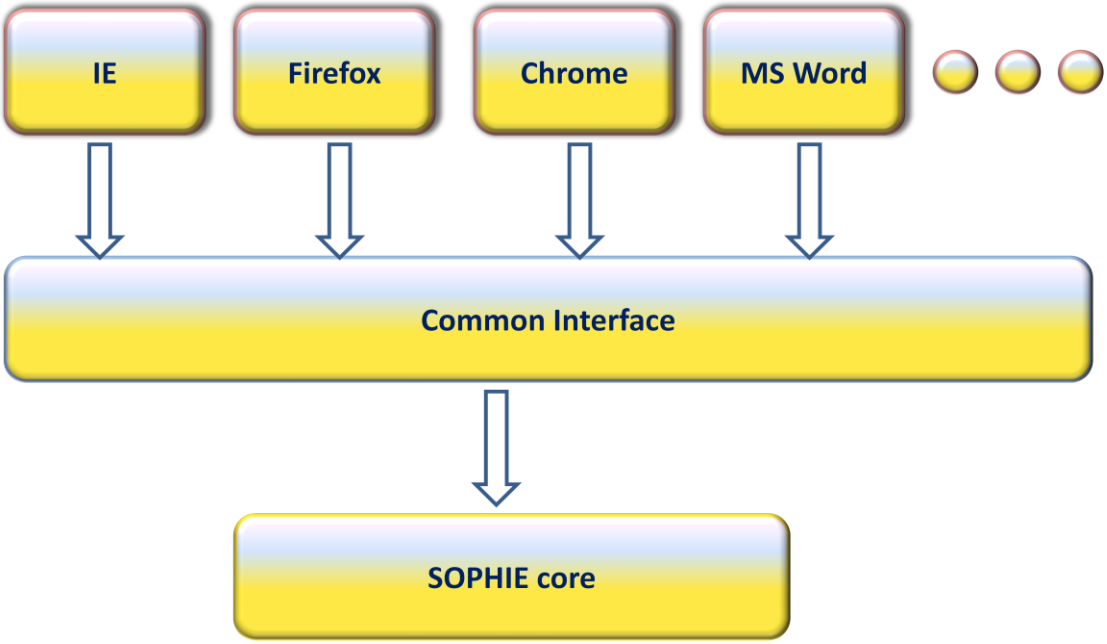

Figure 7 Framework general structure

Later in this chapter, this architecture will be explained in detail.

3.2 Design Analysis

There are several issues needs to be addressed with in this dissertation's scope.

Firstly, the current solution to date provides support for the interrogation of user activity only for web browsing, and specifically for later variants of the Firefox Web browser. This fact limits the user population of SOPHIE significantly.

Secondly, user activities data file are only generated and stored by the peer engine when a browser tab is successfully closed. Data will be lost when browser crash. Furthermore, real time data could not be used for indexing since.

Thirdly, current solution only works for web browsing now. Only user browsing activities are analyzed. Due to the requirements of enterprise search, other desktop applications (e.g. MS outlook, MS office tool kits and etc) should also be able to carry

out implicit feedback.

The work proposed is initially planed to address issues above. The goal is to build a desktop tools support framework that provides a framework for various browsers and desktop applications to cooperate with SOPHIE. After the goal is achieved, I propose to construct a solution to distribute the current peer engine to the peer network.

Generally speaking, there are three major challenges to solve those issues above: heterogeneity of desktop applications, integration with existing system and work with various technologies. The design challenges are discussed in the next section.

3.3 Design Challenges

The development of a desktop tools supporting framework is a difficult task. The challenges faced will be described in this section.

Firstly, desktop tools are quite different from each other not only from functionality but also technology. For example, a web browser's job is to forward user requests and resolve server response. However, a text edit application enables users to edit text. Apart from the differences of functionality and responsibility, they are also implemented differently technically. Science the basic requirement of this project is to access to those applications, the variation of technologies brings a significant challenge. For example, a Chrome browser extension deploys Chrome API and JavaScript. Chrome API and JavaScript are both script language. However, an Internet Explorer extension has to be built based on .NET framework, Object-Oriented Programming language for .NET framework such as C# and C++ can be used. This difference leads to a significant challenge for the framework: they could barely share any code base or common library.

The second challenge is to work with various technologies. Because of the reason above, it is necessary to work with various technologies such as Java, JavaScript, HTML, CSS, C# and other specific browser extension development technologies.

Usually an application extension is developed for two reasons:

1. Add specific capabilities to a larger application, it also enables customizing the functionality of an application. For example, web browser plugins are commonly used to play Flash videos, enhance browse experience and etc.
2. Provide a certain level of automation. For example, .NET based MS Word2003 extension enables Word2003 to edit Word2007 files.

However, the algorithm of this project is different. SOPHIE client side extensions attempt to inspect user activities without affecting user browse or work experience. This kind of task is seldom done. Therefore, a heavy learning curve is involved into this project.

Finally, it is important to integrate with the current system. On one hand, the framework designed should be able to cooperate with the existing Firefox plugin by little modifications on the plugin; on the other hand, the server side adaptor should be able to cooperation with the server without affecting the current server's functionalities.

Challenges are described in this section. Later in this chapter, design of the framework will be described in detail.

3.4 Framework Architecture

The following figure shows the architecture of the framework as an extension of

SOPHIE architecture. It is inherited from Figure 7.

As illustrated in Figure 8, the framework design is consist of three components: an application specific adaptor, a common interface that shared by all application adaptors and a server side adaptor which resides in the SOPHIE server. This section describes each of these three components in detail.

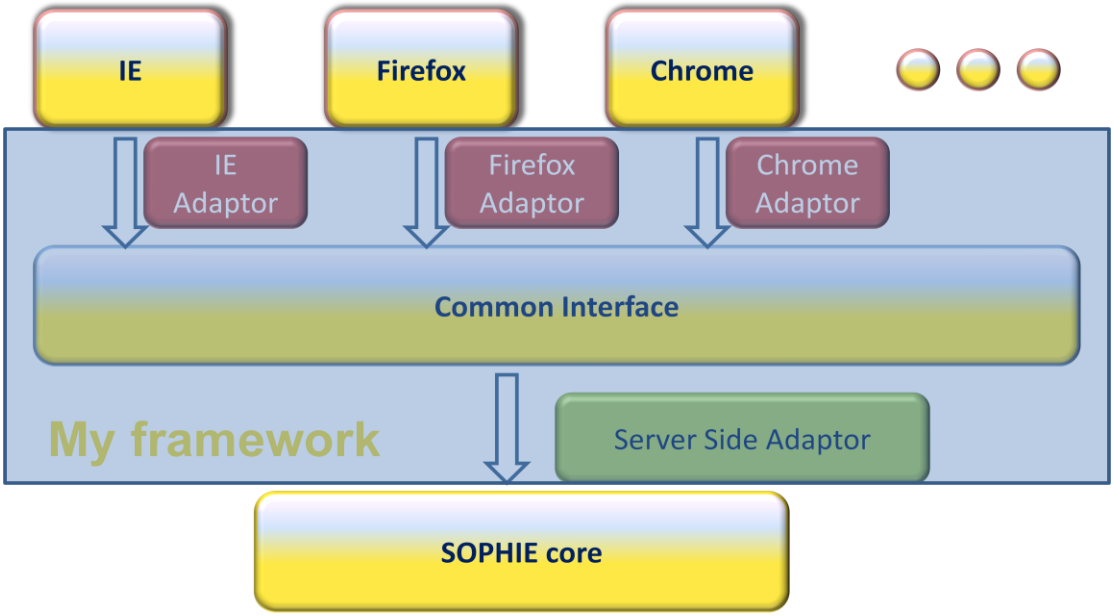

Figure 8 Framework specific architecture

3.4.1 Common interface

By analyzing the existing system and design requirements, it is concluded that SOPHIE desktop tool extensions are supposed to deliver very similar functionalities. Specifically speaking, each extension is required to capture user activates (a full list of activities is displayed on Figure 2), formularize the activity with some other necessary information (such as application type, time stamp) and finally forward it to the SOPHIE server.

Since all these functionalities are shared between various extensions, in the design, a

common interface is proposed to wrap these functionalities. Thereby, a new developing extension could reuse it instead of building from scratch. This is considered as the core of this framework. It is also known as the frozen spot of the framework.

In the development of a normal application, a common interface could be a set of shared abstract interfaces or software libraries. This feature also applied in this framework development. For example, all Microsoft Office tool extensions will be developed on .NET framework. In this case, a shared code base is easy to carry out. However, in the design of this framework, due to the heterogeneity of potential desktop tools, a common interface ends up to be more complex. Sometimes a shared code base is difficult to be achieved. For example, it is impossible for a Java Script based Chrome plugin and a C# based Internet Explorer plugin to share any code base. Chrome plugin deploys Chrome API and JavaScript, any implementations from .NET framework will not be supported. Meanwhile, IE plugin is based on .NET framework. An IE plugin is attached to an IE browser without loading any default HTML or JavaScript pages. Therefore, the two extensions listed above will find difficult to share a common code base. In this case, a shared design pattern will be provided by the common interface. In another word, they could deploy a shared design pattern using their own specific technology.

Later in this section, two issues will be discussed: functionalities that the common interface provides, shared design pattern and code base.

3.4.1.1 Functionalities provided

By analyzing the current Firefox extension and potential desktop tools' extensions, a list of basic functionalities to provide is concluded. These functionalities can be

divided into three categories: user activity capture, activity data formulization and server communication. They needs to cooperate and coordinate together and depend on each other.

- Capture user activities. In this framework, user activities are handled in event-driven model. In another word, the flow of the program is determined by events. Events interested would be: page loads, page update, page close, on focus, lose focus, mouse click, scroll and etc. A list of interested user activities is displayed on Figure2. When an event is captured, the corresponding event handler will be triggered. Activity data formulization and server communication processes reside in the handler will be executed then (Figure 9).

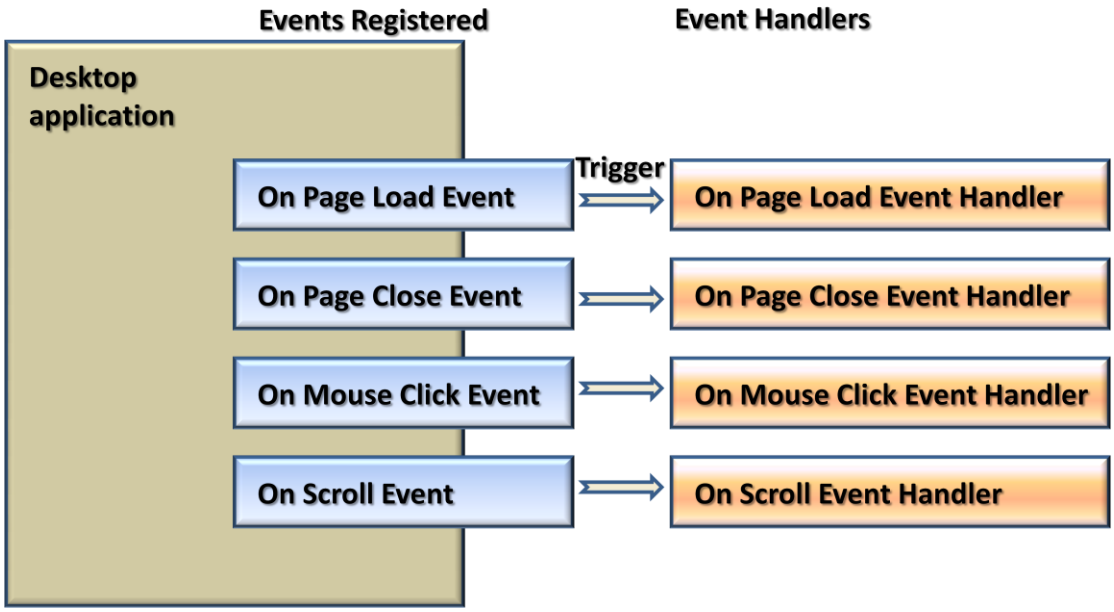

Figure 9 Event-driven user activity capture

- Data formulization. The output of user activity capture process will be treated as input of data formulization. Execution flow of data formulization will be resided within the even hander. Specifically, once an event is captured, a piece of message which contains the event's detail should be generated. All relevant information regarding to the event should be encapsulated within the message. The information contains:

- ✧ Type or name or the application that user is working with
- ✧ URL of the page (web browser specific)
- ✧ Window ID and tab ID
- ✧ Title of the current document
- ✧ Action type, or the type of the event
- ✧ Time stamp of the event
- ✧ Comments that describe more about the event. Such as scroll off set.

Once the message is full filled, it will be forward to the server communication process.

- ➢ Server communication. Communications between the extensions and server are simply achieved by socket stream. The message obtained from the second component above will be encapsulated into a socket package. Then the package will be sent to the server.

So far, the functionalities provided by the common interface are described. However, it only gives an overview of what the framework should do. In next section, it is explained that how the framework provides those functionalities.

3.4.1.2 Shared design pattern and code base

As discussed earlier in this chapter, the design and implementation of this framework is complex due to the heterogeneity of potential desktop tools. Sometimes a shared abstract interface or software library could not be achieved. This section analyzes where the common interface can take the form of shared code base and where can take the form of a shared design pattern.

Application extension development is developing language dependent or at least developing platform dependent. For example, IE plugin and Chrome plugin register events by deploying their own API. Thereby, a shared design pattern is used. However, a Microsoft Word extension and an IE plugin can be developed based on the same platform (.NET) with the same developing language (C#), in this case, a shared code base can be provided.

The shared design pattern is specified with the following components:

- ✧ A full list of interested user activities (event) to register
- ✧ A full set of functionality specification that developer has to follow
- ✧ Specific socket package format

The first component is specified by Figure 2. A full set of functionality specification is displayed from the previous section. Based on the information server required, a specific socket message format is described here:

Figure 10 Socket message format

Science a common design pattern is defined, the focus will be moved on to where a shared code base can be applied. In this case, the common interface can be divided into a .NET platform specific interface and a JavaScript specific interface. Extensions on .NET platform are able to share the .NET platform specific interface such as Internet explorer extension, MS Office tools extension. Same applied to JavaScript specific extensions such as Firefox extension, Chrome extension, Safari and etc (Shown on Figure 11). The level of code can be shared depends on the commonality of developing technologies. Further implementation details will be explained in next

chapter.

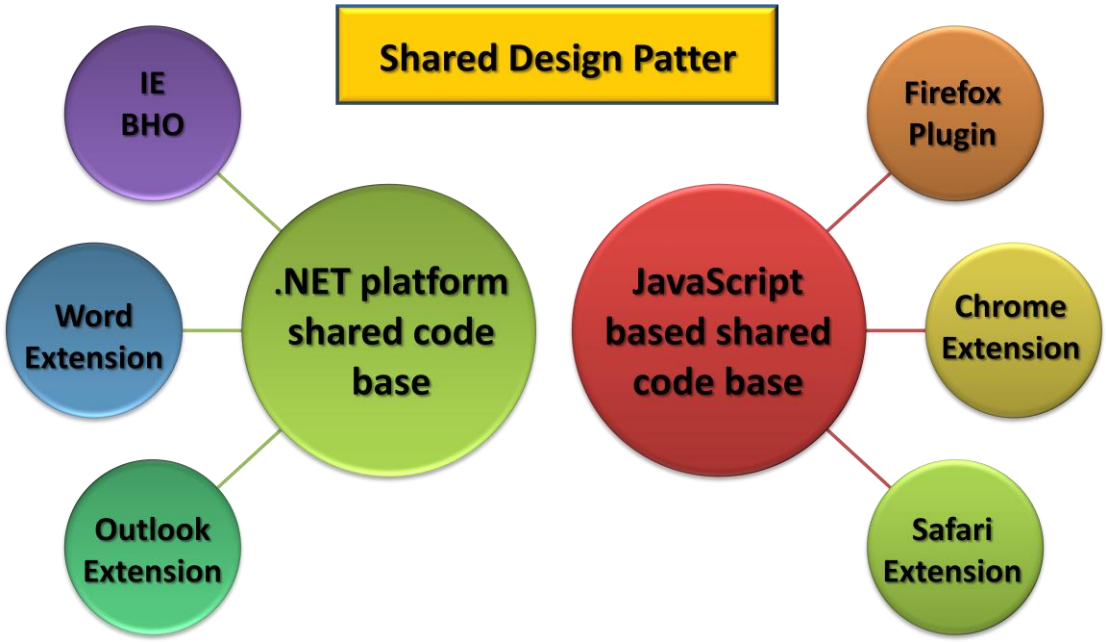

Figure 11 Shared design pattern and Shared code base

This section describes the design of common interface from two aspects: what functionalities the common interface should provide and what form it should take. It outlines why a common shared code base is impossible to achieve. Then it describes an alternative solution that integrates shared design pattern and shared code base. The next section, it will describe how to deploy the common interface by using an application specific adaptor.

3.4.2 Application specific adaptor

Application specific adaptor is deployed by an application to access to the common interface. For example, plugin for web browsers can be viewed as application specific adaptors. An application specific adaptor should be able to deploy the common interface by using its own technology. Once built, it will first attempt to deploy the shared code base where available. If the shared code base is not available, it will then

deploy the shared design pattern with its own specific technology.

An application specific adaptor can take many forms. It can be a plugin for web browsers, or an extension for non-browser desktop tools. It could even be a windows service that runs in the background of an operation system. Depends on the form it takes, an adaptor can reside either within the application or within the operation system.

As this adaptor is application oriented, it has to be technology specific. Each desktop application needs to implement its own adaptor in order to access to SOPHIE. For example, an Internet Explorer adaptor will take the form of Browser Helper Object (BHO, IE's name for plugin). A BHO resides within the Internet Explorer. An instance of BHO will be instantiated when a new IE tab is created. It will register events and formulize captured data by deploying the shared design pattern. Then it will communicate with server by deploying the shared socket code base. Details will not be explained here, three different adaptors and their implementation process will be described in the next section.

3.4.3 Server side adaptor

In previous sections, it mentioned that application adaptors will send messages to the server. A server side adaptor is required to handle client requests and parse them for SOPHIE core use.

A simple work flow of the server adaptor is shown on Figure 12. The adaptor should be able to handle the client socket, parse the information encapsulated and insert it into the database. Additionally, it should be above to generate XML files from the database. The XML files generated should follow the existing SOPHIE XML file

format. This feature can be viewed as part of SOPIE integration. Although it is out of the project's scope, it should be provided for future use.

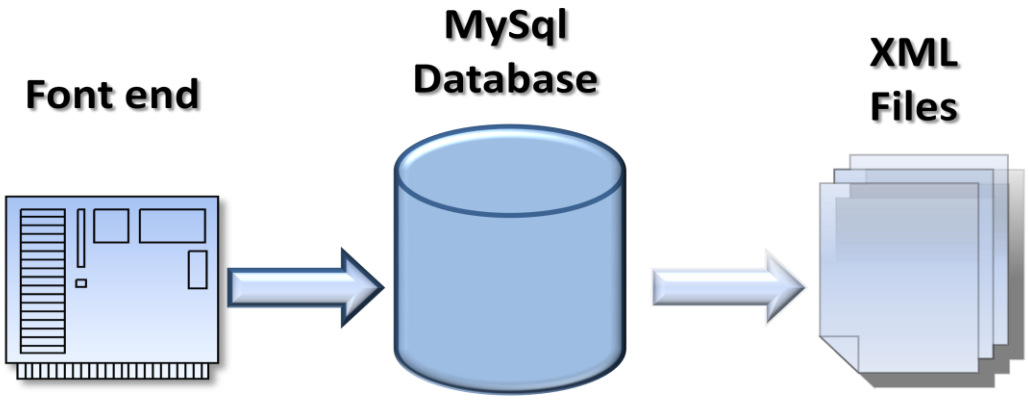

Figure 12 Server adaptor work flow

The server side adaptor can be viewed as a frozen spot of the framework. It is the default behavior of the framework.

3.5 Conclusion

This chapter describes the design of this project. Within the design hot spot and frozen spot of the framework are specifically explained.

A common interface is designed to encapsulate common functionalities among different desktop tool including user activity collection, data formulization and communication with the server. It provides both shared design pattern and code base to achieve this. An application side adaptor is designed to enable any desktop application to access to the common interface. At last, the server side adaptor will be responsible for handling client incoming message, manipulate database record and SOPHIE XML files generation.

In the design, the common interface is viewed as the key component of the framework. Different from the developments of a normal application and other single platform framework, the common interface in this case will take the form of both a shared code base and a shared design pattern. The shared design pattern is applied where the shared code base cannot be achieved.

The next chapter introduces how implementation is carried out from the design illustrated here.

Chapter 4

Implementation

This chapter outlines the implementation of the framework. The implementation is comprised of four steps. It starts from establishing a .NET platform common interface. An IE8 Browser Helper Object (BHO) is developed applying this common interface. Then the server side adaptor is developed to handle the BHO's messages. After that, a JavaScript based common interface is established and a Chrome extension is built to verify the interface. At last a Microsoft Word adaptor is build applying the .NET common interface.

The major goal of the implementation is to build a framework by developing different application adaptors and to verify the framework built by those adaptors.

4.1 .Adaptor Development Overview

As described previously, application adaptors may take different forms according to the types of application. For example, web browsers deploy plugin as their adaptors; Microsoft Office Tools deploy add-on or windows services as their form of adaptors. This section specifically describes the form of adaptors used in the implementation.

For Internet Explorer, Browser Helper Object (BHO) is deployed. For Chrome browser, Chrome extension is deployed and windows service is deployed for Microsoft Word. The implementation of these adaptors will be different. Some of them will be similar enough to share a common code base. Other has to share a

common design pattern.

4.1.1 Browser Helper Object

Browser Helper Object (BHO) [30] is a DLL module designed for IE to provide customized browser functionality. In progress Component Object Model (COM) module object can be created with BHO. Thereby, IE can load it each time it starts up. A BHO object runs in the same memory as the web browser. It can perform many actions on the available window. For example, it can perform go forward, go backward, refresh and other actions provided by the browser window. It can also access to the browser toolbar and make customized changes. Within the scope of this project, it is possible to detect browser's typical events such as DocumentComplete, Navigate, Click and etc.

Each BHO resides within the IE. It is loaded only when the IE starts up. A new instance of BHO will be created when a new browser window or browser tab is created. It dies when the corresponding window is disposed. BHO is available for IE from version 4.

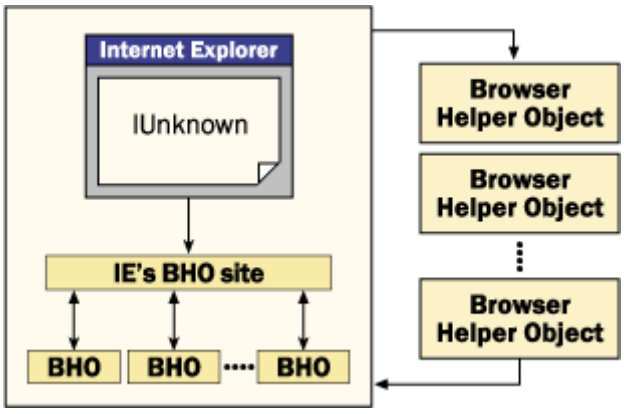

Figure 13 BHO implementation lifecycle

In its simplest form, a BHO is a COM in-process server registered under a certain registry's key. As shown on figure 13, once start up, IE looks up into the registry and load the BHO object. Then IE uses the methods provided to pass its IUnknown [31] pointer down to the helper object. The detail of COM in-process loading and IUnknown interface performance will not be displayed in detail here. For further information, please refer to Microsoft MSDN library.

The development of BHO is based on .NET framework. Science it is loaded up as a COM in-process, non html files or JavaScript can be packaged. In another word, a shared code based or frozen spot cannot be built between a BHO and other JavaScript based browser plugin.

4.1.2 Chrome extension

A Chrome extension is a compressed package of files including HTML, CSS, JavaScript, images files and etc. It is a developing technique from Google Chrome team. Extensions are essentially web pages; APIs from XMLHttpRequest, JSON, HTML5 and etc are currently supported.

The file structure is fixed for a Chrome extension (Figure 14). It must have one manifest file that specifies the general information of the extension. At least one html file is required. Different html files are of different responsibilities. This will be explained later. Some other files can also be included but not necessary such as JavaScript files, images file and etc.

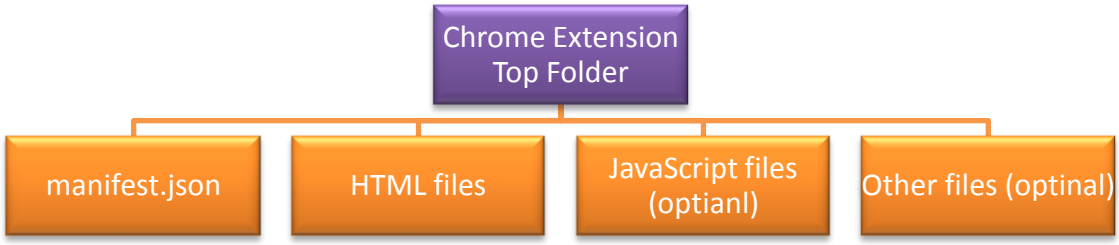

Figure 14 Chrome Extension File format

The manifest gives information about the extension, such as the most important files and the capabilities that the extension might use. A sample manifest file is provided below [31].

```
{
  "name": "My Extension",
  "version": "2.1",
  "description": "Gets information from Google.",
  "icons": { "128": "icon_128.png" },
  "background_page": "bg.html",
  "permissions": ["http://*.google.com/", "https://*.google.com/"],
  "browser_action": {
    "default_title": "",
    "default_icon": "icon_19.png",
    "default_popup": "popup.html"
  }
}
```

Figure 15 Sample manifest file

The file tells general information about the file such as extension name, version,

description and etc. Then it refers to other functional files within the extension. Any HTML, JavaScript or other files must be referenced in the manifest file to deliver their functionality.

4.1.3 Microsoft Office Word adaptor

The most commonly used Microsoft Office Word (referred as Word below) extensions are Word add-ins. Word add-ins perform similar to web browser plugin. Upon starts up, Word initiates an add-in object. An application-level add-in object could extend Word by customizing the user interface (UI) and by automating Word. However, from the aspect of framework, a significant drawback of add-in is lack of extensibility. Other desktop applications need to develop their own add-ins or extensions from scratch. It ends up having a huge number of extensions (regarding to the number of applications). Therefore, Microsoft windows service is deployed.

On Microsoft windows operation system, windows service is a long-running executable that performs specific functions and which is designed not to require user intervention. [33] Once installed, a windows service will run automatically when the operation system boots up. It will run in the background silently without the user's notice. All the features listed above meets the SOPHIE application adaptor's requirements. Even more, a single windows service could perform adaptor work for many .NET based Microsoft Windows applications.

4.2 IE BHO implementation

IE BHO is developed as IE's SOPHIE application side adaptor. It can be developed using both C# and C++ based on .NET framework. C# is used in this project. The

design decision is made based on the following reasons:

1. C# provides more libraries for BHO development. Therefore it gives richer control of the browser.
2. Code in C# could be reused in windows service development.
3. C# provides an easy programming style than C++ in BHO development.

According to the design pattern, a successful BHO should be able to achieve the following goals:

1) Capture user activities on the browser window. (Event registration)
2) Formulize activity data, measure the time user spent on each page, measure user action frequency and etc.
3) Forward data formulized to the server.

However, before those goals can be implemented, a BHO has to be built and ties to IE browser. Later in this section, the implementation of this BHO will be specified in detail. Meanwhile, the shared code base established and shared design pattern deployed will be clearly outlined.

4.2.1 Implementation of an empty BHO

Two objectives have to be achieved by an empty BHO. The first one is to create a BHO class that ties up with Internet Explorer. The second one is to build a register program to register the BHO developed. This section introduces the implementation of how these two objectives are achieved specifically.

4.2.1.1 BHO implementation

To build an IE BHO, an IObjectWithSite [35] has to be build at the very first to notify

that this is a BHO project. Two functions are added into the interface.

```
[PreserveSig]
int SetSite([MarshalAs(UnmanagedType.IUnknown)]object site);
[PreserveSig]
int GetSite(ref Guid guid, out IntPtr ppvSite);
```

Figure 16 BHO implementation 1

Those two functions should be implemented for set and retrieve the pointer to an IE tab instance.

Then the main BHO class is built to implement the IObjectWithSite interface. In this class, two libraries from MSDN are referenced: MSHTML and SHDocVw. MSHTML provides all interfaces for accessing the Dynamic HTML (DHTML) Object Model. SHDocVw stands for Microsoft Shell Doc Object and Control Library. As the main DLL for Internet Explorer, it is used to grab and control instances of IE.

After a BHO template is built, a registration process needs to be added.

4.2.1.1 BHO registration

Dislike normal applications, BHO won't start running under the user's command. It has to be registered with the operation system. Therefore, two functions are needed for register and unregister of this DLL.

4.2.2 Event registration

After a template of BHO is completed, the first process to develop is event registration. Event registration is the most fundamental process in event-driven programming. Each user activity to capture must be registered as event. Specific to BHO event registration, both MSHTML and SHDocVw provide several user event

listeners. Those listeners are encapsulated within the corresponding object. In another word, in order to deploy those listeners, the corresponding class must be correctly initiated.

Specific to this project, SHDocVw.WebBrowser and HTMLDocument are used. With those two classes, the following events can be registered:

- Document Complete (triggered when the page is loaded completely)
- FileDownload (when a file is downloaded either by user or automatically)
- OnQuit (when a tag or window is closed)
- BeforePrint and AfterPrint
- Some other events but not in the concern of this project

From the list above it is noticed that those events are far less from SOPHIE required. This is because that BHO is most commonly used for browser control rather than inspection. In another word, BHO is usually developed to provide user more customized functionalities, such as toolbar, video play and etc. This is the case that applied to all other browser extensions.

In order to solve this problem, System.Windows.Forms library from Microsoft MSDN is deployed. System.Windows.Forms library is often used for general Windows window control. Even with this library, not all events from Figure 2 are captured. Due to the time limitation and scope of the project, those events are left for future work.

Once a user action event is captured, corresponding processes within the event handler have to be triggered. Those processes can be encapsulated and reused. The detail is introduced in next section.

4.2.3 Data Formulization

For most of the event captured, the BHO just simply encapsulate it in the following format:

For example, scroll, click, page update and etc. However, the time user spent is different. Time spent is not an individual event that could be captured. It is the gap between page load and tab/window close or another page load. Meanwhile, the unfocused time has to be excluded.

Therefore, in the case of time spent, data formulization becomes more complex. The solution provided is a Timer class that holds the time already spends for each page. The time should be able to pause when focus is lost and continue when the focus is regained. This timer class will be used as a frozen point of the framework since time spent is a common problem to other desktop applications.

Once a message is formulized, it will be sent to the server side. The next section describes how this process is done.

4.2.4 Data sending

As socket is a platform independent client-server communication tool, it is used in the process of data sending. It is also independent of the BHO. Thereby, it is developed as a frozen point of the framework that can be reused by other C# based adaptors.

A SocketSend class is developed for data sending. It is highly encapsulated since the

only function it provides is to forward messages to the server.

4.3 Server side adaptor implementation

In the previous section, a BHO is developed for IE. It could capture user actions and send it to the server through socket connection. This chapter introduced how a server side adaptor is developed in order to handle the messages coming in.

Server side adaptor is developed based on J2EE platform using Java 1.6. It is a platform independent comparing to other technologies. Therefore, the SOPHIE server is not affected.

The entire adaptor can be divided into three components: client messages handle, database manipulation and XML file generation. XML file generation is related to SOPHIE integration and will be introduced in future work. This section describes how the first two components are completed.

The server provides a socket listener that keeps listening into the network. When a new client socket stream comes in, a separate thread will be created to handle the client message. Therefore, multiple requests can be handled simultaneity.
However, clients' side socket cannot be implemented based on JavaScript based browser extensions. The details will be explained on the next section. For JavaScript based browser extensions, HTML5 web socket is deployed. Therefore, a HTML5 web socket handler is added on the server side adaptor. It will be used similar to a general socket handler. It handles web socket streams and distributed them into separated threads.

Once a client message is received, either from a socket or web socket, the message

will be parsed and stored into database. Despite of the source of the socket scream, the client message will be in the same format. This facilitates the process of parsing on the server side.

An interface is developed on the server side to monitor all incoming messages. All the messages will be modified and stored into the database.

4.4 Chrome extension development

Chrome extension is based on JavaScript and Chrome API. It has significant difference from .NET based extensions. Both Chrome API and JavaScript are scripting language. They have to be supported by the web browser. In another word, they depend on the web browsers. Meanwhile, because of the technical difference between .NET framework and JavaScript, IE BHO and Chrome extension can barely share any code base. This extension is developed to provide a code base for JavaScript based browser extensions such as Firefox, SeaMonkey and etc. The implementation strictly deploys the shared design pattern of this framework.

Similar to the BHO, a Chrome extension should be able to achieve the following goals:

1) Capture user activities on the browser window.
2) Formulize activity data, measure the time user spent on each page, measure user action frequency and etc.
3) Forward data formulized to the server.

This section will introduce the implementation of Chrome BHO in the following steps. Firstly, the files structure of the extension will be introduced, including a specific description of each file. The implementation of a Chrome extension template without

any functionality will be introduced. Secondly, event registration of Chrome extension will be focused. Data formulization will also be introduced. Thirdly, the communication mechanism between client and server will be discussed.

4.4.1 Implementation files structure

Chrome extension API specifically describes the structure of extensions files (shown on Figure 14). This section introduces the file structure specific on this extension, shown on figure.

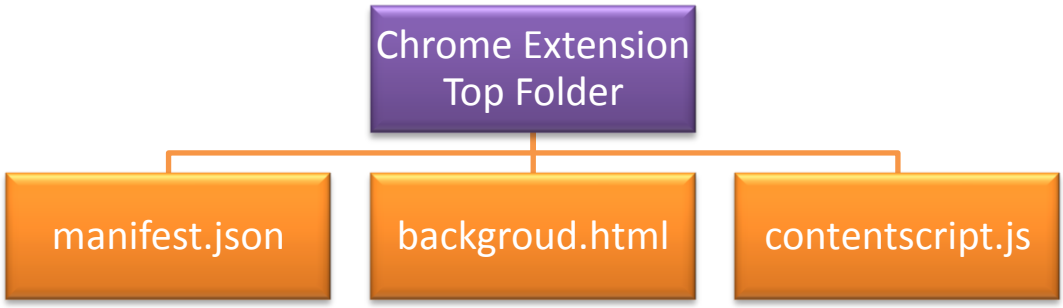

Figure 17 Chrome extension file structure

The structure is very simple. Manifest.json is used to provide general information about the extension including name, version, description and etc. Meanwhile, the file also clarify that background.html will be deployed as background page, contentscript.js will be deployed as external JavaScript file.

Background.html is used as background page. It is a long-running script to manage some task or state. It exists for the lifetime of the extension, and only one instance of it at a time is active. [36]

Contentscript.js is the content script. Content scripts are JavaScript files that run in the context of web pages. By using the standard Document Object Model (DOM), they can read details of the web pages the browser visits, or make changes to them. [37]

Once the structure is realized, the extension can be loaded into Chrome browser. As shown on figure 18 and 19, a Chrome extension can be loaded by the following steps:

1. Bring up the extensions management page by clicking the Tools menu and choosing Extensions.
2. Click the + to add developer information to the page. The + changes to a -, and more buttons and information appear.
3. Click the Load unpacked extension button. A file dialog appears.
4. In the file dialog, navigate to your extension's folder and click OK.

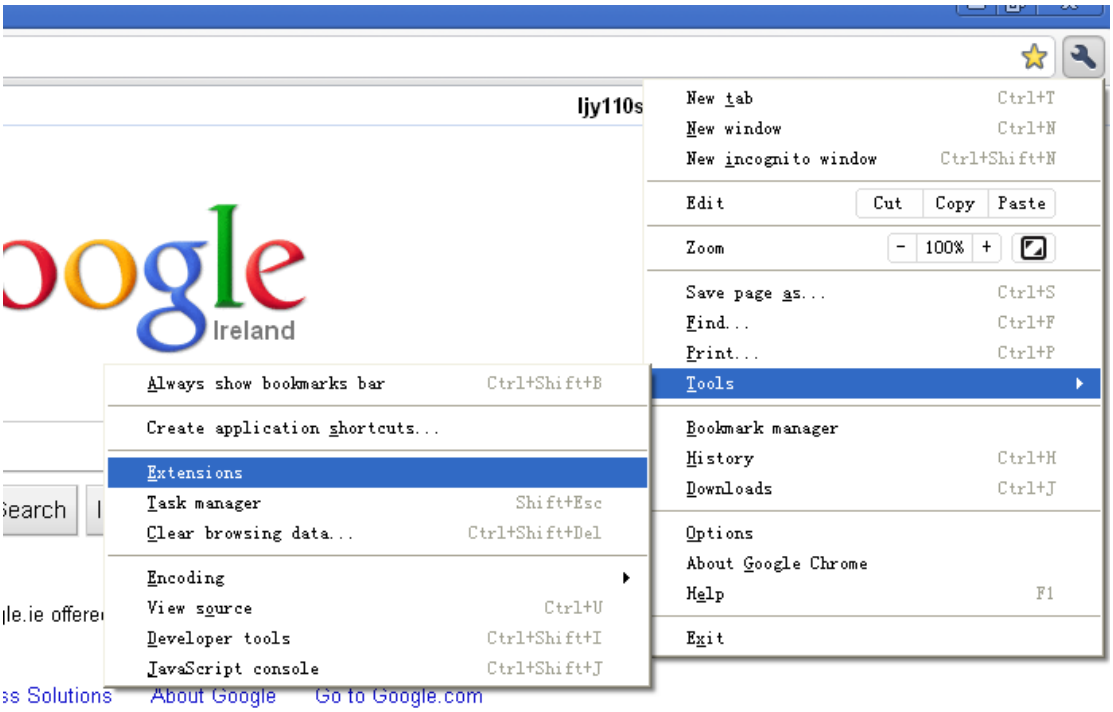

Figure 18 Chrome extension loading 1

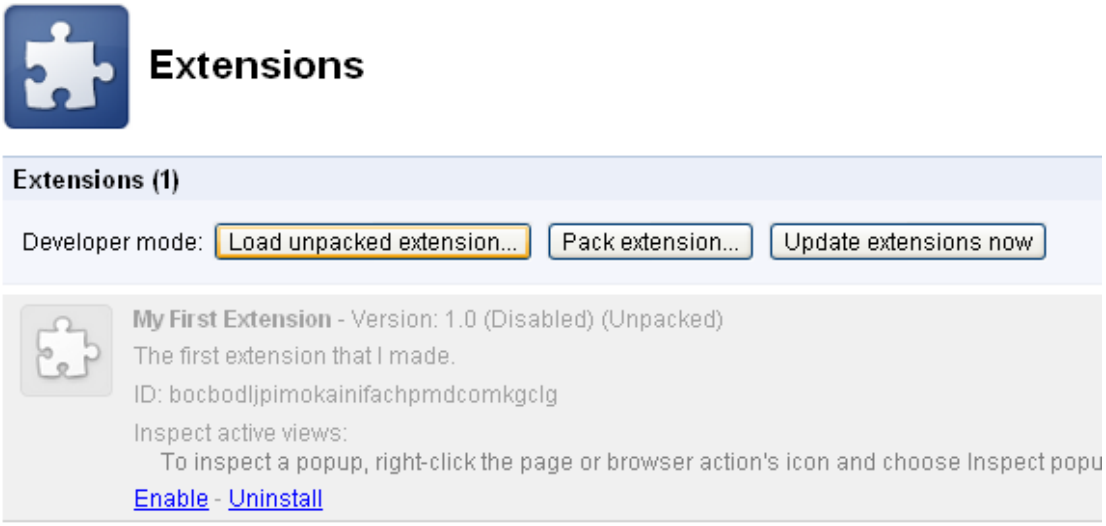

Figure 19 Chrome extension loading 2

So far, an extension doing nothing has been loaded into the Chrome browser. The next section introduces how SOPHIE client side functionalities are added into the extension.

4.4.2 User action collection

Same as the IE BHO, user action collection is implemented in an event-driven model. Both Chrome API and JavaScript user events listeners and handlers are deployed. The actual listeners and handlers used will not be specifically displayed, for future information please refers to Google Chrome extension home page [37].

When a user event is captured, it will be encapsulated into a message with other general browser information such as application type, title, URL and etc. The message format is exactly the same as IE BHO's client message.

Then, the generated message is supposed to be sent to the server as the design pattern specified. However, an issue is come across that JavaScript doesn't support socket

communication. The next section specifically explains how this issue is solved.

4.4.3 Data sending

As a scripting language, JavaScript do not support socket itself. This has been accepted commonly. To enable socket connection in JavaScript, there are generally three solutions:

1. By applying a 3rd party library. A few libraries are provided for JavaScript connection. Jsocket [38] is tried in this project. It works fine for normal JavaScript bonded html files. However, it cannot be recognized by Chrome extension manager. In another word, it cannot be loaded with other extension files. Therefore, the first attempt fails.

2. The second solution is implemented by invoking a .swf Flash file from the content script. The .swf file is responsible for socket connection. This is an informal solution. It requires the client side browser to be Flash enabled. Because of the additional restrictions it brings in, this solution is abandoned.

3. The third solution applied is Java applet. It is able to create a class with socket functionalities encapsulated using Java applet. Then the content script could be able to realize socket connection by referring to the Java applet code. However, similar to the first solution, a Java applet cannot be loaded in the background.html. This solution failed due to Chrome restrictions. The Chrome extension team has set Java applet support as priority -1 [40].

Due to time constrains, the author has to conclude that it is not possible to cooperation socket connection with socket connections. As an alternative solution, HTML5 web

socket is deployed. Web socket provide bi-directional TCP communications for web browsers and web servers. [41] It can be viewed as a web version of socket. It is a full-duplex text based socket that enables web pages to communicate with a remote host. Meanwhile, it traverses firewalls, proxies, and routers seamlessly

The actual deployment of web socket is similar to socket. The initial (), send () and other necessary functions are encapsulated within the content script. A web socket listener has also been added on the server side adaptor. It handles each web socket request on a separate thread to guarantee scalability.

4.5 Word extension development attempt

By implementing the IE BHO and Chrome extension, the entire picture of the framework is presented. Hot spot and frozen spot are clarified. In this section, the focus is moved on to create a Word extension by applying the framework. As it is not fully completed, this section only introduces: the development of a windows service, installation process development and track on running Word instance.

4.5.1 Windows service development

As mentioned previously, windows service could perform adaptor work for many .NET based Microsoft Windows applications.

A windows service class is inherits from System.ServiceProcess.ServiceBase. A Onstart() method is created to encapsulate executing code when the service is started. Similarly, OnStop() and OnContinue() is created to encapsulate executing code when the service is stopped and resumed. In this case, only OnStart() method is focused.

However, OnStart() must return the control to the operation system when it ends executing. In another word, no infinite loop or block can be included. Therefore, all customized functions have to be encapsulated into threads. This provides an opportunity to initiate a hot spot for the framework. Every desktop application that deploys windows service needs to implement its own thread. The thread can be started within the OnStart() block.

For MS Word, it applies the hot spot and initializes a new thread which monitors the user actions. This will be discussed on section 4.5.3.

4.5.2 Installation process

An installer is required for the windows service. It is a fixed code for windows service. Modification is not required for adapting the code. Therefore, it will not be explained in detail. Please refer to the code and MSDN installer class for further information. [42]

An important issue has to be concerned is system security. User name and password will be promoted when any user action triggers the listeners encapsulated within the windows service. This can be solved by modify the Account property to Local System in the windows service.

4.5.3 Tracking Word application instance

So far, an empty windows service is completed. A hot spot is embedded within the windows service. In order to capture user activities on Word, a thread has to be created to track the existence of on-running Word application.

C# has a specific library Microsoft.Office.Interop.Word.Document to control and monitor Word applications. Within the thread developed, multiple listeners and handlers are developed.

Due to the time constrains, this windows service is not fully completed. It will be left to the future work.

4.6 Conclusion

This chapter describes the implementation practice of the framework. Two browser plugin and one non-browser application extension are developed by applying the framework.

A single design pattern is shared by these three extensions. Meanwhile, a code base is shared by IE BHO and Word windows service. During the development of Chrome BHO, a shared code base for JavaScript based extensions is also implemented.

Based on the implementation process, it is concluded that a desktop tools support framework can be achieved. For each of these extensions, not all of the user events on Figure 2 are captured. However, it provides a solution and template of how the user events can be captured and manipulated. Therefore, the competence work can be done in future.

Chapter 5

Evaluation and Conclusion

The design and implementation chapters give a clear view on the project. This chapter aims to evaluate the project. Since the end user of this framework is SOPHIE rather than normal human users. The evaluation will be different. User questionnaire and performance evaluation are not adopted. Therefore, a critical evaluation will be carried out based on the requirements of SOPHIE and framework development.

From the aspect of SOPHIE requirements, there are two issues: user action collection and SOPHIE integration. From the aspect of framework development, three key features will be discussed: extensibility, degree of non-modified code and flow control.

5.1 SOPHIE requirements

In this section, the performance of user action collection and SOPHIE integration will be discussed. The specific requirements is consulted from SOPHIE designer and developers.

5.1.1 User action collection

The competence of captured events is not required by this project. This section aims to clarify the strength and weakness of each technologies deployed. The table below listed all user activities collected for IE BHO, Chrome Extension and the existing Firefox plugin.

	IE BHO	Chrome Extension	Firefox plugin
Page load	Captured	Captured	Captured
Page update	Captured	Captured	Not Captured
Tab/Window close	Captured	Captured	Captured
Tab/Window focused	Captured	Captured	Captured
Tab/Window lose focus	Captured	Captured	Captured
Print	Captured	Not Captured	Not Captured
File download	Captured	Not Captured	Not Captured
Active time spent	Captured	Captured	Captured
Scroll	Captured	Not Captured	Captured
Active text selection	Not Captured	Captured	Not Captured
Copy/Paste	Not Captured	Captured	Not Captured
Bookmark	Not Captured	Captured	Captured
Save as (html file)	Not Captured	Not Captured	Captured
Save as(image file)	Not Captured	Not Captured	Captured
URL/Title	Captured	Captured	Captured
Action frequency	Not Captured	Not Captured	Not Captured

Table 1 Activity capture list

From the table it is concluded that each of these three extensions has their own strength and weakness. .NET based IE BHO is good at system control. It is able to detect operation system events such as print and file download. However, it lacks the control of DOM and web browser itself. As mentioned previously, BHO lacks web browser event listeners even it has a rich set of window event listeners. Time consumption is necessary to solve this problem as MSDN provides a huge set of libraries without specific explanation. This project has provided several examples on

how other libraries are deployed. Chrome extension performs perfect at web page related actions such as copy/paste, active text selection and etc. This is caused by the rich set of event listeners provided by both Chrome API and JavaScript. However, Chrome extension is lack of operation system layer commands control. In another word, neither Chrome API nor Javascript could access to the operation system due to security reasons. Chrome API group is still working on the problem.

5.1.2 SOPHIE integration

In terms of SOPHIE integration, three goals are achieved.

1. A framework on user action collection is built. Two web browser extension and one desktop application extension is developed based on the framework. The user activities collected meet the existing solution.
2. The client side SOPHIE engine is successfully separated from the client. Interface is also provided for the existing solution.
3. User activity data is generated according to the SOPHIE core and it is also stored in the database.

However, the SOPHIE XML file generation is not achieved. It can be done from the existing framework. Due to the time limitation, this will be left to the future work.

Generally speaking, the framework coincides most of the SOPHIE requirements.

5.2 Framework assessment

The framework assessment is made based on three key features: extensibility, degree

of non-modified code and flow control. Those features are commonly utilized to justify a framework. [7]

5.2.1 Extensibility

Extensibility requires the framework should be easily extended by new developing applications. Various desktop application extensions should be easily developed by utilizing the framework. Dislike extensibility for other applicationsThe framework developed provides two features to support extensibility:

- A common interface is provided. Various desktop applications can deploy the shared design pattern for user action collection. A shared code base is also provided by the common interface for data formulization and server communication.
- Two separated functional components are provided for both .NET based extensions and JavaScript based extensions. Extensions could be developed by deploying their desired components.

However, the drawback of this framework in terms of extensibility is on the client side. Each application extension to be developed needs to implement its own event-driven process. The event-driven process has to follow the design pattern provided.

5.2.2 Degree of non-modified code

Non-modified code is also called framework code. It is the shared code base provided in this framework Users could extend the framework but they cannot change the code.

Within this framework, the shared code base contains:

- .NET based timer interface
- .NET based socket connection interface
- JavaScript based socket connection interface
- J2EE based server side adaptor

The detail of the shared code base has been clearly introduced in the previous chapter. Due to the multiplicity and dissimilarity of desktop applications, high degree of non-modified code will reduce the extensibility of the framework. For those framework components where a shared code base cannot be constructed, a shared design pattern is deployed instead.

5.2.3 Flow control

Dislike normal systems, the overall flow control of the framework is dictated by the framework rather than the extension developer. The flow control is displayed on the figure below.

Figure 20 Framework flow control

The flow control is comprised of four dependent phases. The input of a particular phase is the output of its previous phase. It is strictly dictated by the framework from

the design. Extension developer could not ignore any phase and jump into the next phase.

5.3 Conclusion

This chapter evaluates the framework from both the aspect of framework development and SOPHIE requirements. Another important evaluation is supposed to be carried out based on the users – extension developers. However, due to the limitation of time and human resources, this work will be left to the future work.

Chapter 6

Future work and Discussion

This project created a framework to support the development of SOPHIE desktop tool extensions. The framework provides a list of shared code base (frozen spot) and shared design pattern (hot spot). Three extensions are developed based on the framework: an IE BHO, a Chrome extension and an incomplete MS Word windows service. Due to the time restrictions and resource limitation, the framework is not realized fully.

This chapter describes future work to enhance the existing framework. It will be displayed from four different aspects: existing framework evaluation, SOPHIE integration, non-browser application support and alternative solution discuss.

6.1 Existing framework evaluation

As mentioned at the end of Chapter 5, a further evaluation of the framework needs to be carried out by SOPHIE extension developers. It requires developers with the knowledge of application side adaptor development. Then as soon as they understand the framework, an application side adaptor can be developed. For example, a developer with Apple's Safari plugin knowledge, he/she could construct an adaptor to access to the common interface.

Specifically speaking, plugins for different browsers should be developed. Window service should be extended to support more desktop applications. After the development, two conclusions should be made based on the developing process

- Does the framework provide full set of hot spot and frozen spot required?
- Is it possible to extend the framework just by understanding the shared design pattern and code base?

This work involves many different technologies. Therefore, more than one developer with different technique background may be required.

6.2 SOPHIE integration

Apart from the SOPHIE integration already achieved, there are two more features required.

- Generation of SOPHIE XML file. The existing SOPHIE index solution requires user data to be stored in XML file with a fixed format. This framework provides a database for the user data collected. Further work needs to be done to generate the required XML files from the data stored.
- Refact the existing Firefox plugin and fit it into the existing framework. The current Firefox plugin output data into a local file. Refactor needs to be done by modify the output to socket connection. This can be done by applying the HTML5 Web Socket interface provided.

6.3 Non-browser application support

1. Complete the current MS Word windows service

   The current windows service based MS Word extension deploys both shared design pattern provided and shared code base for .NET platform. Windows service can also be viewed as a hot spot for other Windows operation system applications. Due to the time restrictions, very little amount of activities on Word

can be captured. A full functional MS Word windows service should be completed based on the given framework.

2. Extend the windows service to other Windows desktop applications

   Continued from the previous point, once a full functional MS Word windows service is done, it is a good practice to extend the windows service. As explained previously, windows service is developed as a base for multiple desktop applications on Windows operation system. In another word, a windows service is designed to capture events from multiple applications. Therefore, it should be extended to support MS Office Tools such as Excel, Power Point and etc.

3. UNIX Daemon development

   Windows service is deployed for Windows operation systems. A UNIX daemon should be developed as an equivalent to windows service on UNIX. A UNIX daemon process runs in the background and rather than under the direct control of a user; they are usually initiated as background process. Any UNIX application supported by the daemon process could deploy the existing framework.

6.4 Alternative solution discuss

There is an alternative solution for web browser to capture user actions. The solution is based on JavaScript. Instead of deploying plugin at the user side, this solution relies on the proxy server side implementation. A brief introduction to the solution will be provided here: Figure18 displays how Http requests are send from web browser through proxy servers.

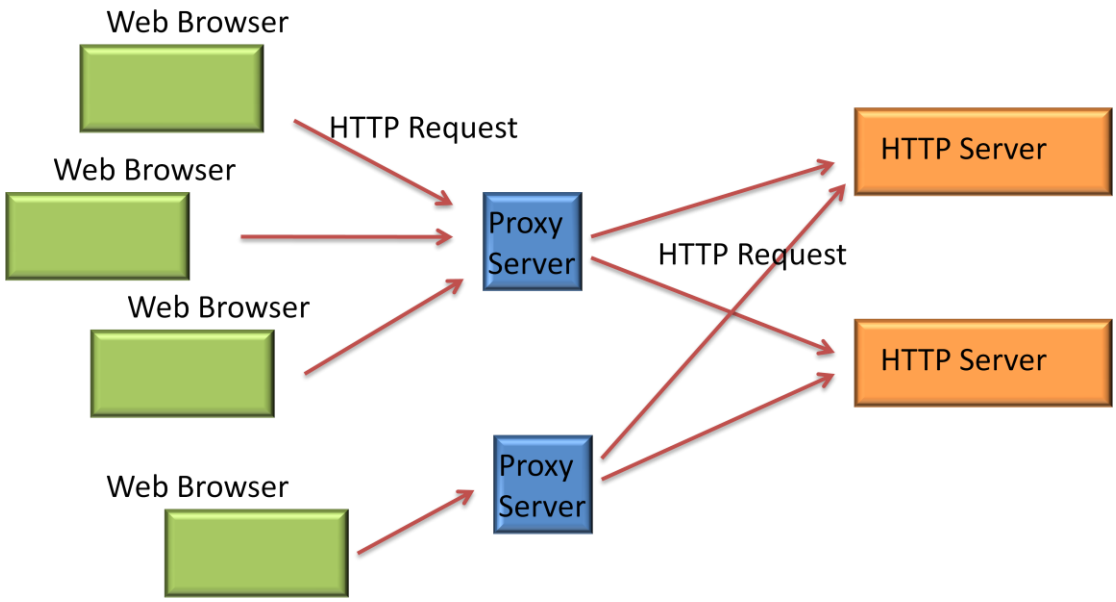

Figure 21 Http request through proxy server

As shown on figure 19, the JavaScript based solution embedded a piece of JavaScript code with any HTML page replied by the HTTP Server. This piece of JavaScript code will be used to capture user activities.

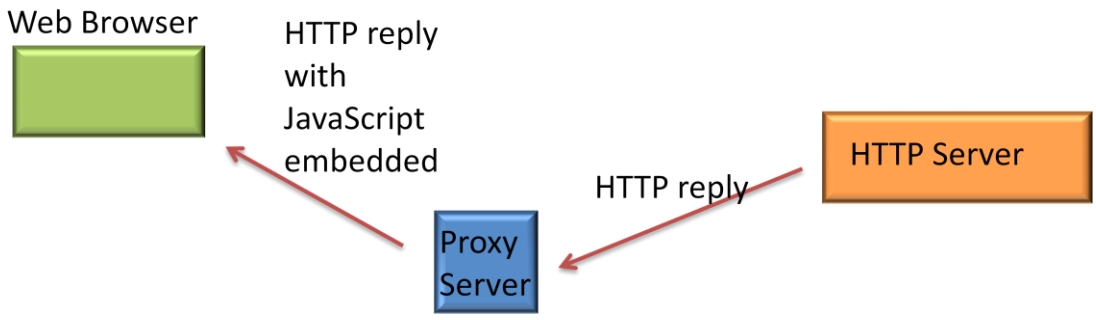

Figure 22 SOPHIE JavaScript solution

Comparing to the framework, JavaScript based solution has several advantages. First of all it's a cross browser solution. All browsers that support JavaScript will do. It saves efforts of plugin development. Secondly, it is a plugin free solution. No plugin is required to install on the client side. It saves the effort of motivating clients to install SOPHIE plugin. However, it won't support any non-browser desktop

applications. Meanwhile, different web browsers support JavaScript in different ways. It’s different to create a single code base that fit all browsers.

By analyzing the JavaScript solution, an integrated solution should be carried out for the future. The new solution will be based on a modified framework. The framework will be in charge of events out of JavaScript’s scope. Meanwhile, the JavaScript solution will also be deployed to enhance the framework solution.

Reference

[1] Rüdiger Schollmeier. A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications, Proceedings of the First International Conference on Peer-to-Peer Computing, pp: 101, 2001.

[2] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F., and Balakrishnan, H. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols For Computer Communications, pp: 149-160, 2001.

[3] Chi, Ed H. Information Seeking Can Be Social, IEEE Computer Society Press, pp. 42-46, 2009.

[4] Longo L, Barrett, S. & Dondio P. Information Foraging Theory as a Form of Collective Intelligence for Social Search, in 1st International Conference on Computational Collective Intelligence - Semantic Web, Social Networks & Multiagent Systems, pp: 63-74, 2009.

[5] Stephens D.W., Krebs J.R., Foraging Theory. Princeton, NJ, 1986.

[6] Pirolli P., Fu W. SNIF-ACT: A Model of Information Foraging on the World Wide Web. User Modeling 9th International Conference, pp: 146, 2003.

[7] Marcus Eduardo Markiewicz, Carlos J. P. de Lucena, Object oriented framework development, Crossroads, Volume 7, Issue 4, pp: 3-9, 2001.

[8] Peter Sommerlad,Guido Zgraggen,Thomas Corbat, Lukas Felber. Retaining comments when refactoring code, companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, pp: 653-622, 2008.

[9] Kelly D., J. Belkin N., Reading Time,Scrolling and Interaction: exploring Implicit Sources of User Preferences for Relevance Feedback During Interactive Information Re-trieval, Proceedings of the 24th annual international ACM SIGIR conference on

Research and development in information retrieval, pp: 408 – 409, 2001.

[10] Agichtein E., Brill E., Dumais S. Improving Web Search Ranking by Incorporating User

Behavior Information, Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval, pp: 19 - 26, 2006.

[11] Atterer R., Wnuk M., Schmidt A., Knowing the User's Every Move - User Activity Track-ing for Website Usability Evaluation and Implicit Interaction, Proceedings of the 15th international conference on World Wide Web, pp: 203-212l, 2006

[12]Fayad, M. E., Schmidt, D. C. Building Application Frameworks. Addison-Wesley Pub Co, 1st edition, 1999.

[13] W. Pree, G. Pomberger, A. Schappert, P. Sommerlad. Active Guidance of Framework Development, Proceeding of the Journal SoftwareConcepts and Tools, Journal Software: Concepts and Tools 16 (3), 94-103. 1995

[14] Han Albrecht Schmid, Systematic framework design by generalization, Communications of the ACM, Volume 40, Issue 10, pp: 48 – 51, Oct 1997

[15] Martin Fowler. Refactoring: improving the design of existing code, ISBN 0201485672, 1999

[16] Kaiping Zeng Huss, S.A. RAMS: a VHDL-AMS code refactoring tool supporting high level analog synthesis, Proceedings of IEEE Computer Society Annual Symposium, pp: 266-267, 2005

[17] www.softomate.com, SoftMate, Retrieved 17 Feb 2010.

[18] Akerman, Richard. How to Enhance Access with Browser Extensions, Internet Librarian International 2005

[19] Plug-in Development Overview, https://developer.mozilla.org , Developer Center, Retrieved 17 Feb 2010

[20] Longo L, Barrett, S. & Dondio P. Towards Social Search: From Explicit to Implicit Collaboration to Predict Users' Interests, in ACM 5th International

Conference on Web Information Systems and Technologies, 2009

[21] M. P. Singh. .Peering at Peer-to-Peer Computing, IEEE Internet Computing, vol. 5, no. 1, pp. 4-5, 2001

[22] B. Evans and E.H. Chi, Towards a Model of Understanding Social Search, Proc. Computer Supported Cooperative Work (CSCW 08), ACM Press, pp. 485-494, 2008.

[23] D. M. Russell, M. J. Stefik, P. Pirolli, and S. K. Card. The cost structure of sense making, In Proc.CHI'93, ACM Press, pp: 269-276, 1993.

[24] Meta Computing. .Peer-to-Peer Architecture Proposal Legion . An Integrated Architecture for Secure Resource Shari January 2001

[25] S. Wray; T. Glauert. A. Hopper. The Medusa applications environment: Multimedia Computing and Systems, Proceedings of the International Conference, pp: 265-273, 1994.

[26] M. Kelaskar, V. Matossian, P. Mehra, D. Paul, M. Parashar. A Study of Discovery Mechanisms for Peer-to-Peer Applications, Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, pp: 444, 2002

[27] Antony Rowstron, Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, In Proceeding of IFIP/ACM Middleware 2001, Heidelberg, pp: 329-350, 2001.

[28] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and routing, Technical Report: CSD-01-1141, 2001.

[29] Barrett, S, Longo L. Sophie White Paper. Trinity College Dublin.

[30] http://msdn.microsoft.com/en-us/library/bb250436(VS.85).aspx, Browser Helper Objects: The Browser the Way You Want It, Microsoft MSDN, 2010

[31] http://msdn.microsoft.com/en-us/library/ms680509(VS.85).aspx, IUnkonwnInterface , Microsoft MSDN, 2010

[32]http://code.google.com/chrome/extensions/overview.html, Chrome Extension Overview, Google, 2010

[33] http://msdn.microsoft.com/en-us/library/d56de412(VS.80).aspx, Introduction to windows service, Microsoft MSDN, 2010

[34] http://msdn.microsoft.com/en-us/library/aa189710(office.10).aspx, Word add-ins, Microsoft MSDN, 2010

[35] http://msdn2.microsoft.com/en-us/library/Aa768220.aspx, IObjectWithSite, Microsoft MSDN, 2010

[36] http://code.google.com/chrome/extensions/background_pages.html, Chrome extension background page, Google, 2010

[37] http://code.google.com/chrome/extensions/content_scripts.html, Chrome extension content script, Google, 2010

[38] http://code.google.com/chrome/extensions, Chrome extension home page, Google, 2010

[39] http://code.google.com/p/jsocket/, Jsocket home page, Jscoket, 2010

[40] http://code.google.com/p/chromium/issues/detail?id=30258, Chrome extension java applet support failure, Google, 2010

[41] http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-01, Internet-Draft at IETF, 2010

[42] http://msdn.microsoft.com/en-us/library/system.configuration.install.installer.aspx, Installer API, MSDN, 2010

APPENDIX I - Abbreviation

P2p – peer to peer

MS – Microsoft

MSDN – Microsoft Developer Network

BHO – Browser Helper Object

DHT – Distributed Harsh Table

C-S model – Client-Server model

PARC – Palo Alto Research Center

OOP – Object Oriented Programming

COM – Component Object Model

DLL – Data Link Layer

DHTML – Dynamic HTML