

國立交通大學

資訊科學與工程所

碩士論文



知識管理中資料自動擷取與追蹤

Automatic Crawling and Tracking for Knowledge Management System

研究生：李杰叢

指導教授：袁賢銘 教授

中華民國九十五年六月

知識管理中資料自動擷取與追蹤

研究生: 李杰鼓

指導教授: 袁賢銘

國立交通大學資訊科學研究所

摘要

面對龐大的數位資料，使用者可以透過知識管理系統的協助，自動整理這些資料，將原本雜亂無章的資料轉換成為有用的資訊，並且給予使用者建議：哪些是建議使用者閱讀的？或者哪些資料需要使用者協助過濾。

網際網路本身就是一個相當龐大的數位資料庫，使用者想要讓知識管理系統可以幫忙從中萃取知識，知識管理系統的管理員可以透過代理程式的幫助，將龐大的資料自動的從網際網路裡抓取及其他的前置處理，提供後端的知識管理系統。然而知識管理系統的管理員必須根據不同的通訊協定使用不同的代理程式，代理程式的操作隨著不同的程式有不同的設定方法，不夠彈性的設定往往讓程式多抓了很多不必要的資料，造成網路頻寬與計算資源的浪費；同時，無從得知每次擷取的一大堆資料中，究竟哪些是已經擁有的，那些才是新的資料或更新過的資料，知識管理系統處理很多重複且已經處理過的資料，將造成計算資源的浪費。

為了解決上述的問題，我們提出一個代理程式的架構，並且將之實做出來；代理程式的開發人員只要照著這個架構針對不同的通訊協定實做程式元件，便能容易的整合進這個程式裡；代理程式不但提供擷取資料的能力，並且還能追蹤資料，透過程式的報告便可以追蹤資料的改變狀況。我們目前整合了檔案傳輸協定（FTP）與超文件傳輸協定（HTTP）兩種通訊協定的擷取與追蹤，並且提出一個演算法支援採用網址改寫（URL rewriting）實做超文件傳輸協定議程（session）的網站的資料追蹤。以外，知識管理系統的管理者只要熟習這個代理程式的多重步驟的使用介面，便可以精確的描述想要擷取的資料，從被支援的網路協定的資料來源下載或者追蹤資料。

Automatic Crawling and Tracking for Knowledge Management System

Student: Jie-Tsung Li

Advisor: Shyan-Ming Yuan

Department of Computer and Information Science

National Chiao Tung University

Abstract

The user can use the Knowledge Management System (KMS) for learning from data. The system analysis the mess data and converts into the useful information. The system not only recommend what is the user have to read but ask the user what is filtered.

The Internet is a large digital library. The KMS manager can use the agent programs to download data automatically and do pre-processes for the KMS. It will provide the useful information. However, the manager has to use different programs for different network protocols with difference usage. The usage is not flexible so that they might download much something that is not needed and waste bandwidth and computing time. Besides, the manager knows nothing about the data difference from the last download. It would cost a lot of time and resources if the KMS has to process something the same and processed again and again.

To solve the problems, we proposed the architecture and implement a prototype. The developers just program the new components for new protocols following it to be easy to be integrated in. The program provides not only crawling but tracking, and report the situation of the data from last time crawled. We have integrated the abilities of FTP and HTTP protocols and we propose an algorithm to support the website designed by HTTP Session with URL Rewriting. Besides, the manager of KMS is only familiar the Multi-Steps configurations. The program will crawl and track data more precisely.

Acknowledgement

能夠完成這篇論文，首先要感謝袁老師兩年來的指導與教誨，讓我了解到做研究方法，並且在做研究的過程中獲益良多、更上一層樓。感謝葉秉哲學長兩年來各種訓練，以及不辭勞苦的為我解答各種問題，給我非常多的協助，讓我可以完成這篇論文。此外還要感謝博士班學長鄭明俊、邱繼弘、蕭存喻、吳瑞祥、高子漢，謝謝你們給我的種種建議，增加此篇論文的思考廣度。

謝謝紀暘跟勇宇，兩年來一同成長一同學習；謝謝實驗室其他的同學與學弟妹，感謝你們的照顧，解答了許多課業與研究上得問題，為實驗室帶來熱絡的研究風氣與實驗精神。

還有我的朋友，佳燕、庭瑄、淨茵、松翰、婉蓉，感謝你們無時無刻的加油，給我支持，並且幫助我生活上遇到的各種問題。感謝同在系計中工作的建明、鈺婷、冠廷、冠華以及其他的夥伴，兩年來相互的幫忙彼此。謝謝慧萍老師，每次都在百忙之中讓你麻煩，並且不斷給予我很大的鼓勵。

最後，將這篇論文獻給我的父母，感謝你們平日的付出與關心，感謝你們辛苦的撫養與教導，並給予我這個機會，可以完成碩士學業；對未來的各種責任與挑戰，我會更加的努力去承受與面對。



Table of Contents

| | |
|---|-------------|
| Acknowledgement | iv |
| Table of Contents..... | v |
| List of Figures | viii |
| List of Tables | xii |
| Chapter 1 Introduction..... | 1 |
| 1.1. Preface | 1 |
| 1.2. Motivation | 1 |
| 1.3. Research Objectives | 3 |
| 1.4. Organization | 5 |
| Chapter 2 Background..... | 6 |
| 2.1. Knowledge Management System (KMS)..... | 6 |
| 2.2. Web Crawlers | 8 |
| 2.3. URL Rewriting in HTTP session | 9 |
| 2.3.1. HTTP Session..... | 9 |
| 2.3.2. Session ID..... | 12 |
| 2.3.3. Cookie..... | 14 |
| 2.3.4. URL Rewriting | 15 |
| 2.4. Related work..... | 16 |
| Chapter 3 System Architecture..... | 20 |
| 3.1. Overview | 20 |
| 3.2. Roles | 22 |
| 3.3. Network Crawling Units..... | 23 |
| 3.3.1. Overview | 23 |

| | | |
|---|---|-----------|
| 3.3.2. | Crawling Abstract Behaviors | 25 |
| 3.3.3. | Common Spider Unit Architecture..... | 27 |
| 3.4. | User Interaction Units..... | 28 |
| 3.4.1. | Overview | 28 |
| 3.4.2. | Web User Interface..... | 29 |
| 3.4.3. | Configuration..... | 30 |
| 3.4.4. | Typical Scenario | 30 |
| 3.4.5. | Project Settings | 31 |
| 3.4.6. | Multi-Steps Configuration..... | 33 |
| 3.5. | Content Publish Units..... | 35 |
| Chapter 4 Detailed Design & Algorithms | | 37 |
| 4.1. | Crawling and Tracking..... | 37 |
| 4.1.1. | Overview | 37 |
| 4.1.2. | Crawling | 38 |
| 4.1.3. | Tracking..... | 40 |
| 4.2. | Tracking in HTTP Session with URL Rewriting | 46 |
| 4.2.1. | Finding Session ID | 47 |
| 4.2.2. | Tracking Algorithm | 54 |
| 4.2.3. | Example | 57 |
| Chapter 5 Implementation | | 65 |
| 5.1. | Environment | 65 |
| 5.2. | Network Crawling Units: use HTTP and FTP..... | 66 |
| 5.3. | Internal Database | 69 |
| 5.4. | Web Server with PHP for User Interaction Units..... | 73 |
| 5.5. | Content Publisher Units..... | 76 |
| 5.5.1. | Log Viewer..... | 77 |

| | |
|---|------------|
| 5.5.2. Content Publish | 78 |
| 5.6. Create Projects, Crawl, Track, and Explore Reports..... | 78 |
| Chapter 6 Evaluation..... | 88 |
| 6.1. Performance of Find Fixed i-length CS Algorithm..... | 88 |
| 6.2. Correctness Evaluation of Finding Session ID Algorithm | 91 |
| 6.3. Evaluation of Multi-Steps Configuration | 95 |
| 6.4. Evaluation for Smart Crawler Tracking Actually..... | 102 |
| Chapter 7 Conclusion..... | 106 |
| 7.1. Conclusion..... | 106 |
| 7.2. Future Works | 106 |
| Bibliography | 108 |



List of Figures

| | |
|---|----|
| Figure 2-1 the relationship between data, information, and knowledge | 7 |
| Figure 2-2 the HTTP stateless connection..... | 9 |
| Figure 2-3 the states between the client & the stateful server | 10 |
| Figure 2-4 the states between the client support states and stateless server..... | 11 |
| Figure 2-5 the states between client and server both with state support | 12 |
| Figure 2-6 an example of HTTP response header | 14 |
| Figure 3-1 the scenario of using spiders/crawlers for KMS process..... | 20 |
| Figure 3-2 the architecture of Smart Crawler..... | 21 |
| Figure 3-3 Roles of Smart Crawler | 22 |
| Figure 3-4 the function of network crawling units for KMS admin..... | 24 |
| Figure 3-5 the mapping of sub units in network crawling units..... | 25 |
| Figure 3-6 the abstract behaviors of the network crawling units | 26 |
| Figure 3-7 the abstract behaviors of the network crawling units | 28 |
| Figure 3-8 the processes of using web UI to exec jobs on C, D..... | 29 |
| Figure 3-9 the sequence of using web UI to control network crawling units..... | 31 |
| Figure 3-10 the sequence of using web UI to control Smart Crawler | 32 |
| Figure 3-11 the sequence of using web UI to control Smart Crawler | 32 |
| Figure 3-12 the sequence of using web UI to control Smart Crawler | 33 |
| Figure 3-13 the sequence of using web UI to control Smart Crawler | 35 |
| Figure 3-14 the process of content publish unit | 36 |
| Figure 4-1 the tree structure of crawling and tracking | 38 |
| Figure 4-2 the state diagram of crawling..... | 38 |
| Figure 4-3 the mirror algorithm..... | 39 |

| | |
|---|----|
| Figure 4-4 the structure of Mirrored list..... | 40 |
| Figure 4-5 the list algorithm | 41 |
| Figure 4-6 the structure of the listed list..... | 41 |
| Figure 4-7 the state diagram of tracking..... | 42 |
| Figure 4-8 the tracking scenario for list support protocol | 43 |
| Figure 4-9 the updating algorithm for list support protocol | 44 |
| Figure 4-10 the structure of Updated List | 44 |
| Figure 4-11 the tracking scenario of non-list support protocol | 45 |
| Figure 4-12 the updating algorithm of non-list support protocol | 46 |
| Figure 4-13 the common substrings | 49 |
| Figure 4-14 find all common substring with the common way..... | 50 |
| Figure 4-15 the algorithms using KMP for common substring..... | 51 |
| Figure 4-16 the algorithms using BM for common substring | 52 |
| Figure 4-17 we use BM for our L length common string finding algorithm..... | 52 |
| Figure 4-18 the algorithm of get session id..... | 53 |
| Figure 4-19 the pseudo code of tracking for URL rewriting..... | 56 |
| Figure 4-20 the pseudo code of GET_DOCUMENT | 57 |
| Figure 4-21 the pseudo code of GET_ALL_LINKS..... | 57 |
| Figure 5-1the scenario of Smart Crawler environment | 65 |
| Figure 5-2 the architecture of the network crawling unit | 67 |
| Figure 5-3 the states in network crawling unit component | 67 |
| Figure 5-4 the state sequence of a FTP mirror project | 68 |
| Figure 5-5 the ER model of the internal database | 69 |
| Figure 5-6 the tree structure of web pages | 74 |
| Figure 5-7 the architecture of the Log Viewer..... | 77 |
| Figure 5-8 the architecture of the Content Publisher..... | 78 |

| | |
|---|-----|
| Figure 5-9 login page..... | 79 |
| Figure 5-10 the main page after login successfully | 79 |
| Figure 5-11 create a new project | 80 |
| Figure 5-12 create new sources | 80 |
| Figure 5-13 create new basic preference | 81 |
| Figure 5-14 create the project..... | 82 |
| Figure 5-15 the configuration page | 83 |
| Figure 5-16 the detail configuration | 84 |
| Figure 5-17 run a selected project | 85 |
| Figure 5-18 the execute result | 86 |
| Figure 5-19 the report query result..... | 86 |
| Figure 5-20 the result report..... | 86 |
| Figure 5-21 the content of downloaded data | 87 |
| Figure 6-1 the result diagram of three comparison..... | 90 |
| Figure 6-2 the configuration GUI of JoBo | 95 |
| Figure 6-3 the seed configuration screenshot of Teleport Ultra | 96 |
| Figure 6-4 the download rules screenshot of Teleport Ultra | 97 |
| Figure 6-5 the adding new task screenshot of Web Scraper Plus+..... | 97 |
| Figure 6-6 the seeds configuration screenshot of Web Scraper Plus+..... | 98 |
| Figure 6-7 the download rules screenshot of Web Scraper Plus+ | 98 |
| Figure 6-8 the form handler screenshot of Web Scraper Plus+ | 98 |
| Figure 6-9 the main setting screenshot of PhoPicking | 99 |
| Figure 6-10 the first level configuration..... | 100 |
| Figure 6-11 the second level configuration | 101 |
| Figure 6-12 the difference between copies of the website 1 | 104 |
| Figure 6-13 the differences between copies of the website 2..... | 104 |



List of Tables

| | |
|--|-----|
| Table 2-1 the comparison of the products and our Smart Crawler | 19 |
| Table 4-1 the result of finding old session id candidates in our example..... | 60 |
| Table 4-2 the result of finding new session id candidates in our example | 61 |
| Table 6-1 the performance of finding fixed i-length algorithms for website 1 | 89 |
| Table 6-2 the performance of finding fixed i-length algorithms for website 2 | 89 |
| Table 6-3 the performance of finding fixed i-length algorithms for website 3 | 90 |
| Table 6-4 the evaluation result of calculating session id with networking..... | 93 |
| Table 6-5 the evaluation result of calculating session id without networking..... | 94 |
| Table 6-6 the comparison of calculation time and network time | 94 |
| Table 6-7 the crawling and tracking result of the website 1 | 102 |
| Table 6-8 the crawling and tracking result of the website 2..... | 103 |

Chapter 1 Introduction

1.1. Preface

With knowledge-based economy coming, people in academia or in enterprises realize the power of knowledge. There are more and more organizations that use the KMS (knowledge Management System) to automatically create the value of information and knowledge from the experiences and data belong with their selves. Because of the exploded growing up of the Internet, we can say the Internet itself is a super large digital library. People also can use KMS to create the value of the Internet. The created value of the knowledge does increase the competitiveness and bring the much more benefits. More and more success cases are coming out.



1.2. Motivation

In these days, the use of KMS is more and more popular. The KMS getting the input data could tell the user not only what is new and what is important but integrate data input by times according the specific rules by the user. Usually the process of the KMS creating the value of the input data or information is automatically. The KMS admin just provides the input data and rules, and takes time to train the KMS to arrange data expectedly and correctly. But how does the user collect the input data for the KMS? If there are a lot of data expected? The better way is the KMS admin uses a program instead himself. The used programs are called spiders or crawlers. These programs can crawl in the Internet and download data automatically according the specific conditions. And then the data can be feed into the KMS and created its value of knowledge faster and more quickly than crawling and arranging by human beings.

The spiders or crawlers usually support some network protocols, not all. The KMS admin has to use multiple spiders or crawlers for its specific protocols. Therefore, to collect data from various data sources, the KMS admin has to use corresponding spiders, be familiar with every spider and manage each used crawler. We believe it will be better to integrate all used crawling programs into one.

Another problem is the spiders only download data from the Internet, so the admin couldn't know what is downloaded without changed, what is downloaded with changed, and what is new from the last download. It will increase the both loading in local and remote network and servers.

The other problem is the usage of spiders. Although most spiders provide the preferences for the downloading rules, for example, the network bandwidth limitation, the allowed format of data, the permitted size of document, and even the pattern of universal resource location (URL), their need of data from data sources is rear. The KMS admin just gives the start URL and the spider would crawl every document discovered from the URL. But the spider will download something that is not expected, i.e. advertisements.

Using these spiders for collecting data or tracking also provides a benefit. That is we can do some preprocesses just after automatically downloading everything from the data sources, like converting the data format, more precisely programmable content filter, or something else. But the paper won't focus the topic but it would be our future work.

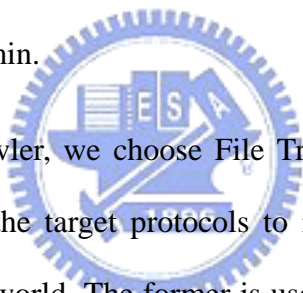
To solve these problems mentioned before, we propose a spider architecture to integrate multiple protocols and a detailed configuration called multi-steps. We also implement it called Smart Crawler.

1.3. Research Objectives

There are three main objectives: multiple protocol crawling architecture, tracking on crawled data or information, and multi-steps user interface for configuration.

Multiple protocol crawling architecture

By the growing up of the Internet, there are more and more network protocols running in the world. But each spider supports just some of them. The KMS Admin has to use one spider for one protocol in the worst case. Besides, the KMS admin should be familiar with every used crawler program and manage all of them so that he can't be concentrated on the management of the KMS. Integration all spiders into one will provide convenient and be easy to be managed by the KMS admin.



To implement Smart Crawler, we choose File Transfer Protocol (FTP) and Hyper-text Transfer Protocol (HTTP) as the target protocols to integrate. The two protocols are very simple and popular in the real world. The former is used to transfer files between computers, the later could be used in web sites and applied in every point. Such like spread news, weather prediction, and personal daily; provide platform for communities, i.e. discussion boards, forums, and Netnews; even bring the e-commerce, like shopping, stocks , and other business activities. How to integrate these two very different network protocol sliders is a big problem.

Tracking on crawled data or information

The KMS is used by more and more individuals or companies getting the value for the future. It is automatic to collecting data by spiders or crawlers and it does also be faster and quicker than crawling by man. But the program always downloads whole data and information on the data sources. So if the KMS admin doesn't pick out repeated data from the downloaded one, the KMS must process it again. It costs time whether the KMS compares the

document with the processed data or the KMS creates the value of the document. If the crawlers or spiders can distinguish the same and unchanged data, it means the input data is promised newer or updated than before downloaded, the KMS won't waste any computing time and resources. In the other side, the KMS admin can analysis the refresh/update rate of the data sources by the report of the spiders, he can change the crawling frequency to the best one of the data source. It will save the loading of the network bandwidth and servers in the both local and remote sides.

We implement not only crawling but tracking in our Smart Crawler. The tracking mechanism will provide the function that the program can find the changing status of each file from last time crawling and tell the KMS admin which files changed and which not. So the KMS admin or KMS can fetch data avoiding the repeat and unchanged one. Tracking makes the knowledge management process more efficiently and less loading with KMS.

Tracking for HTTP session with URL rewriting

The pre-condition of comparing one document between times is we can always access the document by its own URL. By this condition, we could determine the corresponding documents in different downloads and do the track process. But there are some websites which maintain http session by URL rewriting. The documents there could be access by the URL with session id that is always embedded and changing with different sessions. That will make the URLs of these documents different in each time crawling by spider. We couldn't find the pair of these documents and do the track process.

In our Smart Crawler, we propose an algorithm to solve this problem to find the corresponding pair of the downloaded documents in different time crawling.

Multi-Steps user interface for configuration

The configuration of the target documents in the existed spiders or crawlers is too simple

to have less limitation. The spider is usually given a start URL that describes where the spider starts to crawl. Although the spider might provide other settings, for example, the format of documents, the size of documents, the age (time) of documents, and the URL pattern of documents, it might download something the KMS admin doesn't want. If we consider the structure of the documents in the data source as a tree, the start URL is just one node in the tree. And then the spider would download the leaves of the node and apply other rules.

Take an example, if we would like to download special topic articles in a discussion board with membership, we have to login first, fill the keyword in the search field, and then the result articles are what we really want. For the spider, the only setting way is to give the login URL as the start URL, to set the search form, and then to crawl all finally. Some documents belong to the login page and the search pages are not needed but they still are downloaded by the spider.

So we propose the multi-steps configuration to solve this problem. The multi-steps configuration is more complexity than the original, but it provides more detail and flexibility to the spider when crawling.



1.4. Organization

In Chapter 2, we introduce the technologies of the KMS, spiders or crawlers, and discuss the related research. In Chapter 3, we illustrate the architecture of our program and components. In Chapter 4, we discuss the design issues of each topic, and how these problems to be solved. In Chapter 5, we show the implementation result of our program. In Chapter 6, we show some evaluation about our solutions. In Chapter 7, we give the conclusion and future work.

Chapter 2 Background

In this chapter, we make a brief introduction of the technologies which are involved in the paper. We introduce the technologies of the KMS and the spiders or crawlers, and discuss about the URL rewriting in HTTP session. Besides, related work would also be discussed.

2.1. Knowledge Management System (KMS)

Before introducing the knowledge management system (KMS), the first thing is to understand knowledge and knowledge management (KM). People always put knowledge, information and data together. Information and data are used interchangeably in many contexts, and the terms knowledge and information are frequently used for overlapping concepts.

Data is the fact without organized and processed. It is static. Information is original from the word inform. It means to give shape to. Information represents to give shape to what we observe from our eyes. Unlike data, information has the understanding relations [1]. It means the mean, the purpose, and the relationships. From data to information, we can use 5C to describe: Condensation, Contextualization, Calculation, Categorization, and the Correction [1]. The process only keeps the topic-concerned data; analyze them and its analyzed result; finally drop the wrong and mistake out. The thing left at last is information. What is knowledge? Knowledge is the framed experience, the value, the contextual information, the expert insight, and the grounded intuition [2]. But the definition of knowledge is still not clearly defined. So we can say in brief the information that provides the decision of the behaviors is knowledge. Generally speaking, Figure 2-1 shows a graph of the relation of data, information and knowledge.

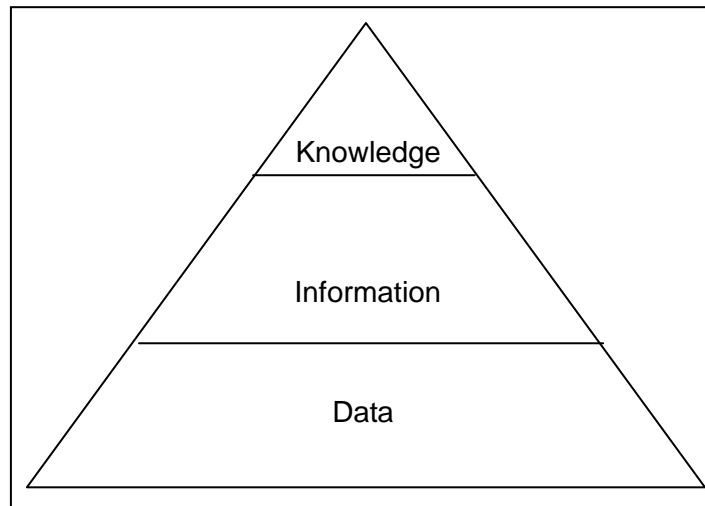


Figure 2-1 the relationship between data, information, and knowledge

The main idea of knowledge management (KM) is to assist the company or organization to understand the market trend; to increase the working ability of employees; to create the business value for the enterprise [3]. KM is widely used. Hence, KM here we talk about is only collecting, classifying, and searching. So the program that does the KM process is knowledge management program; the system that does the KM process is knowledge management system.

A knowledge management system (KMS) is a distributed hypermedia system for management knowledge in organizations. The target of the system is to provide the useful and valuable information. It might retrieve information automatically and remind the user what is new, what is useful, and what he should focus and understand. Besides, the system can learn the experience from the past process as feedback to find more knowledge. The system might also filter the data and information for the user. KMS user could learn and be aware of things thought the help of the system. The manager who manages the KMS is called KMS admin. He maintains the operation of the system and sets the specified process logic to the KMS. And he also maintains the output of the KMS to ensure the result would be satisfied and expected for the KMS user.

2.2. Web Crawlers

A web crawler is a program which browses the World Wide Web in a methodical, automated manner. It is also known as a web spider, a web robot or an automatic indexer. Web crawlers are mostly used to create a copy of all the visited pages for later processing by a system that will have its particular purpose and process the downloaded pages with its logic, such like a search engine that provides a quick and faster searches or a KMS that gather knowledge from downloaded pages. Crawlers can also be used for automating maintenance tasks on a Web site, such as checking links or validating Hyper-Text Markup Language (HTML) code [7]. Also, crawlers can be used to gather specific types of information from Web pages, such as harvesting e-mail addresses.

A Web crawler starts with a list of Universal Resource Locations (URLs), called the seeds or start URLs. As the crawler visits these URLs, it identified all the hyperlinks in the page and adds them to the list of URL to visit, called the crawl frontier. URLs from the frontier are recursively visited according to a set of policies. The simplest policy is the recursive level or the max depth. If we treat the frontier as a tree structure with the seed as the root node, the max depth is the height of the tree and all the pages are the leaves in the tree connected with its parent nodes that are the pages containing the links of their child nodes.

Generally speaking, for other network protocols we also can design their crawlers. Just like the Web crawlers, we can give the resource locations as the seeds to them and they will start to copy and download data on the remote side. If there is any new resource location found, they add it to its frontier and continue visiting according to a set of policies. Also, the downloaded data could be processed later by systems with expected purposes. For example, a crawler designed for File Transfer Protocol (FTP) [9] is called FTP crawler.

2.3. URL Rewriting in HTTP session

2.3.1. HTTP Session

With the great growing up of the Internet, there are more and more Web applications and services. As we know, HTTP is a stateless network protocol. The advantage of a stateless protocol is that hosts don't need to retain information about users between requests. But it also is the disadvantage for some purposes. For example, there is an on-line bookstore that provides the book buying service in the Internet. If the Website can't identify the customers from a lot of requests, the book sellers have no idea about which book should be sent to which customer. Figure 2-2 shows a normal stateless HTTP connection.

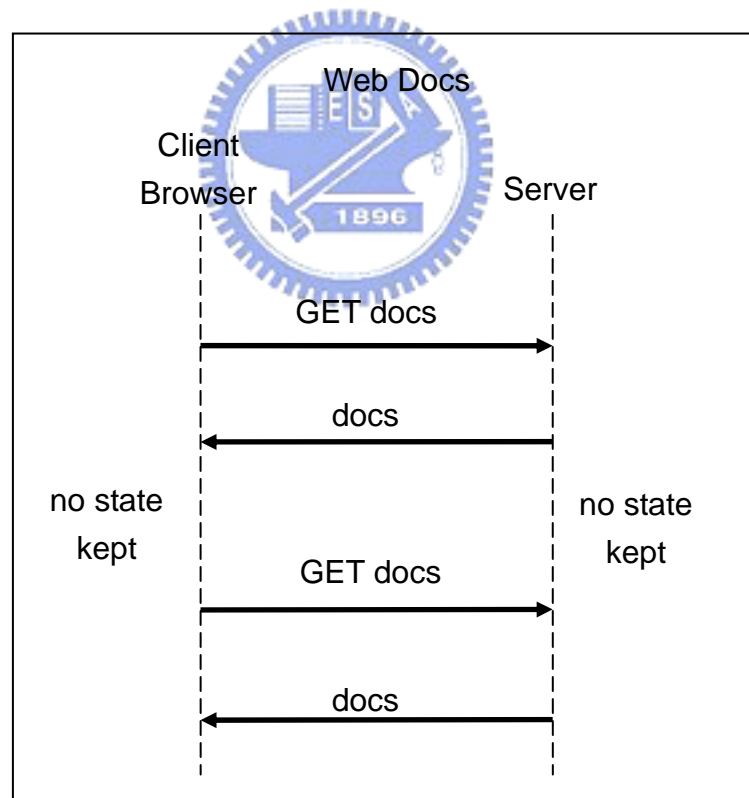


Figure 2-2 the HTTP stateless connection

If the state can be kept and maintained between the server and the client, what the customer books can be known by the server. So the seller can know which book is booked by

which customer. The on-line book selling service is on-line as we see today. In order to solve this kind of the problems, the HTTP session [4] is designed. The HTTP session has to have some features: (1). each session has a start and an end; (2). each session is relatively short-live; (3). either the user agent or the original server may terminate a session; (4). the session is implicit in the exchange of state information. The main purpose is the connection state can be saved or maintained between requests and responses so that the Websites or Web application can finger out each connection of its client, and the original benefit of HTTP is kept. In order that, Web application are expected to maintain states. It means that the server support states if the Web application maintain states.

There are various ways to reach this goal. First, we can design a stateful server that the server will handle the state changes and maintenances. Figure 2-3 shows that. The client doesn't know the state of the connection. For each request, the server will record the state change and send back the response. In the period, the server will keep the state maintained until next request.

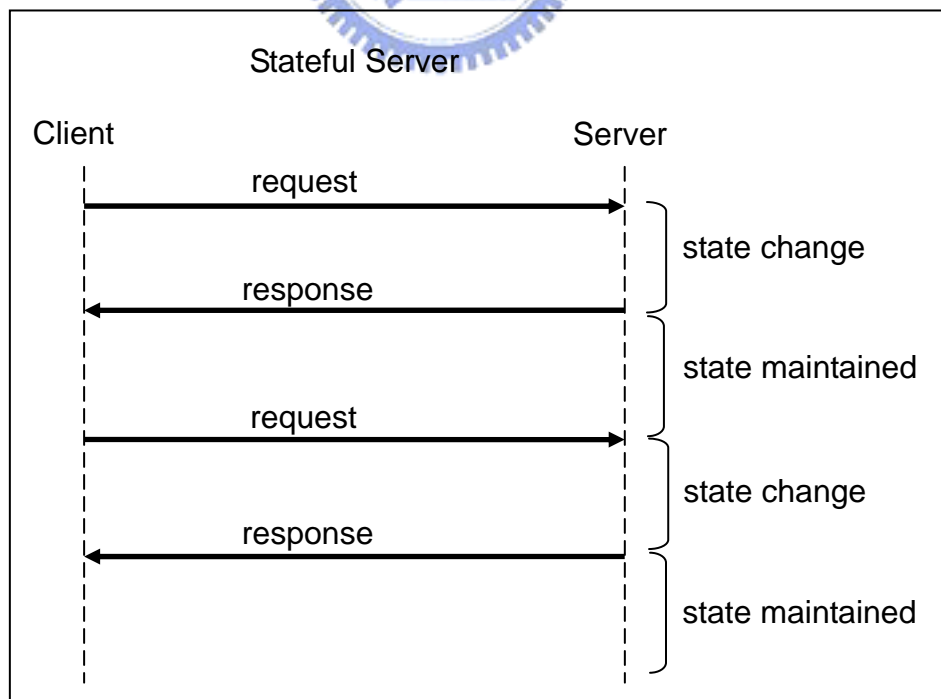


Figure 2-3 the states between the client & the stateful server

We also can let the client to maintain all of the states. Before each request, the client will record the state change and then sent it out with the request. After receiving the request, the server knows the state of the client and send the response back but the state is lost in the server. At the same time, the client will maintain the state until receiving the response from the server; and then do the process again. Figure 2-4 shows that.

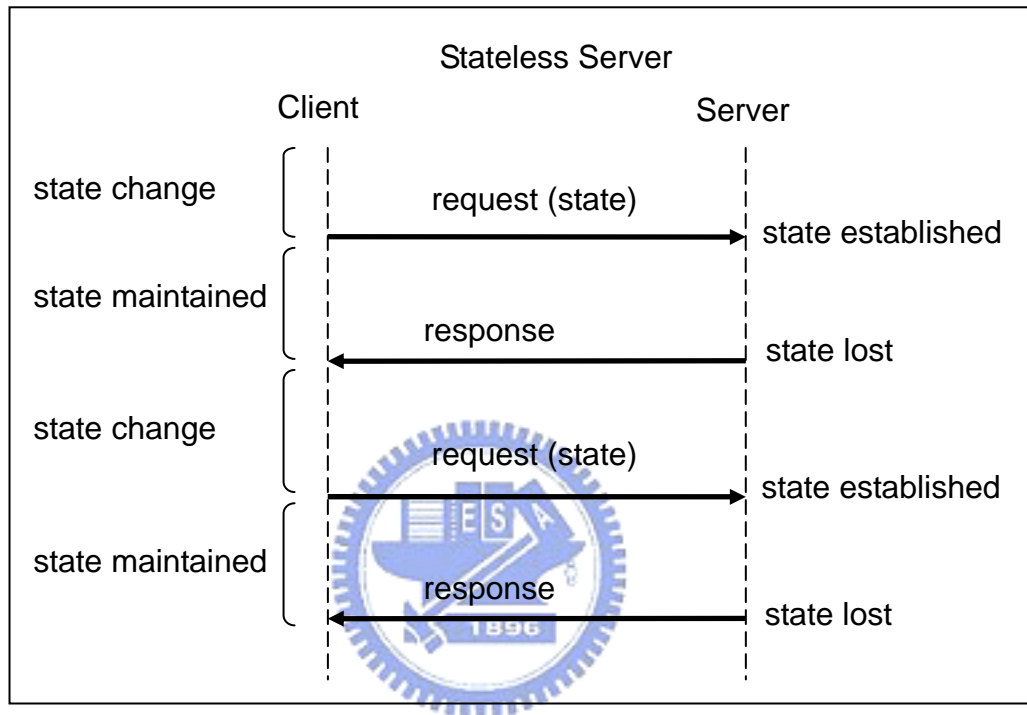


Figure 2-4 the states between the client support states and stateless server

There is another solution used today is between the two mechanisms. The server can maintain all of the states but the client maintains partial. Typically, the client or the browser cannot keep all of the states but it keeps and returns a key for the server. First, the client might send request with the state information; the server will establish the full state according the key sent from the client. And then the server will process the request and change the state. Before sending the response, the server will save the state. In the response, there is not all information of the state but partial as long as the client understands. So the client will maintain the state on his side and send it back with the next request. The state will be restored on the server and then do the same thing again. The key used here might be the address of the

client, the username, the random long string, or something else. It has to have the feature that the server can make out the client sending the current request. Figure 2-5 shows that.

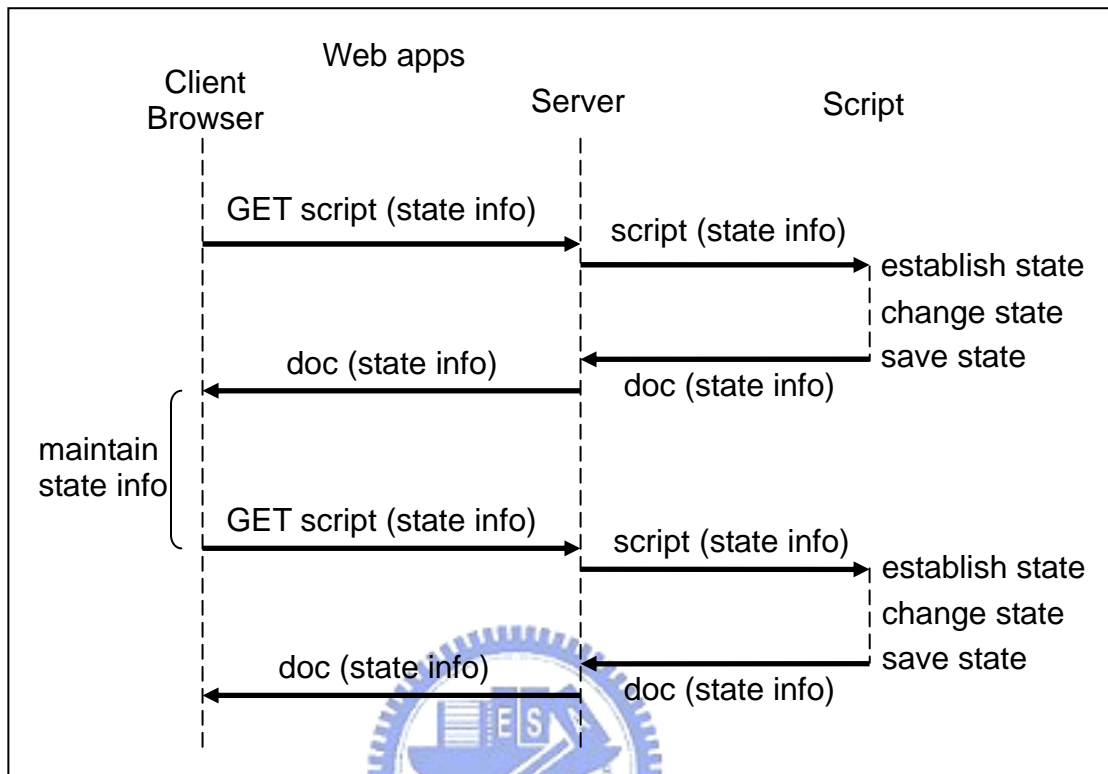


Figure 2-5 the states between client and server both with state support

There are still many ways to maintain HTTP session. We will introduce the two most popular solutions later. One is cookie that is based on the third mechanism and the other is URL rewriting that is the first one. Another difference of maintaining the state between them is one uses the HTTP header and the other uses the URL in the documents. We will discuss more in next section.

2.3.2. Session ID

In Figure 2-5, it uses a key sent to the client to transmit the state. One kind of the key popularly is the session id. When a client accesses the server, the server gives the session id to the client. In the requests and responses later, the server will use the session id to recognize the client and find the resources of the client stored in the server. Formally, the form of the

session id (SID) [5] is defined as:

SID:type:realm:identifier[-thread][:count]

Where the fields *type*, *realm*, *identifier*, *thread*, and *count* are defined as follows:

type: type of session identifier. The field allows other session identifier types to be defined.

realm: Specifies the realm within which linkage of the identifier is possible. Realms have the same format as DNS names.

identifier: Unstructured random integer specific to realm generated using a procedure with a negligible probability of collision. The identifier is encoded using base 64.

thread: Optional extension of identifier field used to differentiate concurrent uses of the same session identifier. The thread field is an integer encoded in hexadecimal.

count: Optional Hexadecimal encoded Integer containing a monotonically increasing counter value. A client should increment the count field after each operation.

Here is a formal example of the session id:

SID:ANON:www.w3.org:j6oAOxCWZh/CD723LGeX1f-01:34

From that, we can see the *type* is ANON, the *realm* is www.w3.org, and the *identifier* is j6oAOxCWZh/CD723LGeX1f with the *thread* 01 and *count* 34.

Besides, the session id must follow the format of Universal Resource Identifier (URI). SO It has to use the normal characters, not special characters like ';', '/', '&', '%', '?', and '=' [6].

The session identifier not only identifies the client but also avoid something evil. Because of the session id stands for the client identifying the resource that might is the private data or privacy information, i.e. credit card number. Someone who is not the user of the client might use the session id to connect to the server and get the private data not belong him. It is very dangerous. So the session identifier should have prevention of replay attacks. In the implement issue, a convenient method of constructing session identifier which does not require extra storage on the server is to encode the real key and other information by a cryptographically secure ensure one way function such as hash or more complex algorithm.

In most real Web application nowadays, the session id is the identifier only. It might be

appeared in the URL embedded with the variable PHPSESSIONID, jsessionid or sid, and it also might be added in the HTTP cookie with the variable session-id. Because of the session id, the Web application can differentiate each client and provide more and more personalized and customized services.

2.3.3.Cookie

An HTTP cookie or Web cookie is a short message in the HTTP header sent by a server to a Web browser and then sent back by the browser every time it connects to the server until the user interrupt. It is used for authentication, tracking, and maintaining some special purpose information for users such like the customization, personalization and connection states. For examples, the Wikipedia Web site allows authenticated users to choose the webpage skin the like best; the Google search engine allows user to decide how many search result per page they want to see. The HTTP cookie establishes the relations between requests and responses. Without cookies, retrievals of a Web page or component of a Web page are isolated events with each other.

Figure 2-6 shows an example of response HTTP header. The field Set-cookie records the cookie sent from the server. The cookies contains a lot of information like the session-id-time, path, expire time, session id, etc. Another state management header is Cookie.

```
Date : Fri, 31 Mar 2006 14:23:54 GMT, Fri, 31 Mar 2006 14:23:54 GMT
Server : Server, Server
x-amz-id-1 : 1HJGTP2TW1N8WQDKJCZ4, 1BJDC3X7TARZVD8XAVN0
x-amz-id-2 : Er8dvB5cTBYvhDs2g48KS29VGkslRkL+, HpuSjPiMWFjPvwykC9jYYOyLe505cJ2h
Set-cookie : session-id-time=11443968001;
             path=/; domain=.amazon.com;
             expires=Fri Apr 07 08:00:00 2006 GMT,session-id=102-0867729-4416118;
Location : http://www.amazon.com/gp/homepage.html/102-0867729-4416118
Vary : Accept-Encoding,User-Agent, Accept-Encoding,User-Agent
Content-Type : text/html; charset=ISO-8859-1, text/html; charset=ISO-8859-1
Content-Length : 0
Cneonction : close
nnCoection : close
Transfer-Encoding : chunked
```

Figure 2-6 an example of HTTP response header

The two state management headers, Set-cookie and Cookie [8], have common syntactic properties involving attribute-value (av) pairs. The token used here is a sequence of non-special, non-white space characters. The grammar is followed:

```
av-pairs = av-pair *(";" av-pair)
av-pair  = attr ["=" value]
attr     = token
value    = word
word     = token | quoted-string
```

Attributes are case-sensitive and the value is optional in an *av-pair*. According the grammar, we can see that the *av-pair* in our example in Figure 2-6 “path=;/ domain=.amazon.com;” is legal. There are two attributes, path and domain, and their values.

By this, the state information or client customized information can be sent between the server and the browser. So the state can be maintained between the client and the server. The Web application can save the session id and other customized or personalized information in the cookie and send to the client with the message. The client might change the settings or add information in the cookie, and send it back with the request. After receiving the request, the server read the cookie and it can know which client and what he want to do.

Using cookie to maintain the state might need to use a little storage and resource on the client, but the client user won't feel any changes with the static stateless Web pages.

2.3.4.URL Rewriting

Another way to maintain the state is the URL rewriting. Unlike the cookie that needs to use some resources of the browser, the URL rewriting embeds some information into the URL. The main advantage is it doesn't use any resource on the client. The server parses the request URL and identifies the client. It will find the corresponding resource of the client and process the request expectedly. In the process, the client is not necessary to know the state of the client.

There are many ways to approach URL rewriting. For example, we can rewrite the URL with username/password. The most approach is encoding the URL with the session id. [10] It means adding the session id into the URL and the links in the every sending document. It might be the value of the variable PHPSESSIONID or jsessionid in the query string. It might be a substring in the URL mixed with the other information. Here are some examples with the marked session id:

```
http://dcs3.cis.nctu.edu.tw/phbb2/index.php?sid=b8138e6a7099286787ff3c62e3b71614
```

```
http://www.amazon.co.jp/gp/browse.html/503-5625069-4655127?node=489986
```

```
http://www.tfccs.com/index.jhtml;jsessionid=M1BMILU43NQKTKGL4L2SFEQ
```

2.4. Related work

We will introduce some exist products and tools about their functions and features. And then we will compare them with the objectives we propose.



Heritrix

Heritrix [13] is a purely Java 5.0 crawler and has tested on the Linux. It can achieve HTTP/HTTPS documents recursively and mirror thousand of independent websites and resources non-stop collection for configurable download limitations. It can fill information into the form automatically. It also respects the robot exclusion protocol. But it needs a sophisticated operator to configure crawls within machine resources, or Heritrix will exhaust everything when it is running. Besides, it only provides command line interface (CLI).

JoBo

JoBo [14] is a purely Java 1.3 crawler with both CLI with XML configuration and graph user interface (GUI). JoBo can recursively mirror HTTP documents with the user predefined depth by a given seed. Besides, it supports download limitations and the robot exclusion protocol. Jobo also provides the automated form handling and cookie support.

GNU Wget

GNU Wget [15] is a free software package written in C for retrieving files using HTTP, HTTPS and FTP. It provides the recursively crawls and converts the absolute links to the relative one. So the downloaded documents will be links with each other. It also support automated form filler and cookie. But it only provides the CLI and has to be set for each job.

Open WebSpider

Open WebSpider [16] is an Open Source multi-threaded Web spider and search engine. The crawler part is designed in C and the search part is written by PHP. The user only gives the seed to the Open WebSpider and then it will cache and index HTTP documents found. After that, the user can use the Web part to search what he wants. The configuration of Open WebSpider is few and the crawler only provides the CLI settings.

Where Spider

The purpose of the Where Spider [17] software is to provide a database system for storing URL addresses with GUI operation. The software uses a pure XML database which is easy to export and import. It is designed by C# with .NET 1.1 frameworks. It not only rips the links but provides the ability to browser the documents offline. But for crawling, the user can't determine first what is inclusive and what is exclusive.

WebScrapper Plus+

In a sentence, Web Scrapper Plus+ [11] takes data from the Web and puts it into a spreadsheet or database. It has a simple wizard-driven configuration for tasks. It provides a lot of crawl customization, like cookie, automated form handler, control depth, and number of pages. But the user have to explore the original source HTML code and find which tags have what he wants in. And the setting is very complex.

Teleport Ultra

Teleport Ultra [12] is a very popular all-purpose high-speed tool for getting data from the Internet. It can download all or part HTTP/HTTPS of a website with the user's restriction to the local and save with the rewrite relative links or original directory tree structure. It can borrow the browser's cookie cache that letting the user performs complex authentication with his browser first and then crawling by it. It also can synchronize the offline copy with the remote side.

PhoPicking

PhoPicking [18] can download images and only images from the Internet albums. It uses the silver key to describe the wanted images and the operations of download. Not only the user can create his own silver key and share it but he can use the shared key created by someone else so that sharing the album images and crawling operations. Besides, the user can modify the exist key for new albums. But the process of settings is very complex and it needs many times practices. The user should understand the HTML document source and find what he really wants himself step by step. The user should point out which tag should be noted and crawled. The user also should point out what text in the document standing for the author, the title, the description of the target images.

Google Mini

Google Mini [19] is an integrated hardware and software solution designed to help the organization make the most of its digital assets. By setting the seeds, Google Mini will crawl everything found and indexes them in its system. It provides the search interface just like Google Search for these indexed data. It also provides the reporting for the administrators. The Google Mini summary reports include the search or query result, and the broken URL belonged in the crawling set.

Comparison

The table 2-1 shows the comparison result of these exits products and our Smart Crawler.

From the table, we can know that the tracking is lack in all the other products. These programs just only download everything that might be expected. Maybe some of them might support an interface for search. But the user can't know what's different with the last time soon. Besides, most products are only feeds one seed, called One-Steps that only describes data roughly. It is not flexible enough. It is very possible to crawl something unwanted.

| | Protocol | UI | Cookie | Form | CD rules | Indexing | Tracking | M-Steps | Report |
|--------------------------|-----------|---------|--------|------|----------|----------|----------|---------|--------|
| Heritrix | HT,HS | CLI | | Yes | Yes | | | | |
| JoBo | HT | GUI+CLI | Yes | Yes | Yes | | | | |
| GNU Wget | HT,HS,FTP | CLI | Yes | Yes | | | | | |
| Open Spider | HT | Web | | | | Yes | | | |
| Where Spider | HT | GUI | | | | Yes | | | Yes |
| Web Scraper Plus+ | HT | GUI | Yes | Yes | Yes | Yes | | Yes | Yes |
| Teleport Ultra | HT,HS | GUI | Yes | Yes | Yes | | | | |
| PhoPicking | HT | GUI | Yes | Yes | Yes | Yes | | Yes | Yes |
| Google Mini | HT,HS | Web | Yes | Yes | Yes | Yes | | | Yes |
| Smart Crawler | HT,FTP | Web | Yes | Yes | Yes | Yes | Yes | Yes | Yes |

Table 2-1 the comparison of the products and our Smart Crawler

The word “CD rules” means Customized Download rules, “HT” means HTTP, “HS” means HTTPS and “M-Steps” means Multi-Steps..

Chapter 3 System Architecture

3.1. Overview

First of all, we have to explain the scenario about our Smart Crawler. Taking a simple example, people may read news and discussion articles from the Internet and internal reports in the local FTP server. They could use a KMS to analysis these data and information, to remind them to read something recommended, and to create valuable and useful knowledge. The KMS admin uses the spiders or crawlers for collecting data.

By given required data source information and the customized conditions, the KMS admin could use the specific spider of the corresponding network protocol to mirror everything from the data sources and input to the KMS. And then, the KMS admin just package these downloaded data and output to the KMS as input. People will be notified by the KMS. The scenario is showed as Figure 3-1.

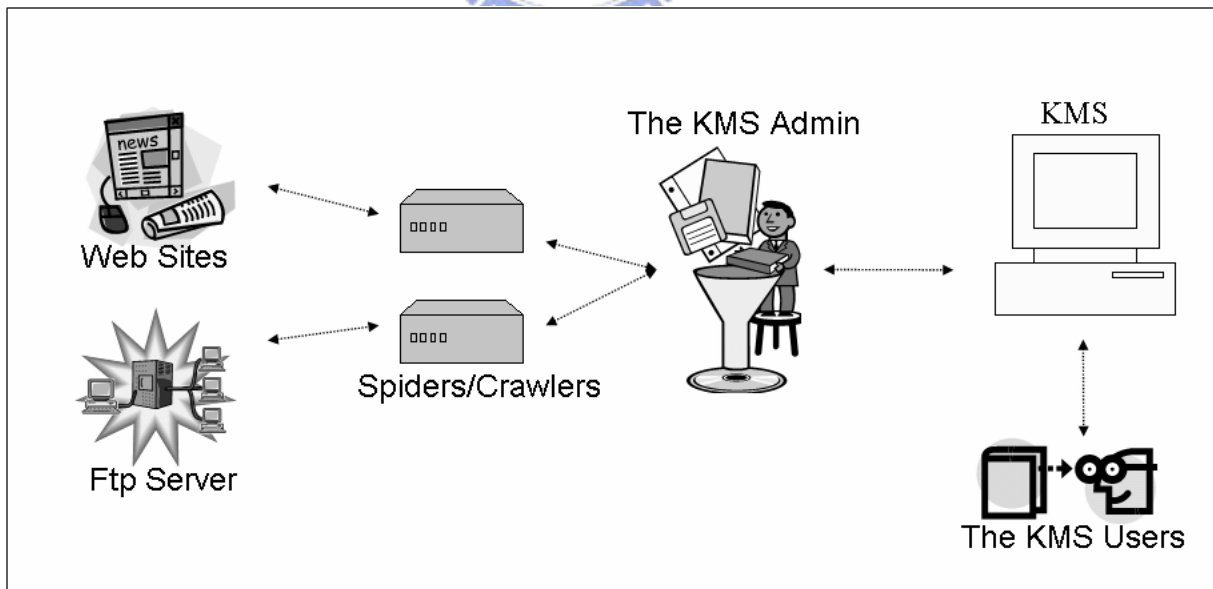


Figure 3-1 the scenario of using spiders/crawlers for KMS process

About the former, the KMS admin has to use spiders matched the specific network

protocol of the data source; these programs just download everything from the data sources without exclude the documents that was downloaded and unchanged since last time download. It will increase the load of the KMS admin, and it will also increase the load and designed complexity of the KMS. Thus, we proposed a program named Smart Crawler that provides a smart data collecting mechanism so that the KMS admin can easily collect data from various data sources with different network protocols and efficiently exclude unnecessary portion of the collected data. Otherwise, the smart crawler provides the Multi-Steps configuration user interface that help the KMS admin describes the data of the data sources more precisely. We could say the Smart Crawler makes collecting data more convenient, more efficient and more precisely.

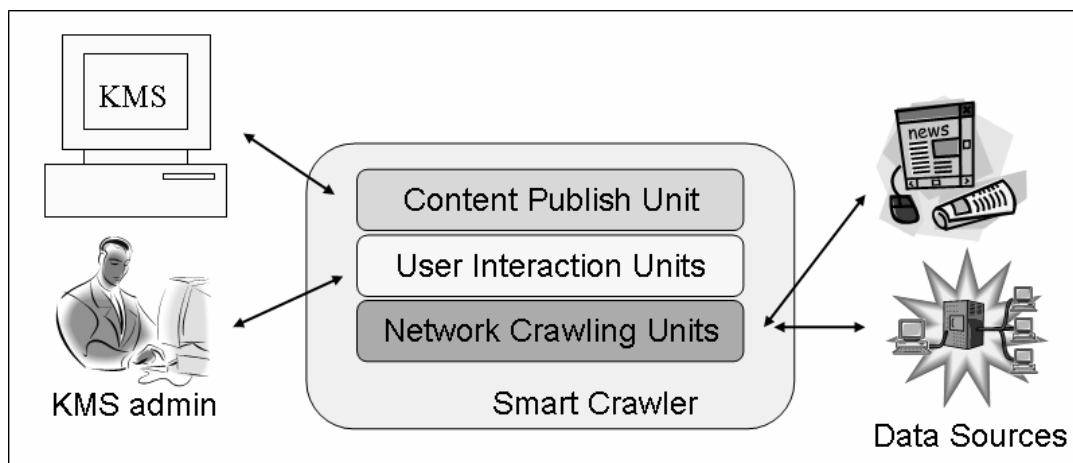


Figure 3-2 the architecture of Smart Crawler

The simple architecture of Smart Crawler is indicated as Figure 3-2. When the KMS admin has set the configuration and limitation of the data sources with the User Interaction Units, the Network Crawling Units would start to work according the specific conditions on various data sources with supported network protocols if the cache files downloaded last time were confirmed by the KMS admin. After mirroring or tracking, the KMS admin could view the report with the User Interaction Units to see if there is anything new on this crawling. If yes, according the report, the KMS could connect to the Content Publish Unit and then download the cache files downloaded by the Smart Crawler. Finally, the KMS admin has to

inform the Smart Crawler all cache files is confirmed. It means the KMS has gotten input by these files.

After explanations of the roles and terms we use, we introduces the three major part of Smart Crawler indicated in the Figure 3-2.

3.2. Roles

We define several terms that will be referred in Smart Crawler. They are schematized in Figure 3-3.

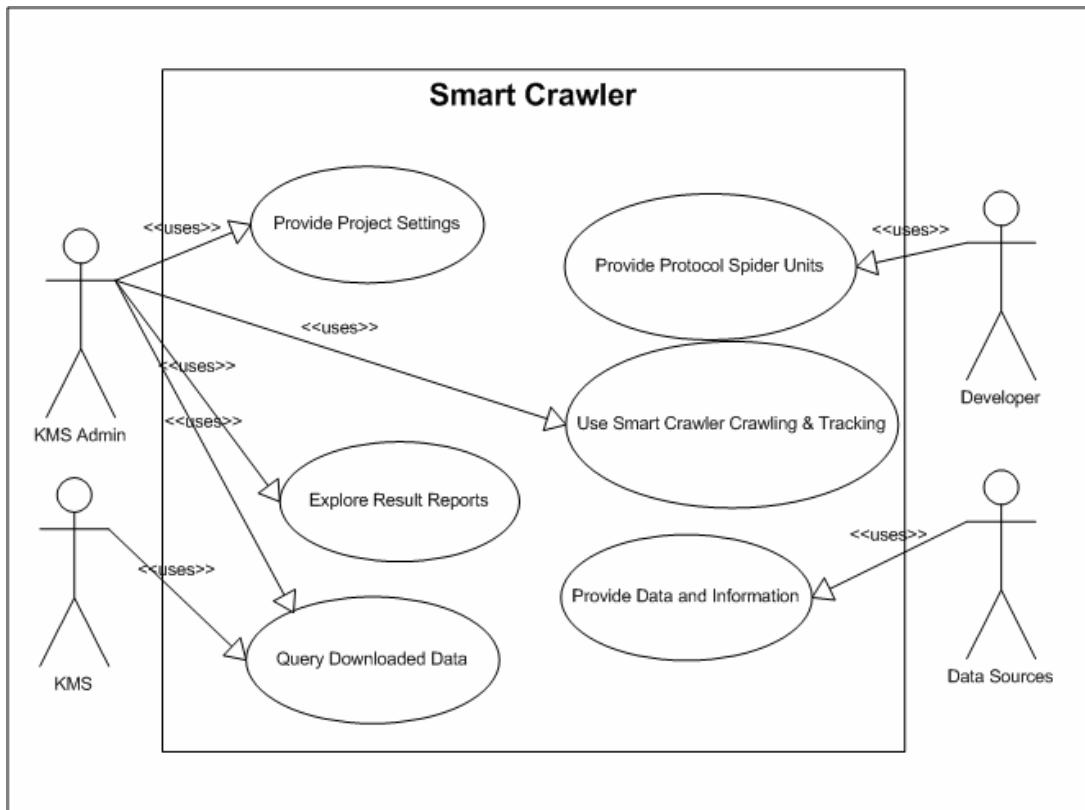


Figure 3-3 Roles of Smart Crawler

- *KMS Admin* is the manager of the KMS who uses Smart Crawler to collect data. He provides all information of project settings, trigger Smart Crawler to crawl and track of data on the data sources, and explore result reports of situations of crawling or tracking.
- *KMS* is a system that will process input data to create valuable and useful knowledge. It could query downloaded data as input and take knowledge management processes.

- *Develop* is programmer. He could program and design new component of Smart Crawler that support new network protocols.
- *Spider unit* is a component of Smart Crawler and provides the crawling and tracking ability with its supported protocol.
- *Data sources* provide services to share data and information, like website, or FTP Server. They provide data and information that are crawled and tracked by Smart Crawler. KMS would take these things as input data.
- *Project settings* represent data sources settings and Smart Crawler download limitation. There are including information of data sources, user preferences, download rules, etc.
- *Result reports* are generated by Smart Crawler after crawling or tracking. They describe the situation of downloaded data from last time download.
- *Downloaded data* is what Smart Crawler downloads from data sources.

3.3. Network Crawling Units



3.3.1. Overview

One of the significant contributions of Smart Crawler is supporting multiple network protocols. As the Figure 3-1 indicates, there may be more than one type network protocols of data sources. If developers write programs for these protocols respectively, it will cost lots of time. Hence, it is clear that we need to design the architecture to integrate different network protocols.

For normal user, like KMS admin, this part provides the abilities to crawl and track from data sources. According the features of the units: supporting multiple network protocols and easy to integrate new protocol spider unit; download data from data sources; comparing everything this time with last time and report the status, the user just provides the detail of data sources and data, such like configurations, the units would crawl or track from data

sources with various supported network protocols. Figure 3-4 describes that.

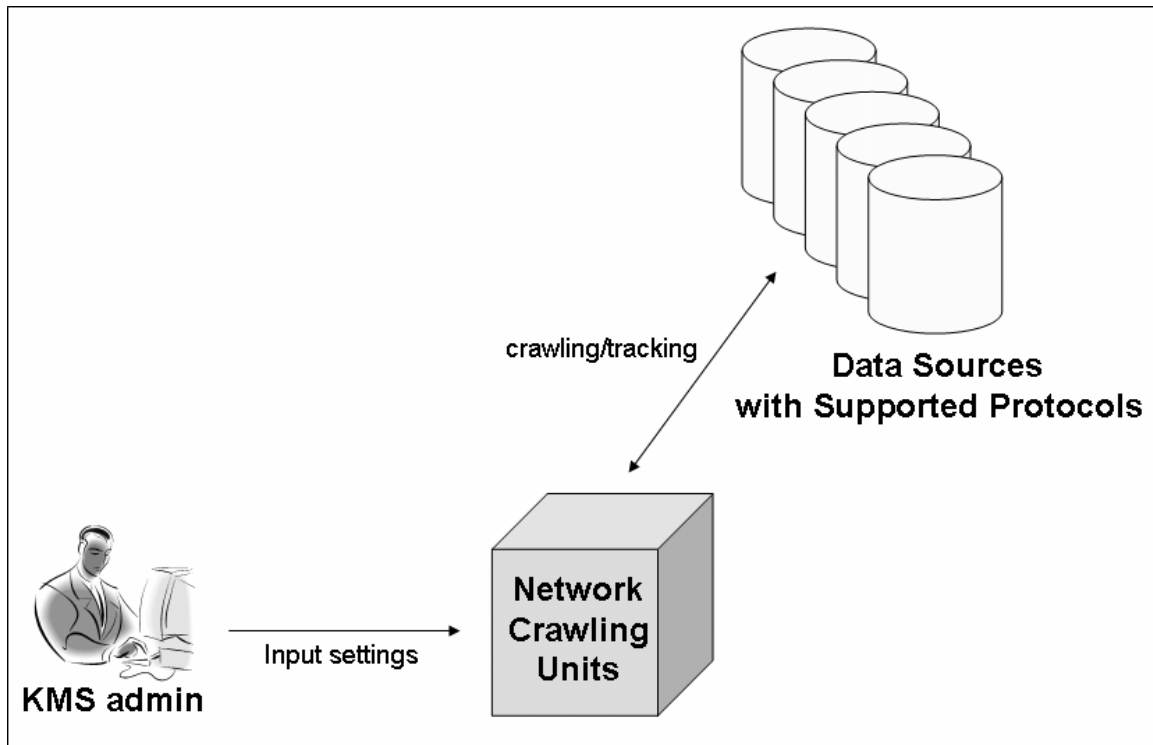


Figure 3-4 the function of network crawling units for KMS admin

For developers, they just write a program integrated in followed by the network crawling unit interface implementing these procedures; the new network protocol would be supported for crawling from data sources communicated with these protocols. In other words, the developer designs and programs a new unit that satisfied these function, i.e. get the configurations, mirror, track, by the network crawling unit architecture, it would be easily integrated and then used by Smart Crawler. It is indicated by the Figure 3-5.

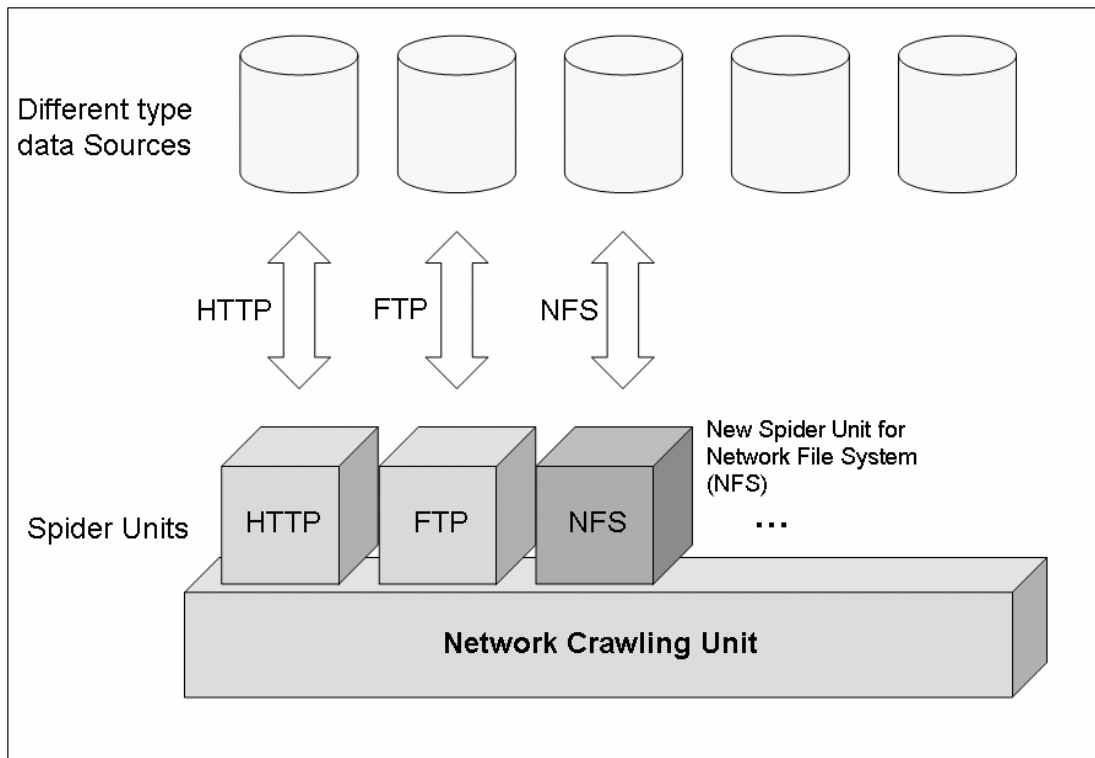


Figure 3-5 the mapping of sub units in network crawling units

In brief, the main purpose to the normal users is to provide the function of crawling and tracking; the main purpose to the developers is to provide the architecture that developers can implement a new spider unit and easily integrate in. The detail of the architecture would be discussed in the follow section.

3.3.2. Crawling Abstract Behaviors

Before we discuss how to integrate different spider units, we summarize the common behaviors of a spider crawling. There are several steps: (1) the spider is ready executed in the beginning; (2) the spider is initialed by loading the configurations about the data source and the crawling rules; (3) the spider crawls and downloads everything specified and saves all in the local disk; (4) the spider release the occupied resources, and close. In the process, we hide the detail of network communication, so we do not know how the spider negotiates in the network layer. The main idea is we propose an architecture that controls the behavior sequences of all spider units to reach the multiple protocol integration.

According to these common behaviors, the network crawling units are followed by these procedures basically and do some improvement. As Figure 3-6, at the beginning, the control part of the network crawling units is ready and executed. Next, according to the input configuration, Smart Crawler would choose which unit should be executed. Then, the unit loads the configurations and limitations. So the major work is running next: crawling or tracking. If the KMS admin would like to do crawling, the unit would execute the mirror function. If the KMS admin would like to do tracking, the unit would be executed the track function. Later the unit will record the metadata of fetched data into the database. The last step is generating reports and close. Hence, the network crawling units could be easy to support multiple protocols crawling.

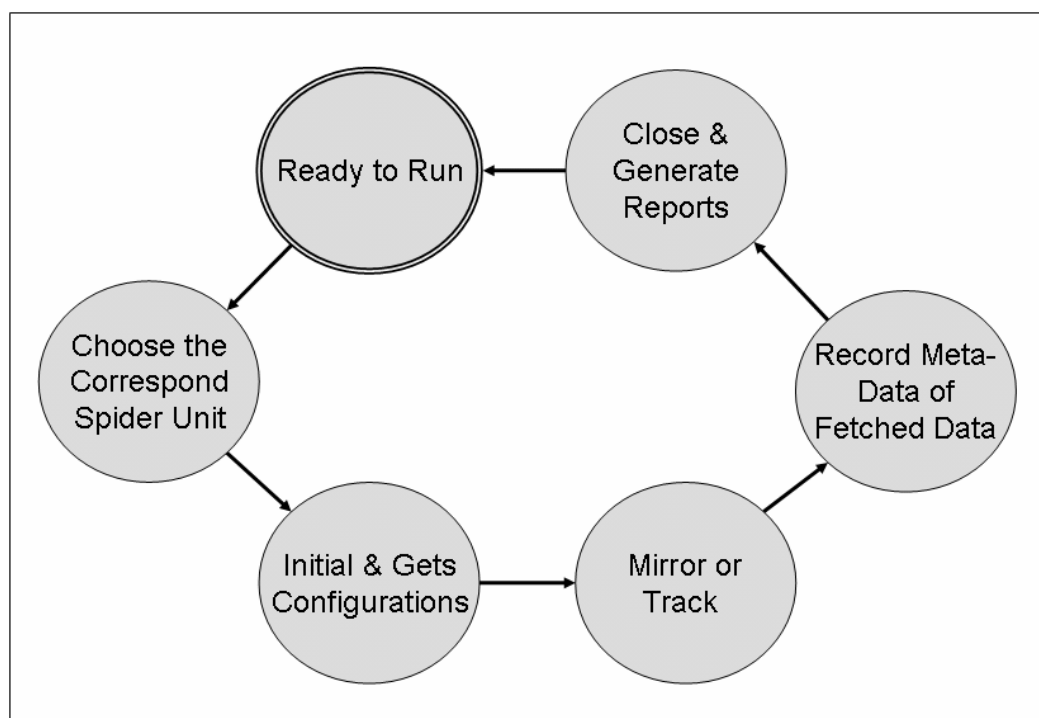


Figure 3-6 the abstract behaviors of the network crawling units

For the advantages of these abstract behaviors of spiders, Smart Crawler does not know how a network protocol working except which unit should be used for known protocol. Another key point is how to choose the right spider unit. In the internal of Smart Crawler, there would be one list recorded which network protocol should be used for which spider unit.

Smart Crawler only looks up list searching the corresponding pair and then runs. We propose a sequence state stand for these abstract behaviors of each spider unit, so the developer only implements these states of its spider unit and hiding the detail about how the network protocol work. After programming, the developer only changes a little part of Smart Crawler to integrate new unit. Then, Smart Crawler can trigger the unit by the configurations. The state of abstract behaviors would be talked in Chapter 4.

3.3.3.Common Spider Unit Architecture

To implement the states of each spider unit, we use the observer behavioral pattern [22][27]**Error! Reference source not found.**, also called Publisher-Subscriber model. Before we explain the detail about our implementation, we introduce what the model is in brief. The observer is the component which observers. When there is any change of the components, called subjects, which are listed in the observer pattern, the observer is notified and process what it is expected. The advantage of this design pattern is to program something about changing states is very convenient. The observer is always notified by subjects to realize the changing state and makes the decision what to do on that spot. Network crawling units is right with this feature. Each spider unit has its own state, such like INIT, MIRROR, UPDATE, etc. So the control part of Smart Crawler just monitors the changes and provides the right resources to the right spider unit without understanding the detail of each network protocol communication.

So in our Smart Crawler, we design an observer, called ControlObserver, which monitors its subjects, the spider units, and waits notifications. For developers, the spider unit is only implemented and integrated in Smart Crawler according the states that stand for the abstract behaviors mentioned before, called Observable. And then the ContorlObserver would attend to the progress of state changes with spider units and provide the needed information by these units. Besides, the process of crawling would be integrated with the database; it means Smart

Crawler would store the metadata of downloaded files into the database for using in the future; we also integrate database units by the design pattern. Finally, the ControlObserver would monitor the state changes of each spider unit and database units, and wait the notifications to decide what the next step is.

Hence, Smart Crawler could provide multiple protocol crawling ability easily and quickly. The architecture of Smart Crawler is showed in Figure 3-7.

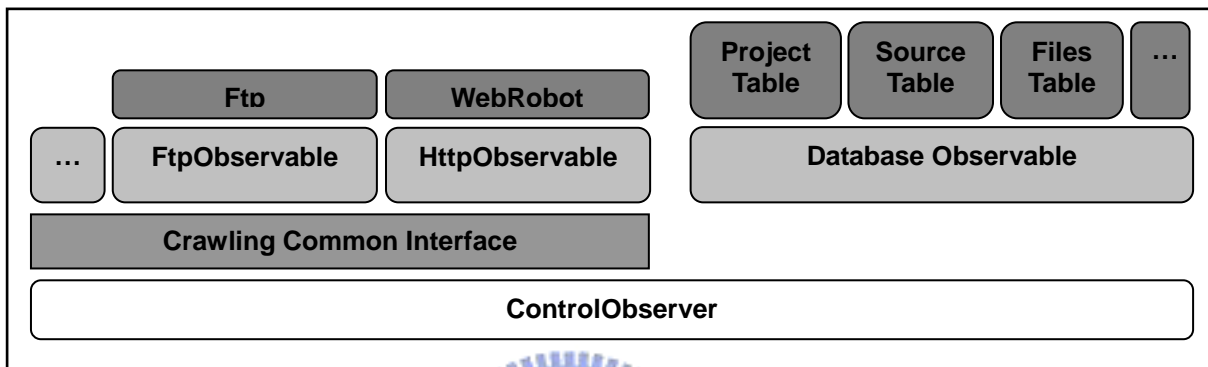


Figure 3-7 the abstract behaviors of the network crawling units

The bottom is ControlObserver which monitors changes from the upper components and decides the progress. The components in the left side are spider units: FtpObservable is the spider unit for FTP protocol, and the HttpObservable is for HTTP protocol. They are designed followed the crawling common interface that describes the states and interfaces of spider units. The component in the right side is the DatabaseObservable that provides the ability communicating with the backend database that keeps the information of files' metadata and user configurations. The information or configurations queried out by the DatabaseObservable would be passed by the ControlObserver to the right spider units when the spider unit notifies.

3.4. User Interaction Units

3.4.1. Overview

The user interaction units are responsible for interaction with Smart Crawler user, like

KMS admin. In another word, the units are just the user interface (UI) of Smart Crawler that would get all configurations and information from KMS admin and then pass these data to the network crawling units discussed before, and show the result to KMS admin after crawling work is finished, and other functions.

3.4.2.Web User Interface

Unlike common programs with console UI, we design a web UI for Smart Crawler. For programs with console UI, when there are several programs running, the manager has to use their own UI for configurations one by one, and the setting could not be used by another machines. We think it is not convenient enough, so we propose the web UI. The web UI could provides a uniform usage and passes the configuration to the backend machine that the crawling unit of Smart Crawler is running in. So the manager just uses a browser connecting to the web UI, picks up a target machine, and then he could set all information to another machine. The settings filled in the web UI are also reused by another network crawling unit by pointing out which machine is specified. We summarized the process in the Figure 3-8.

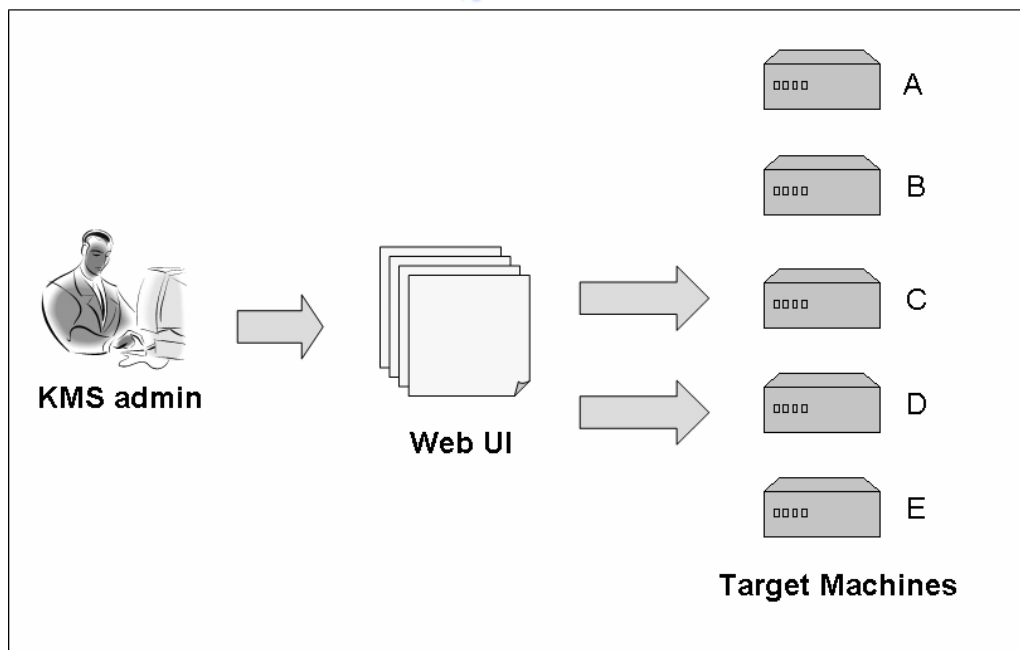
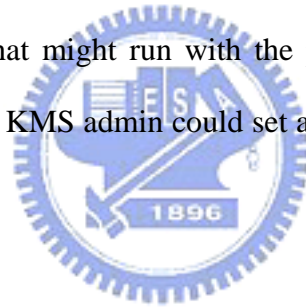


Figure 3-8 the processes of using web UI to exec jobs on C, D

3.4.3.Configuration

Although the web UI provides automatically to connect to target machines instead of the managers do themselves, there is still one problem there. That is the manager has to fill everything again when he wants to ask the program working. In another word, the web UI does not memory what the manager puts in so that he should put all in again.

For solving this problem, we use a database as a bridge. We use a database to store every details KMS admin inputs. That's why the network crawling units mentioned before have DatabaseObservable for querying user configurations. Because the database would save the user's input configurations, the user, usually KMS admin, just types in the configurations once and runs Smart Crawler many times without giving something already given last time. For network crawling units, that might run with the project settings stored in the database without the user asks. It means KMS admin could set a schedule running the specified project without executes by himself.



3.4.4.Typical Scenario

The whole process would become like this: (1) KMS admin inputs setting by web UI and these setting would be stored by a project id in the database; (2) KMS admin chooses what to do and where to do by web UI; it means KMS admin should specify which project id and which target machine; (3) the web UI pass the project id to the target program; (4) the program queries the detail settings by the project id and runs what KMS admin wants; (5) after finishing running, KMS admin could use the project id specified in step 2 to query the result report by web UI. The sequences of using the web UI is indicated in Figure 3-9.

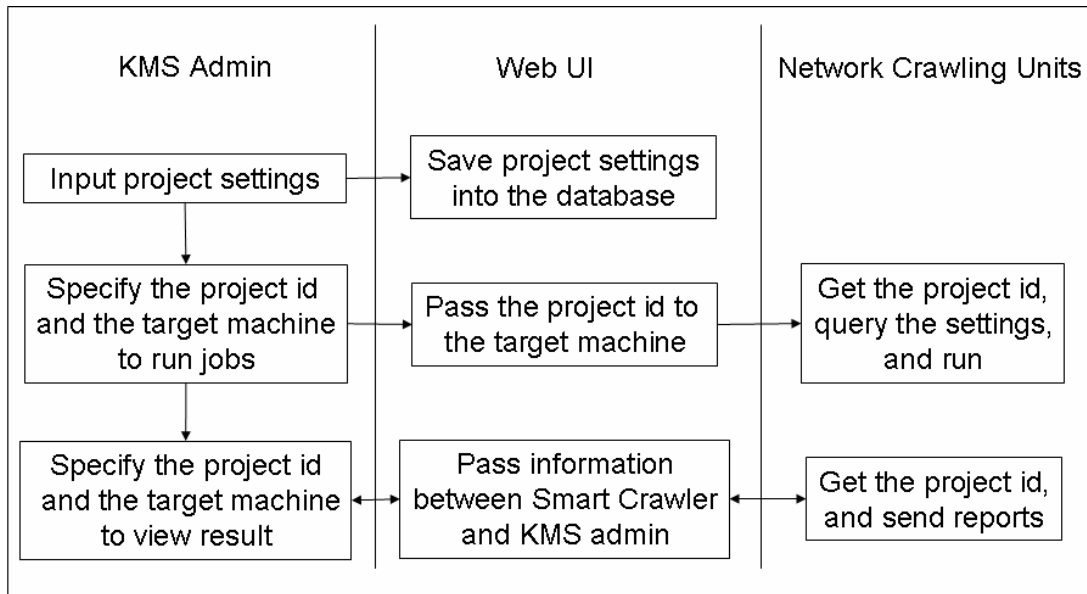


Figure 3-9 the sequence of using web UI to control network crawling units

3.4.5. Project Settings

The first step of the working process mentioned before is KMS admin inputs the project settings. Before we introduce what information should be inputted, we explain what the project is. The project we use here is KMS admin want Smart Crawler to collect information from one data source with one purpose. In brief, the project is a name of some special collected data of KMS admin. So the settings of the project are called project setting.

Project settings are including the detail information of the specified data source, the basic preference, the advanced preference, and so on. The detail information of the specified data source contains the nickname name named by KMS admin, the protocol type, the actually network hostname, the port number, and the description for comment. All are needed except the comment. The basic preference is used by every spider unit. In another word, the basic preference provides the basic configuration. It contains the username, the password, the base URL, the max depth, the type number, and the description of comment. The username and the password are only needed if Smart Crawler needs the pair to access the data source. The base URL and the max depth are essential for telling Smart Crawler where the root of the

document tree is and how many levels it should visit. The type field stands for the tiny of the network protocol. For example, the protocol HTTP is specified in the data source configuration. The type would indicate the HTTP session type is cookie or URL rewriting. The comment is always optional. We summarize these in Figure 3-10.

| Data Source Configuration | | | | | |
|--------------------------------|----------|----------|-----------|---------|---------|
| Name | Protocol | Hostname | Port | Comment | |
| | | | | | |
| Basic Preference Configuration | | | | | |
| Username | Password | Base URL | Max Depth | Type | Comment |

Figure 3-10 the sequence of using web UI to control Smart Crawler

The third part of the user input information is the advanced preferences including URL allow rules, form filling information, and multi-steps configuration. As long as KMS admin inputs the pattern and the permission for URL allow rules, Smart Crawler would follow these rules to determine which URL is allowed and which is denied. KMS admin also provides the form setting appearing in the web page and Smart Crawler would fill them automatically. The multi-steps configuration would be discussed later. Figure 3-11 indicates that.

| URL Allow Rules | | |
|--------------------------|------------|---------|
| URL Pattern | Allow/Deny | Comment |
| | | |
| Form Filling Information | | |
| Target URL | Parameters | |

Figure 3-11 the sequence of using web UI to control Smart Crawler

The URL pattern is a regular expression. The allow/deny field is true for allow and false for deny. The optional comment can be filled by the description of the rule. The target URL is the action parameter of the form tag. The parameters are the string with the form: name1=value1&name2=value2&....

3.4.6. Multi-Steps Configuration

Another contribution of Smart Crawler is Multi-Steps configuration. In the basic preferences, there is one field called base URL. The base URL is the start point to visit for Smart Crawler. And everything under that would be visited, too. This way is a traditional method and used in most spider now. If we treat data on the data source as a tree, the base URL would be the root node and all collected data would be the sub tree of the tree from the base URL node. The Figure 3-12 indicates what we talk about.

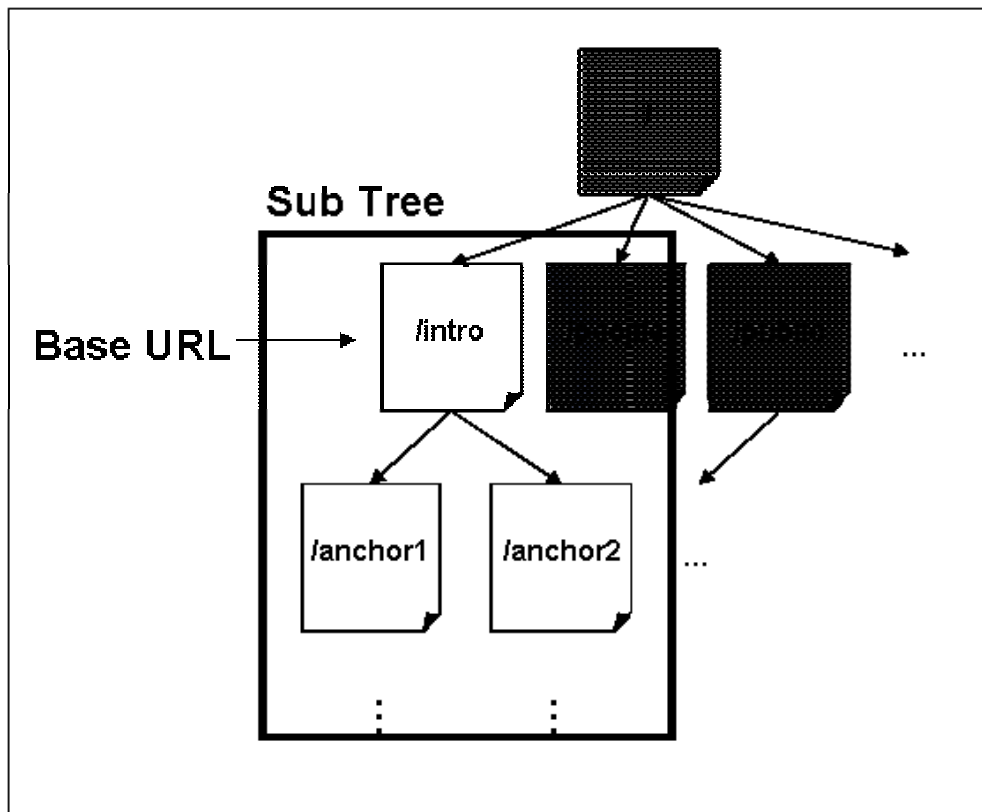


Figure 3-12 the sequence of using web UI to control Smart Crawler

But there is not everything we want actually. For example, we want to collect the articles about some topic of the Java programming language in a private forum. We have to login the forum first. Next, we choose the discuss board about Java and make a search for the special topic. Finally, what we want is coming out. If Smart Crawler only has the base URL, the base URL should be set for the login page. So everything linked from login pages would be

downloaded, not only the special topic articles. The collected data would be messy and complex for KMS. So we propose the Multi-Steps configuration.

The main idea of Multi-Steps configuration is to simulate the behaviors of human beings. KMS admin inputs not only the base URL but others URLs for steps. Take the example just talked. The base URL should be the login page and we specified the max depth is 0. It means only the page is downloaded. Another mean is Smart Crawler just finishes the login operation. The step 1 is to point the search page of the discuss board about Java and make the search. The max depth would be set n if we want to download n levels. So Smart Crawler would make the search and download everything appearing in the result.

There is another condition that the traditional way could not handle. The traditional configuration could not configure two independent sub trees or more. Take an example. In the remote FTP, there are several directories in the root, like /intro, /people, /private, and /public. If we want to collect everything contained by the sub directories /intro, and /public/doc1, we have to create two projects to handle that. Base URL is only set for one URL. In other word, base URL only stands for a sub tree, and it can't stand for more than one independent sub trees. But in Multi-Steps configuration, we can set base URL as /pub and the step 1 as /public/doc1. The max depth of both is max. Smart Crawler would collect what we really want and with nothing unexpected. We illustrate all in Figure 3-13.

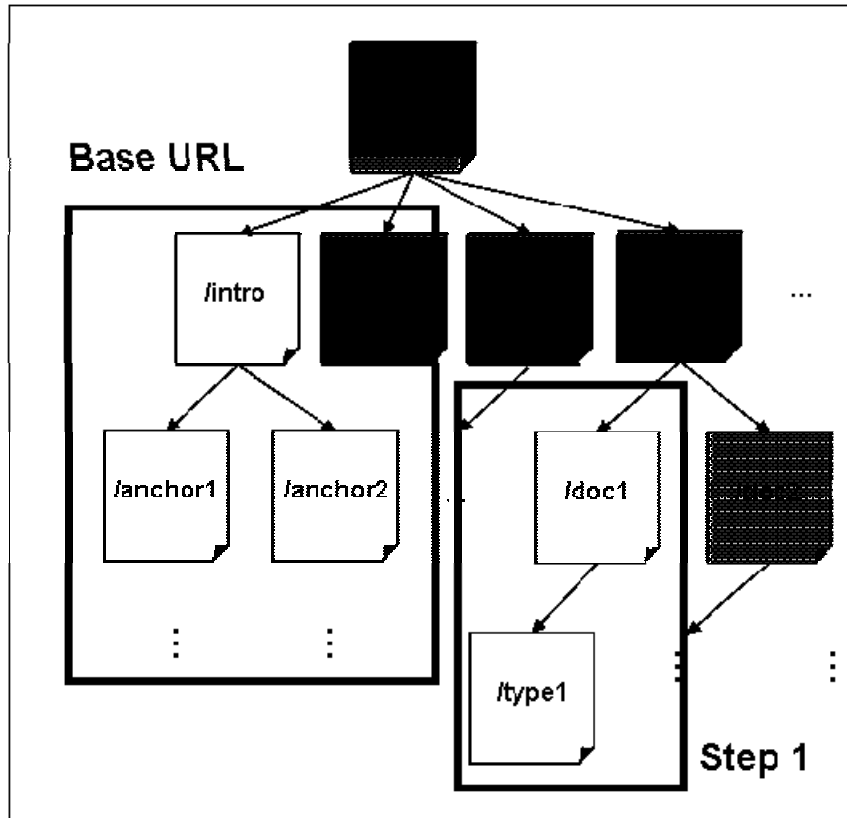


Figure 3-13 the sequence of using web UI to control Smart Crawler

We believe Multi-Steps configuration provides a flexible way to describe the sequences path of crawling although there is more complex than the traditional way. It makes Smart Crawler travel from the document root without downloading unexpected data and it also makes Smart Crawler collect multiple sub trees from the data source in one project. In other word, Multi-Steps configuration provides easier configuring way not only in the depth but in the width. For that, we believe we can describe what we expect from the data source more precisely and appropriately.

3.5. Content Publish Units

For the main target of Smart Crawler is collecting data for KMS, Smart Crawler should provide some way to let KMS gets the downloaded data. The third part is response for this. By clear purpose and one direction function, the units just get the request identified to the

downloaded data and send the content back as the response. The normal unit is called the content publisher.

The same thing happens on the result report. Although KMS admin can access the result report by Web UI, it is not convenient and lack automation telling KMS the data list manually. We propose a unit called log viewer by using the same access method with the content publisher unit to provide result reports. Actually, it is still one of the content publisher units replacing data content instead of result reports. We believe that the additional accessibility will provide more automation. For example, KMS can automatically access the result report by the log viewer and parse it. According the parsing result, KMS can automatically access data contents by the content publisher unit. The implementing issues of the unit will be discussed more in Chapter 5. We indicate that in Figure 3-14.

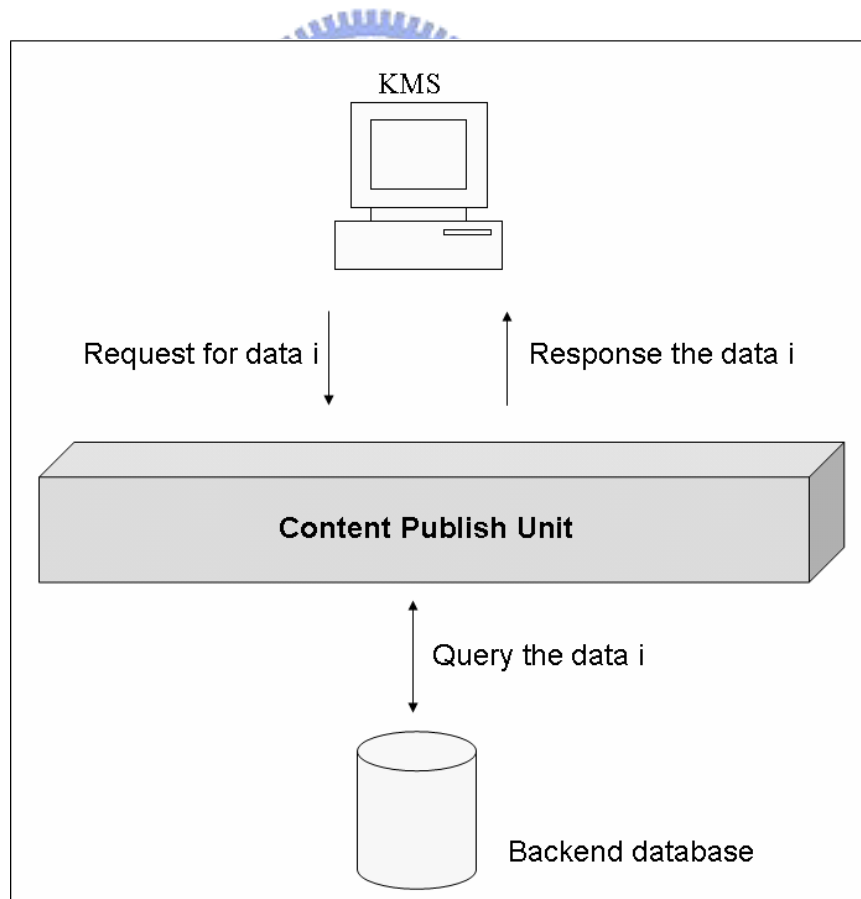


Figure 3-14 the process of content publish unit

Chapter 4 Detailed Design & Algorithms

In this chapter, we would discuss the detailed design and algorithm that Smart Crawler uses. We talk about the algorithm about crawling and tracking in the normal case first. Next, we propose a solution for the exception of our algorithm mentioned before, HTTP session with URL rewriting. That includes the solution about the HTTP session id. Finally, we take an example to explain how the algorithm works.

4.1. Crawling and Tracking

4.1.1. Overview

Before we introduce the crawling and tracking mechanisms, there are several issues should be discussed. There is one assumption we use here. As we usually know, the resource in the Internet would have a URL and we can always use its URL to access the resource in the same way. So when we download a document with its URL and save it in the local side, we can say that the document stands for the content of the URL on the remote side.

We take an example, the first page of the Distributed Computing System (DCS) Laboratory, called index.html, could be accessed with the URL <http://dcs3.cis.nctu.edu.tw/> by a web browser. We can construct data as a tree; the saved index.html stands for the node corresponding to the URL <http://www.cis.nctu.edu.tw/index.html> on the local side. So we can say the crawling activity is to rebuild a tree exactly the same with the remote side on the local side, and the tracking activity is to update the tree nodes and their content of the local side identical with the remote side. We summarize the normal case in the Figure 4-1 as followed. The right side is the remote data tree of DCS Lab. The crawling is building an identical tree such like the top left diagram. The down left diagram is tracking that updating the nodes and their content.

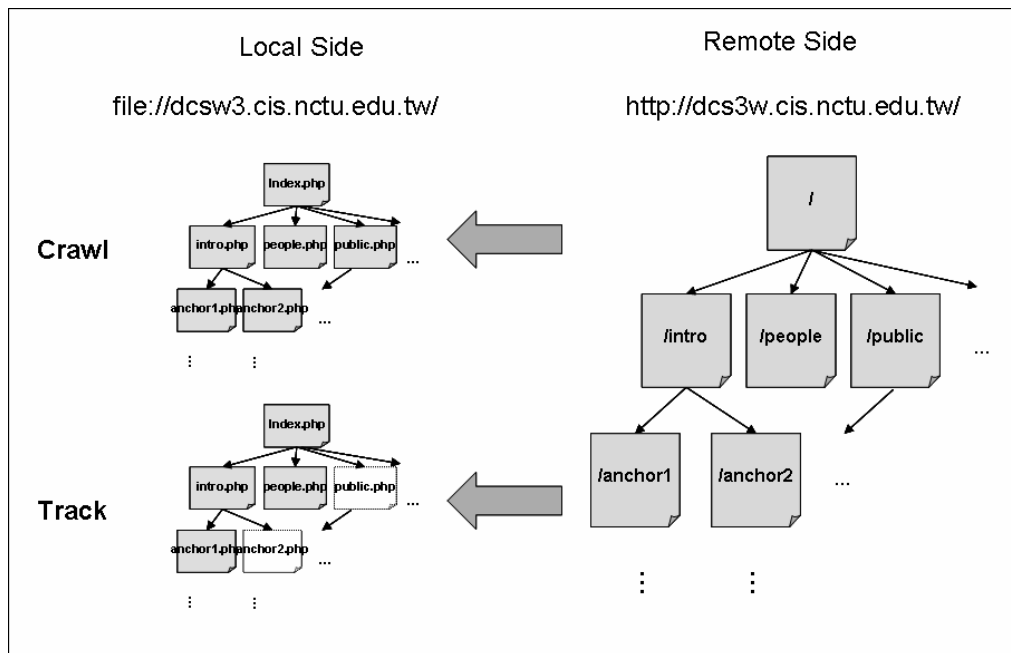


Figure 4-1 the tree structure of crawling and tracking

4.1.2. Crawling

We illustrate the abstract behaviors in Chapter 3. For integration, the spider unit should follow the state sequence with the abstract behaviors. The corresponding state sequence of spider unit crawling is quite simple. As common as the normal spiders: (1) Smart Crawler begins to be ready (Start); (2) Load the configurations (Init); (3) Crawl and download everything on the remote side and save all into the local file system (Mirror); (4) Release the resources (Close). It is summarized in the Figure 4-2.

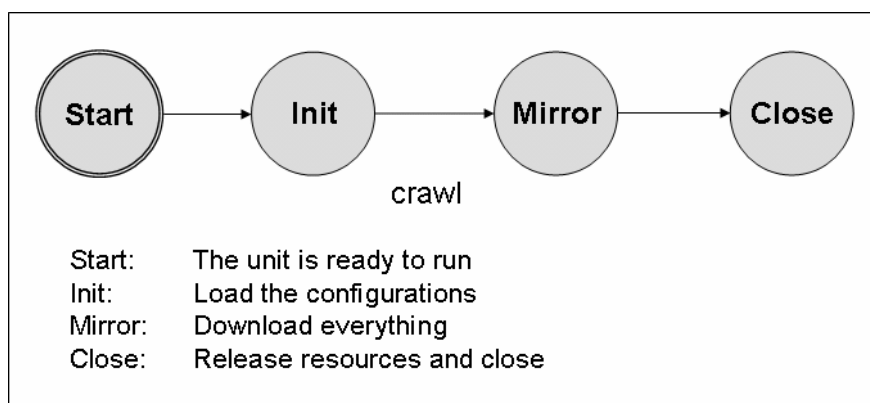


Figure 4-2 the state diagram of crawling

In the most spiders or crawlers, the process of mirror is crawling from the base URL. Our Smart Crawler is almost the same. First, Smart Crawler makes a list of URLs from the base URL and the multi-steps configurations and it loads all right configurations. Then, we come to the key state: mirror. Smart Crawler starts crawling data and save everything by the mirror algorithm. Besides, Smart Crawler also records the pair of each downloaded document, corresponding URL, and other information into the database. At last, Smart Crawler releases the resources and closes. The algorithm of mirror state is shown in Figure 4-3.

```

MIRROR(list)
{
    for each u with its d in list with order
        _MIRROR(u, d);
    end for
}
_MIRROR(u, d)
{
    if d is not -1 AND u is allowed
        d ← d-1;
        access the content c of u and save it in the local;
        If there is any new URL u' discovered in c
            add new pair (u', d) in the list;
    end if
    add the pair (c, u) in the MirrorList;
    end if
    return ;
}

```

Figure 4-3 the mirror algorithm

As we see, the list is the start list consisted from the base URL and Multi-Steps. For each URL u with its depth d with order, we pass the pair value to the `_MIRROR` function. In `_MIRROR` function; if d does not equal -1 , it means that u is not the leaf node; we decrease the value of d . At the same time, we check u is allowed or not. If the two conditions are both true, we do the next. We access the content from the remote side and save it in the local, called c . If there is any new URL u' discovered, i.e. subdirectory in FTP, add the pair (u', d) in the list. The URL u' would be visited later. After that, we add the pair (c, u) in the Mirrored list. The Mirrored list will be used for updating database records and generating reports to the

user or external systems. Hence, Smart Crawler would provide a function call called `getMirrored list` allow the interaction units access. The method would report what is downloaded and the details. Figure 4-4 shows the structure of the Mirrored list.

| Mirrored List | | | |
|---------------|---------|-----------|------------|
| State Flag | File ID | File Name | Remote URL |

Figure 4-4 the structure of Mirrored list

The state flag here is only “NEW” representing the new document. The file id is the unique id record in the database, just like the unique name we give. The file name is what it is and the remote URL is the file’s u in the algorithm.

4.1.3.Tracking

4.1.3.1.List



Before introducing tracking, we explain what is the method, list. Some network protocols provide a way to access the metadata of the remote data without download all first. For example, FTP is one of these protocols. When FTP client connects to the FTP server, the command “ls” would list the condition of files in the current directory, like name, size, modified date, owner, etc. The kind of network protocol is with list ability. Some other protocols, like HTTP, support the function optionally. When the web server supports the command “head”, the browser can read the metadata of the content without download it first. But we can’t ask each web server to support the function. So we treat HTTP as protocol without list ability. We call the protocol, like FTP, list support protocol and the others, like HTTP, no-list support protocol.

Figure 4-5 shows the algorithm of the list function. The list passed into the LIST function is the same with the list passed in MIRROR function. And the LIST function is very similar

with the MIRROR algorithm except it accesses the file's metadata instead of the content. In the other word, we just do a special mirror getting their metadata instead of their content.

```

LIST(list)
{
    for each u with its d in list with order
        _LIST(u, d);
    end for
}
_LIST(u, d)
{
    if d is not -1 AND u is allowed
        d ← d-1;
        access the metadata m and compare with the local record
        If m is changed
            add new pair (u', d) in the ListList;
        end if
    end if
    return ;
}

```

Figure 4-5 the list algorithm

The listed list is a set of URLs which metadata is not the same with the local record stored last time. Besides, the listed list also contains the URL's depth. The Figure 4-6 shows the structure of listed list.

| Listed List | |
|-------------|-------|
| Remote URL | Depth |

Figure 4-6 the structure of the listed list

Why do we need list? Generally speaking, the metadata of a file is smaller than its content by size. The other advantage is the metadata can stand for the status of the file on the remove side without downloaded it. Using list method, we can save a lot of bandwidth and other network resources. In another point; there are various files from data sources, the common way to compare one with another is using the information independent of their content. So, the metadata is the best choice. That is the other reason we need list.

4.1.3.2.Track

After we realize the benefit of list method, we introduce how to track data and update it in Smart Crawler. We introduce the tracking mechanism that is just like the crawling mechanism; we also propose a state sequence with the abstract behavior described in Chapter 3. Here are the states: (1) Smart Crawler begins to be ready (Start); (2) Load the configurations (Init); (3) if the protocol is with list ability, get the metadata on the remote side and compare the difference part as an update list. If not, the update list would be filled by the visited URLs visited last time (List). (4) Download and compare everything listed and update the old copy in the local side. Smart Crawler will update records in the database, too. (5) Release the resources (Close). Figure 4-7 is summarized what we discuss about.

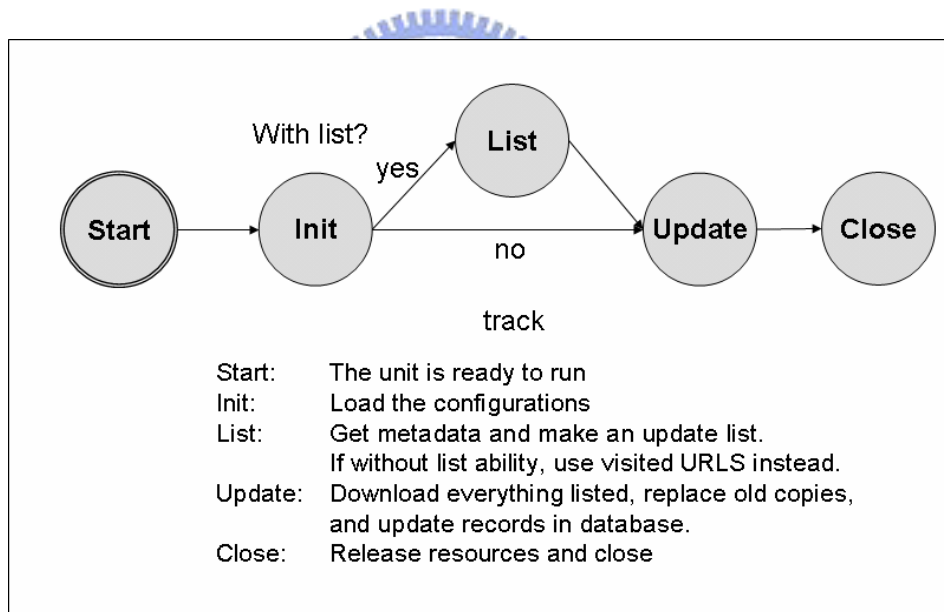


Figure 4-7 the state diagram of tracking

The key point is the protocol is with list ability or not. If the answer is positive, Smart Crawler could get metadata first and compare them with the record in the database by the same URLs. After comparison, Smart Crawler would make an update list that contains URLs which record in the database is difference the corresponding remote metadata. And then, Smart Crawler just downloads these listed documents, replace the local copies, and update the

listed records in the database. We will save some network bandwidth and computer resources.

Figure 4-8 shows the scenario.

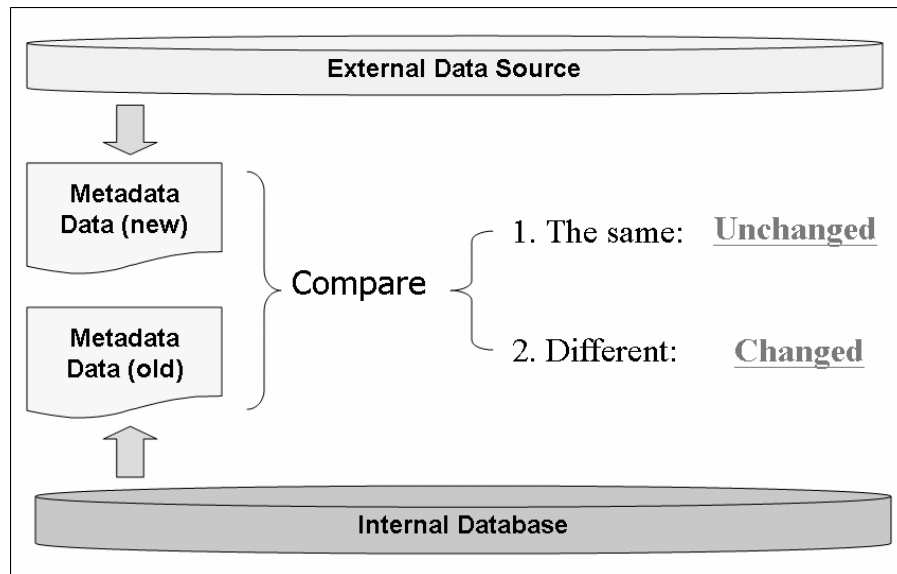


Figure 4-8 the tracking scenario for list support protocol

So we propose an algorithm called UPDATE for this condition in Figure 4-9. In the beginning, we get the listed list that contains the URLs confirmed for different content. For each u and its d , we pass them to the $_UPDATE$ function. It is similar with the MIRROR function, too. And then, we access the content of u . There are two conditions. One is we get the content successfully. We replace the old copy in the local and discover if there is a new URL u' . If there are new URLs discovered and these URLs match our rules, we have to download its content and create a new record for it. At the same time, we add the pair (c', u') into the UpdateList that reports the result of tracking and mark the pair "NEW" (new). It means the record is new created. We also add the pair (c, u) into the UpdateList and mark "UPD" (update). If the content c of u is null, it means the content of the URL is removed. So we add the pair (c, u) into the UpdateList and mark the pair "MIS" (missing). Otherwise, it means the copies are the same so we just mark it as "SAM" (same). We also provide a method called $getUpdateList$. That would return the UpdateList. The list contains the URLs, the state flag, such like "UPD", "MIS", or "NEW", and some other information. The structure is the same

with Mirrored list. We show it in Figure 4-10. The only difference with the Mirrored list is the state Flag. There are more flags used in the UpdateList.

```

UPDATE(ListList)
{
    for each u and its d in ListList with order
        _UPDATE(u, d);
    end for
}
_UPDATE(u, d)
{
    access the content c of u;
    If c is accessible and c has the difference size with the old copy
        replace the old copy in the local;
        for each new URL u' discovered in c if u' is allowed and d is not 0
            download its content c' and new a database record;
            add the pair (c', u') in the UpdateList and mark NEW;
        end for
        add the pair (c, u) in the UpdateList and mark UPD;
    else if c is null, it means it is removed from the remote side
        add the pair (c, u) in the UpdateList and mark MIS;
    else
        add the pair(c, u) in the UpdateList and mark SAM;
    end if
    return ;
}

```

Figure 4-9 the updating algorithm for list support protocol

| Updated List | | | |
|--------------|---------|-----------|------------|
| State Flag | File ID | File Name | Remote URL |
| | | | |

Figure 4-10 the structure of Updated List

If the protocol is without list ability, we have to download everything. So, Smart Crawler would walk again as the same as crawling. For each downloaded document, Smart Crawler can compare the new copy with the old one by the same URL and see if there is changed or not. After crawling and comparison, Smart Crawler can make a list recorded the updated files. So Smart Crawler just updates these listed records in the database. Finally, Smart Crawler releases the resources and close. The scenario is indicated in Figure 4-11.

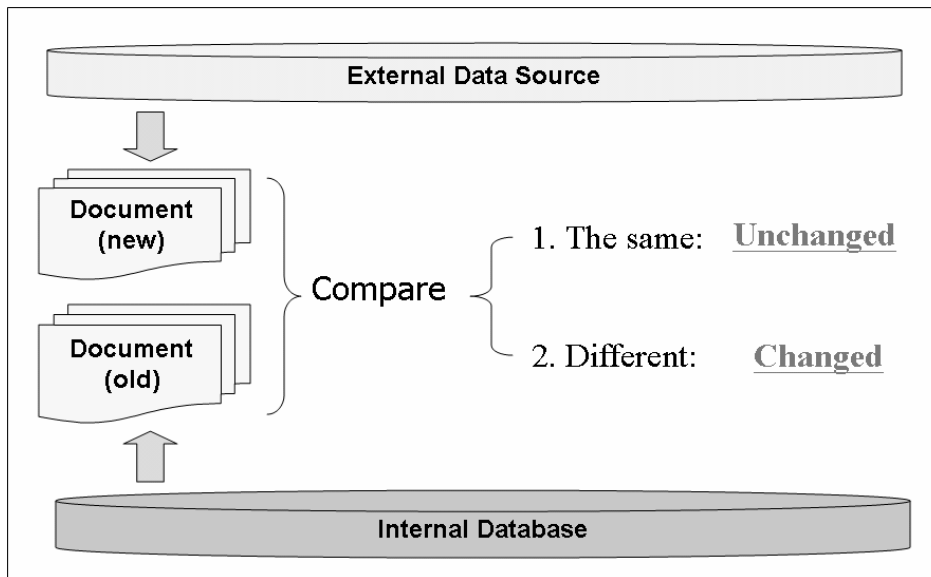


Figure 4-11 the tracking scenario of non-list support protocol

So the algorithm now is a bit of different with the one we talked. It looks like the merge of LIST function and UPDATE function with list support. It is shown in Figure 4-12. The list passed is consisted of the start list. For each u and its d in the list, we call the function `_UPDATE`. Because we have to walk again, we check the permission first just like LIST function. If allowed, we get its content c and compare with its local copy. There are three conditions here. One is we get c successfully and c has its local copy c' . If they are not the same, we replace the old one and mark it “UPD” (update). Or we mark it “SAM” (same). And we also discover new URLs in the updated content c . If we find new URLs, unlike the UPDATE function talked before, we use the recursive to call `_UPDATE` again. That would check the new URL u' and its corresponding depth d and recursively update. The second condition is we get c successfully but c has no local copy at all. It means the u is not crawled last time. So we create a new database record for it and mark it “NEW” (new). The third condition is we can't access the content c ; c is null. It means there is nothing on the remote side for the URL u . So we add the pair in the UpdateList and mark it as “MIS” (missing). After all, the update process is done.

```

UPDATE(list)
{
    for each u and its d in list with order
        _UPDATE(u, d);
    end for
}
_UPDATE(u, d)
{
    If d is not -1 and u is allowed
        d ← d-1;
        access the content c of u;
        If c is accessible and its local copy c' exists
            if c is different with c'
                replace the old copy in the local;
                add the pair (c, u) in the UpdateList and mark UPD;
                for each new URL u' discovered in c
                    _UPDATE(u', d);
                end for
            else
                add the pair (c, u) in the UpdateList and mark SAM;
            end if
        else if c is accessible but c has no local copy
            new a new record;
            add the pair (c, u) in the UpdateList and mark NEW;
        else if c is null, it means it is removed from the remote side
            add the pair (c, u) in the UpdateList and mark MIS;
        end if
    end if
    return ;
}

```

Figure 4-12 the updating algorithm of non-list support protocol

Whether the network protocol is with list support or not, we propose both algorithms for them. By these algorithms, Smart Crawler can easily find the change data and update the cache in the local. The most important key is we can find the new content and its old copy by its unique URL. If we can't find the relation between these documents according their remote URL, these algorithms can't work rightly. The exception is discussed in Section 4.2.

4.2. Tracking in HTTP Session with URL Rewriting

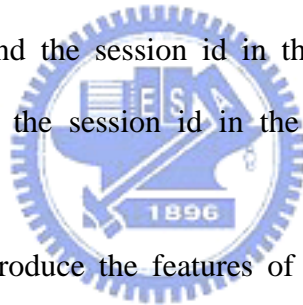
The HTTP session with URL rewriting is an exception in our tracking algorithm talked in Section 4.1. As we know, when we travel a web site which is based on HTTP session implemented by URL rewriting, we will get a session id embedded in the URLs in this time connection. When we travel the same site another day another time, we will get another session id. The different session ids would make the same document in the same web site has

more than one URL. If we use the algorithm mentioned before, Smart Crawler could not find the same URL and determine the pair of the document and URL. So for the condition, we propose a pre-process to solve the problem. After pre-processing, Smart Crawler can find the corresponding URL before and after.

4.2.1.Finding Session ID

4.2.1.1.Preface

Before we introduce the section, why do we find the session id? In our tracking algorithm for URL rewriting discussed in next section, the main idea is to replace the old session id to the new one and then to re-travel the traveled URLs for comparing the differences. Hence, how to find the session id in the HTTP connection is important and necessary. Even, how to find the session id in the history URLs is also important and necessary.



So in the section, we introduce the features of the session id. According the feature discussed later, we illustrate the conclusion: the session id is one of the fixed-length i common substrings between URLs. Hence, we propose several algorithms to find this kind of substrings. According that, we design an algorithm that can find the session id in the HTTP connection.

4.2.1.2.Features of the Session ID

There are some features about the session id used in HTTP. The basic condition is the session id is composed from the normal characters. The opposite of the normal characters is the special characters. The members of the special characters are ‘;’, ‘/’, ‘&’, ‘%’, ‘?’, and ‘=’. In brief, the session id is consisted from everything except the special characters. Because these characters have their special means in the URL. There is not important constraint in the

id except this. We can say that the next character followed with the session id must be a special character or end of string (EOS).

Although there are not many constraint in the id, most implementation the session id is usually a hashed value and with a fixed length. The function of session id is to identify clients and data stored in the server without using any client resources, like cookie. But some hacker might try to guess the using session id so that he could access the client's personal information; replace the client, or something evil. So for avoiding the security problem, the implementation of session id usually uses encryption to be guessed easily or be tried out quickly but without too much complexity. The hash table is a common way in the real world, and some features are bringing out. One of the useful features is the session id in the same web site is a fixed length string.

The session id also has another feature by nature. The links that is used for the HTTP session with URL rewriting should be encoded for embedding the session id into the URL. It means the session id must be a substring of the URL.

Hence, we propose the session id has the two features:

1. It has the fixed length and its content won't change in the same session.
2. The id is embedded in the URL.
3. The next character followed with the session id must be a special character or EOS.

So, we can say that the session id is one of the common substrings between all URLs. We propose an algorithm to find the fix-length common substring in two known strings later.

4.2.1.3. Find the common fixed-length i string

Before we introduce the algorithm to find the common fixed-length i string, we use a graph to explain what is the common substrings and their length. We show that in Figure 4-13.

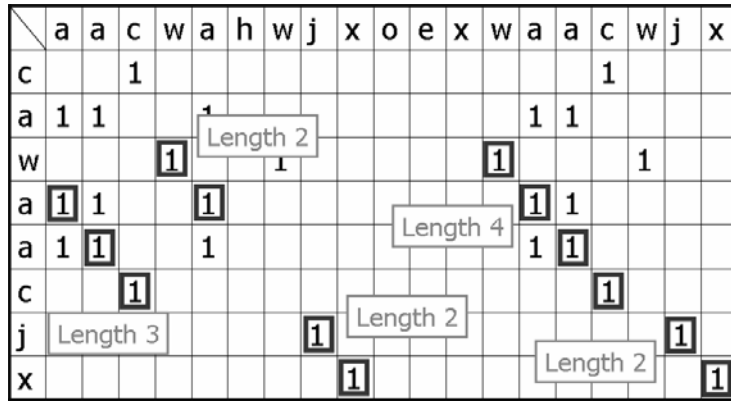


Figure 4-13 the common substrings

The figure is usually used to introduce the longest common substring (LCS) [23]. But our focus is on all common substrings. In Figure 4-13, we compare the two strings: “aacwahwjxoexwaacwjx” and “cawaacjx”. We use an array to record the comparing result of each character one by one. The value 1 means the two compared characters is the same, otherwise the value is empty. In the array, the common string is the sequence that value of each character is 1 and the height is decreased. In the other word, the slope of the sequence in the array map is -1. The length of the common string is the sequence length. So, in the Figure, we can see one length 4 common substring, one length 3 common substring, and three 2 common substrings. The substring with length 4 is the longest common substring.

We summarize the useful features about the session id as follows: the session id is a fixed length substring embedded in dynamical URLs in the html document. According the feature, we propose an algorithm to find the common fix length l string between two strings, called. We can use the algorithm to find the session id between URLs.

For the way to find the common fixed-length l substring between two strings a and b ; we compare three methods. The first is finding the all common strings between two strings, and then choose the substring with length l . The second way is make a substring with length l of one string, and then use Knuth-Morris-Pratt Algorithm (KMP) algorithm [20][25][26] to do string pattern match [23] process seeing if the substring is matched or not. It positive, the

substring is the candidate we want, otherwise not. The final way is using Boyer-Moore (BM) algorithm [21][24] instead of KMP.

```

FIND_CANDIDATE_ALLCS(a, b, l)
{
    C is the candidate set;
    S = {'/', '&', '%', '=', '#'};
    a is a n length string;
    b is a k length string;
    l is the length of the session id;
    i ← 0; j ← 0; m ← 0; e ← -1; p ← -1;
    for m ← 1 to n
        if a[m] is not in S
            for i ← m to n
                if a[i] == b[j] and b[j] is not in S
                    if p == -1
                        p ← j;
                    end if
                    e ← j;
                    j ← j+1;
                else
                    if p != -1
                        if the length of the substring from p to e, called s, is l
                            C = C ∪ {s};
                        end if
                        j ← p+1; p ← -1; e ← -1; i ← m-1;
                    else
                        j ← j+1; i ← m-1; p ← -1; e ← -1;
                    end if
                end if
            end for
        end if
        if p != -1 and the length of the substring from p to e, called s is l
            C = C ∪ {s};
            j ← 0; p ← -1; e ← -1;
        end if
    end for
    return C;
}

```

Figure 4-14 find all common substring with the common way

The first algorithm is very simple and intuitively. For every character a[m] in a sequentially do comparison with each character in b. If a[m] is a special character, we get the next one and restart the process. If it is not, for every character from a[a[m] to the end character of a marks a[i] following compares with every character from head to tail in b, marks in b[j]. If a[i] and b[j] is the same and b[j] is not the special character, do the comparison for next a[i] and next b[j] and use a pivot p to record the start point until they are not the same or b[j] is the special character and use another pivot e to record end point. From the b[p] to b[e], the

substring is what we want the common substring between a, and b. If the length is l, put it into the candidate set C. If the comparison is in the condition that reaching the end of string a or string b and the pivot is marked, it means the common substring is the chars at the end of one of these string. If the length of the substring is l, put it into C or use another “a[m]” again. If the “a[m]” is the last character of string a, the comparison is going to the end and we have found all of the substring between the two string in the candidate set C. The pseudo code is shown in the Figure 4-14.

```

FIND_CANDIDATE_KMP(a, b, l)
{
    C is the candidate set;
    S = {'/', '&', '%', '=', '#'};
    a is a n length string;
    b is a k length string;
    l is the length of the session id;
    for b[m] i ← 1 to (k-l+1)
        ss is the a substring with length l from b[m];
        if ss doesn't contain the char in S, b[m+l] is in S or end of string,
        and ss is found in a with KMP algorithm
            C = C ∪ {ss};
    end for
    return C;
}

```

Figure 4-15 the algorithms using KMP for common substring

The second algorithm and the third one use the feature of pattern match. The main idea is the session id must be a substring in both strings, so we can find the session id exactly is matched in both strings. So we choose the famous pattern match algorithm to help us.

The second algorithm uses KMP algorithm. We show the pseudo code in Figure 4-15. From the string b, we get the l length substring ss as the pattern. The pattern string must compose of all but the special chars, and the next character followed must be one of the special chars. If the next character is not in the special character set, the pattern is not an ended string for URL. In other word, the pattern is not the session id. And then, we use KMP algorithm to find if the pattern was matched or not. If matched, the pattern is added to the

candidate set C. If not, do the process for next pattern. The third one replace KMP algorithm instead of BM algorithm. We show the pseudo code in Figure 4-16.

```

FIND_CANDIDATE_BM(a, b, l)
{
    C is the candidate set;
    S = {'/', '&', '%', '=', '#'};
    a is a n length string;
    b is a k length string;
    l is the length of the session id;
    for b[m] i = 1 to (k-l+1)
        ss is the a substring with length l from b[m];
        if ss doesn't contain the char in S, b[m+l] is in S or end of string,
            and ss is found in a with Boyer-Moore algorithm
            C = C ∪ {ss};
    end for
    return C;
}

```

Figure 4-16 the algorithms using BM for common substrings

After testing the performance of these three algorithms, we get the best result of the algorithm 3: using Boyer-Moore pattern match finding the common and fix length l string. It is shown in Figure 4-17. The evaluation will be discussed in Chapter 6.

```

GET_THE_L_LENGTH_COMMON_STRING(a, b, l)
{
    return FIND_CANDIDATE_BM(a, b, l);
}

```

Figure 4-17 we use BM for our L length common string finding algorithm

4.2.1.4. Find the Session ID

We discuss about the session id more. A HTML Web page usually has not only static links that is just point to a location with no parameters but also dynamic links that is consisted by web location, a lot of parameters, and the session id. So the average length of static links is usually shorter than the average length of dynamic links. If we find all of the common substrings as candidates between two dynamic links with the same length as the session id,

there must be one string which is the real session id.

So we choose the longest links in the page first and find all of the common substring with a given length e , called session id candidates, and then each candidate c has a corresponding integer d that record how many other links has the same part in them. If there is the same part as the candidate c in one link, the d increases by one. After all, there is only one candidate which has the most value of d and it is what we want, the session id. Pseudo code is shown in Figure 4-18. If the candidate set is empty in the beginning, it means there is more than one static links in the two longest links we choose. We can drop the longest one and find two the longest links in the rest links until we get the candidates.

```
GET_SESSION_ID(A)
{
    the session id length is e;
    i, j is the two longest URL in A and remove them from A;
    C = GET_THE_L_LENGTH_COMMON_STRING(i, j, e);
    while C is empty
        let i, j be the new two longest URL in A and remove them from A;
        C = GET_THE_L_LENGTH_COMMON_STRING(i, j, e);
    end while
    for each c in C which has a corresponding integer d in D
        for each m in A
            if c is a part of m
                d ← d+1;
            end if
        end for
    end for
    return the c in C corresponding with d in D where d is the most of all;
}
```

Figure 4-18 the algorithm of get session id

According the algorithm, we can easily find the session id between URLs. But the mechanism has some limitations. If the page of the base URL has fewer dynamic links, the find session id might make a mistake on determining the session id. If the static links in the page are more and longer than the dynamic ones, it also might make a mistake, too.

We do some evaluations in Chapter 6. More detail is introduced in Chapter 6.

4.2.2. Tracking Algorithm

Now, we introduce the track algorithm to do update when the Web pages is designed in URL rewriting. The main idea of the algorithm is replacing the old session id with the new session id. When we travel the same page with different time, we get a different session id. That we have already told. In the other word, we get the same thing in every time except the session id. So we can visit the link last visited just replace the session id to this time. Through the benefit of the find session id algorithm, we can calculate the old id from the past records and calculate the new id from the links on the page visited this time. The detail is followed.

Here are the terms we used later:

- A: The set of the visited links last time.
- T: The set of the visiting links this time.
- F: The set of the non-updated links this time.
- U: The set of the updated links this time.
- E: The set of the error links this time.
- b : The base URL this time, the first task of the project.
- s : The session id last time.

Before we do the update, we get the base URL b and the other URLs from the database. According b , we can download its document. In this document, there are many URLs and these URLs are mixed with static links and dynamic links. In the static links, there is not session id in it because they are not rewritten by the web server. But the dynamic links are rewritten by the server, so they do have the session id. In these dynamic links, there are many common strings between them. The session id must be in them. So we can find session id from these URLs, then replace the old session id to the new one for every link pinked up from the database. After that, we add these updated URLs into the job pool, which contains links that will be visited this time. Finally, download all of them and we can easily find the old one

by its URL to compare with. If the old copy is different with new one, not only store the new one but record it in the Update URL set U. If the document is null, it means we don't download normally. Just save it in the Error URL set E. The others URLs which are neither in the U nor E, it will be put into the non-updated Finished URL set F. The URLs in the set tell us the contents in the remote web server are the same with the local copy. So we call the set A stand for the URLs in the database. There is a relation between A, U, F, E: $A \cup \{b\} = F + U + E$; The user can know what is updated, what is error, and what is unchanged. The pseudo code is shown in Figure 4-19.



```

UPDATE(b, A)
{
    T = {};
    F = {};
    E = {};
    d <- GET_DOCUMENT(b);
    if d is not null
        if d is not changed
            F = {b};
        else
            U = {b};
            update content of b by d in the database
        end if
    T = GET_ALL_LINKS(d);
    s' <- GET_SESSION_ID(T);
    s <- GET_SESSION_ID(A);
    for each link m in A
        replace the string s in m with s';
    end for
    T = T ∪ A';
    for each link n in T
        d <- GET_DOCUMENT(n);
        if d is not null
            if d is not changed
                F = F ∪ {n};
            else
                U = U ∪ {n};
                update content of n by d in the database
            end if
            T = T ∪ GET_ALL_LINKS(d);
        else
            E = E ∪ {n};
        end if
    end for
    else
        E = {b};
    end if
    return F, U and E;
}

```

Figure 4-19 the pseudo code of tracking for URL rewriting

How do we download a document? First we send the HTTP request for the URL we want, and we will get the server's response. In the first line of response, we can get the http code. If the code is greater than 400, there is something wrong with this URL. For example, 404 is File Not Found. If the code is almost 300 or so, such as 301, 302, it tells us we have to do a relocation action. In the header after the first line, we can get the new location and send a new

HTTP request for it. If the code is 200 or so, we get the document content followed the header.

Figure 4-20 shows that.

```
GET_DOCUMENT(n)
{
    if these link n is not allowed by the user defined rules
        return null;
    end if
    Send HTTP request for link n and get the return data with return code r, header h and content c;
    if r is 2xx
        return c;
    end if
    if r is 4xx or r is 5xx
        return null;
    end if
    if r is 3xx
        Get redirection location l from h;
        return GET_DOCUMENT(l);
    end if
}
```

Figure 4-20 the pseudo code of GET_DOCUMENT

Just link other spiders, we parse the document and record the links in the tag a and img. In the tag a, the attribute href has the web link. In the tag img, the attribute src has the image source. That is summarized in Figure 4-21.

```
GET_ALL_LINKS(d)
{
    Let A is the set of the links in the same host from HTML tag/attribute <a>/href, <img>/src;
    return A;
}
```

Figure 4-21 the pseudo code of GET_ALL_LINKS

Hence, according the algorithm, Smart Crawler will find the corresponding URLs of crawling with different times, and track the data on the remote server.

4.2.3.Example

We take an example to explain how the algorithm works.

The base URL b is <http://www.amazon.com/>. We have known the session id is 19 in this site. The Links in the document of the BASE URL, called A.

1. http://www.amazon.com/exec/obidos/subst/home/home.html/ref=three_tab_gw/002-9355727-0611208
8
2. [http://www.amazon.com/exec/obidos/tg/browse/-/229220/ref=gw_subnav_gft/002-9355727-0611208?
%5Fencoding=UTF8](http://www.amazon.com/exec/obidos/tg/browse/-/229220/ref=gw_subnav_gft/002-9355727-0611208?%5Fencoding=UTF8)
3. [http://www.amazon.com/exec/obidos/tg/stores/static/-/gateway/international-gateway/ref=gw_subnav_in/002-9355727-0611208?
%5Fencoding=UTF8](http://www.amazon.com/exec/obidos/tg/stores/static/-/gateway/international-gateway/ref=gw_subnav_in/002-9355727-0611208?%5Fencoding=UTF8)
4. [http://www.amazon.com/exec/obidos/tg/new-for-you/new-releases/-/main/ref=gw_subnav_nr/002-
9355727-0611208?%5Fencoding=UTF8](http://www.amazon.com/exec/obidos/tg/new-for-you/new-releases/-/main/ref=gw_subnav_nr/002-9355727-0611208?%5Fencoding=UTF8)
5. [http://www.amazon.com/exec/obidos/tg/new-for-you/top-sellers/-/main/ref=gw_subnav_ts/002-9
355727-0611208?%5Fencoding=UTF8](http://www.amazon.com/exec/obidos/tg/new-for-you/top-sellers/-/main/ref=gw_subnav_ts/002-9355727-0611208?%5Fencoding=UTF8)
6. [http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_el_feat%2F002
-9355727-0611208%3F%255Fencoding%3DUTF8%26node%3D172282&pf_rd_p=163187901&pf_rd_s=left-nav
&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ](http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_el_feat%2F002-9355727-0611208%3F%255Fencoding%3DUTF8%26node%3D172282&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ)
7. [http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_gf%2F002-9355
727-0611208%3F%255Fencoding%3DUTF8%26node%3D3370831&pf_rd_p=163187901&pf_rd_s=left-nav&pf_r
d_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ](http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_gf%2F002-9355727-0611208%3F%255Fencoding%3DUTF8%26node%3D3370831&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ)
8. [http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_jw%2F002-9355
727-0611208%3F%255Fencoding%3DUTF8%26node%3D3367581&pf_rd_p=163187901&pf_rd_s=left-nav&pf_r
d_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ](http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_jw%2F002-9355727-0611208%3F%255Fencoding%3DUTF8%26node%3D3367581&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ)
9. [http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirec
t.html%2Fref%3Dgf_gw_wine%2F002-9355727-0611208%3Flocation%3Dhttp%3A%2F%2Fwww.wine.com%2Fp
romos%2Famazonwine.asp%253Fan%253Damazon%2526s%253Damazon%2526cid%253Damazon%255Fgateway%2
55Ffpbox%26token%3D21513C47B07C95040C8597937CAB64718E97425C&pf_rd_p=163187901&pf_rd_s=left
-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ](http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Dgf_gw_wine%2F002-9355727-0611208%3Flocation%3Dhttp%3A%2F%2Fwww.wine.com%2Fpromos%2Famazonwine.asp%253Fan%253Damazon%2526s%253Damazon%2526cid%253Damazon%255Fgateway%255Ffpbox%26token%3D21513C47B07C95040C8597937CAB64718E97425C&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ)
10. [http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirec
t.html%2Fref%3Damb_link_649402_2%2F002-9355727-0611208%3Flocation%3Dhttp%3A%2F%2Fwww.amazo
n.com%2Fgp%2Fsearch.html%2F%253Fplatform%253Dgurupa%2526url%253Dnode%253D11055981%2526keyw
ords%253DT%26token%3DD2E7DF9A889DD105A02CC33159540A6D52DE4D8B&pf_rd_p=160346101&pf_rd_s=r
ight-4&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ](http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Damb_link_649402_2%2F002-9355727-0611208%3Flocation%3Dhttp%3A%2F%2Fwww.amazon.com%2Fgp%2Fsearch.html%2F%253Fplatform%253Dgurupa%2526url%253Dnode%253D11055981%2526keywords%253DT%26token%3DD2E7DF9A889DD105A02CC33159540A6D52DE4D8B&pf_rd_p=160346101&pf_rd_s=right-4&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ)
11. [http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding
=UTF8&path=http%3A%2F%2Fwww.amazon.co.uk%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usst
oreh-ukhome-21%26site%3Damazon](http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.uk%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-ukhome-21%26site%3Damazon)
12. [http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding
=UTF8&path=http%3A%2F%2Fwww.amazon.de%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstore
h-dehome-21%26site%3Dhome](http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.de%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-dehome-21%26site%3Dhome)
13. [http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding
=UTF8&path=http%3A%2F%2Fwww.amazon.co.jp%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usst
oreh-jphome-22%26site%3Damazon](http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.jp%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-jphome-22%26site%3Damazon)
14. [http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding](http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.com%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-ushome-23%26site%3Damazon)

- =UTF8&path=http%3A%2F%2Fwww.amazon.fr%2Fexec%2Fobidos%2Fredirect-home%3Fsite%3Damazon%26tag%3Dusfr-storeh-footer-21
15. http://ec1.images-amazon.com/images/G/01/marketing/visa/amzn_visa-save-30-today.gif
 16. http://ec1.images-amazon.com/images/G/01/gateway/partner-logos/target_logo._V58636053_.gif
 17. http://images.amazon.com/images/P/6305428050.01._PE30_SCTZZZZZZZ_.jpg

First, we add the BASE URL into the working pool. And then we get one link, obviously it is the BASE URL, from the pool and remove it for downloading its document.

After downloading the document of the BASE URL this time, we get the links following, called B:

1. http://www.amazon.com/exec/obidos/subst/home/home.html/ref=three_tab_gw/104-5791018-2027935
2. http://www.amazon.com/exec/obidos/tg/browse/-/229220/ref=gw_subnav_gft/104-5791018-2027935?%5Fencoding=UTF8
3. http://www.amazon.com/exec/obidos/tg/stores/static/-/gateway/international-gateway/ref=gw_subnav_in/104-5791018-2027935?%5Fencoding=UTF8
4. http://www.amazon.com/exec/obidos/tg/new-for-you/new-releases/-/main/ref=gw_subnav_nr/104-5791018-2027935?%5Fencoding=UTF8
5. http://www.amazon.com/exec/obidos/tg/new-for-you/top-sellers/-/main/ref=gw_subnav_ts/104-5791018-2027935?%5Fencoding=UTF8
6. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_el_feat%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D172282&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN
7. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_gf%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D3370831&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN
8. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_jw%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D3367581&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN
9. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Dgf_gw_wine%2F104-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.wine.com%2Fpromos%2Famazonwine.asp%253Fan%253Damazon%2526s%253Damazon%2526cid%253Damazon%255Fgateway%255Ffpbox%26token%3D21513C47B07C95040C8597937CAB64718E97425C&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN
10. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect

- t.html%2Fref%3Damb_link_29290301_30%2F104-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.amazon.com%2Fgp%2Fsearch.html%2F%253Fnode%253D3888811%2526keywords%253DDesigner%2526index%253Djewelry%2526rank%253D-price%2526x%253D7%2526y%253D14%26token%3D20975AF20B5B70DB8D6C01314A59A3FFC56364D7&pf_rd_p=162964201&pf_rd_s=center-8&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN
11. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.uk%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-ukhome-21%26site%3Damazon>
 12. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.de%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-dehome-21%26site%3Dhome>
 13. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.jp%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-jphome-22%26site%3Damazon>
 14. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.fr%2Fexec%2Fobidos%2Fredirect-home%3Fsite%3Damazon%26tag%3Dusfr-storeh-footer-21>
 15. <http://ecl.images-amazon.com/images/G/01/x-locale/common/transparent-pixel.gif>
 16. http://ecl.images-amazon.com/images/G/01/gateway/partner-logos/target_logo._V58636053_.gif



For A, the link 9, 10 are the top 2 of the longest string. So we use them to find the session id last time. Just find the common string with length 19 between them. And add 1 if the candidate is the substring of A, else 0. So we can get the result in Table 4-1.

| Length | String | Count |
|--------|---------------------|-------|
| 19 | 002-9355727-0611208 | 14 |
| 19 | PS2V6KKYBZ34F3RK1PJ | 5 |

Table 4-1 the result of finding old session id candidates in our example

So the string "002-9355727-0611208" is the last time session id. For the links this time, B, we do the same thing: The link 9, 10 is the longest string and then find the common string with length 19 between them. After that, do a counting process. We can get the result showing in Table 4-2.

| Length | String | Count |
|--------|---------------------|-------|
| 19 | 104-5791018-2027935 | 14 |
| 19 | 5BNH99VVKBB0CDBQGQN | 5 |

Table 4-2 the result of finding new session id candidates in our example

So the string "104-5791018-2027935" is the session id this time. Replace the session id in A with one in B. Here is the result, called A':

1. http://www.amazon.com/exec/obidos/subst/home/home.html/ref=three_tab_gw/104-5791018-2027935
2. http://www.amazon.com/exec/obidos/tg/browse/-/229220/ref=gw_subnav_gft/104-5791018-2027935?%5Fencoding=UTF8
3. http://www.amazon.com/exec/obidos/tg/stores/static/-/gateway/international-gateway/ref=gw_subnav_in/104-5791018-2027935?%5Fencoding=UTF8
4. http://www.amazon.com/exec/obidos/tg/new-for-you/new-releases/-/main/ref=gw_subnav_nr/104-5791018-2027935?%5Fencoding=UTF8
5. http://www.amazon.com/exec/obidos/tg/new-for-you/top-sellers/-/main/ref=gw_subnav_ts/104-5791018-2027935?%5Fencoding=UTF8
6. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_el_feat%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D172282&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
7. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_gf%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D3370831&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
8. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_jw%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D3367581&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
9. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Dgf_gw_wine%2F104-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.wine.com%2Fpromos%2Famazonwine.asp%253Fan%253Damazon%2526s%253Damazon%2526cid%253Damazon%255Fgateway%255Ffbox%26token%3D21513C47B07C95040C8597937CAB64718E97425C&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
10. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Damb_link_649402_2%2F104-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.amazon.com%2Fgp%2Fsearch.html%2F%253Fplatform%253Dgurupa%2526url%253Dnode%253D11055981%2526keyw

- ords%253DT3%26token%3DD2E7DF9A889DD105A02CC33159540A6D52DE4D8B&pf_rd_p=160346101&pf_rd_s=r
ight-4&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
11. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.uk%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-ukhome-21%26site%3Damazon>
 12. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.de%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-dehome-21%26site%3Dhome>
 13. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.jp%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-jphome-22%26site%3Damazon>
 14. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.fr%2Fexec%2Fobidos%2Fredirect-home%3Fsite%3Damazon%26tag%3Dusfr-storeh-footer-21>
 15. http://ec1.images-amazon.com/images/G/01/marketing/visa/amzn_visa-save-30-today.gif
 16. http://ec1.images-amazon.com/images/G/01/gateway/partner-logos/target_logo._V58636053_.gif
 17. http://images.amazon.com/images/P/6305428050.01._PE30_SCTZZZZZZZ_.jpg



So we can add A' into the working pool for updating the document downloaded last time. Just download it again and compare with the local copy.

And then we add B into the working pool. But there are some duplicated links and some new links between A' and B. The duplicated links are the link 1, 2, 3, 4, 5, 11, 12, 13, 14, and 16. The new links in B is 6, 7, 8, 9, 10, and 15. The missing links that are in A but not in B are 6, 7, 8, 9, 10, 15, and 17. So we just add the new links in B into the working pool.

Here is the complete link list in the working pool this time, called C:

1. http://www.amazon.com/exec/obidos/subst/home/home.html/ref=three_tab_gw/104-5791018-2027935
2. http://www.amazon.com/exec/obidos/tg/browse/-/229220/ref=gw_subnav_gft/104-5791018-2027935?%5Fencoding=UTF8
3. http://www.amazon.com/exec/obidos/tg/stores/static/-/gateway/international-gateway/ref=gw_subnav_in/104-5791018-2027935?%5Fencoding=UTF8
4. http://www.amazon.com/exec/obidos/tg/new-for-you/new-releases/-/main/ref=gw_subnav_nr/104-5791018-2027935?%5Fencoding=UTF8

5. http://www.amazon.com/exec/obidos/tg/new-for-you/top-sellers/-/main/ref=gw_subnav_ts/104-5791018-2027935?%5Fencoding=UTF8
6. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_el_feat%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D172282&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
7. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_gf%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D3370831&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
8. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_jw%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D3367581&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
9. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Dgf_gw_wine%2F104-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.wine.com%2Fpromos%2Famazonwine.asp%253Fan%253Damazon%2526s%253Damazon%2526cid%253Damazon%255Fgateway%255Ffbbox%26token%3D21513C47B07C95040C8597937CAB64718E97425C&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
10. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Damb_link_649402_2%2F104-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.amazon.com%2Fgp%2Fsearch.html%2F%253Fplatform%253Dgurupa%2526url%253Dnode%253D11055981%2526keywords%253DT3%26token%3DD2E7DF9A889DD105A02CC33159540A6D52DE4D8B&pf_rd_p=160346101&pf_rd_s=right-4&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
11. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.uk%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-ukhome-21%26site%3Damazon>
12. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.de%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-dehome-21%26site%3Dhome>
13. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.jp%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-jphome-22%26site%3Damazon>
14. <http://www.amazon.com/exec/obidos/redirect-to-external-url/104-5791018-2027935?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.fr%2Fexec%2Fobidos%2Fredirect-home%3Fsite%3Damazon%26tag%3Dusfr-storeh-footer-21>
15. http://ecl.images-amazon.com/images/G/01/marketing/visa/amzn_visa-save-30-today.gif
16. http://ecl.images-amazon.com/images/G/01/gateway/partner-logos/target_logo._V58636053_.gif
17. http://images.amazon.com/images/P/6305428050.01._PE30_SCTZZZZZZZ_.jpg
18. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_el_feat%2F104-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D172282&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQQGN

19. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_gf%2F014-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D3370831&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN
20. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_jw%2F014-5791018-2027935%3F%255Fencoding%3DUTF8%26node%3D3367581&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN
21. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Dgf_gw_wine%2F014-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.wine.com%2Fpromos%2Famazonwine.asp%253Fan%253Damazon%2526s%253Damazon%2526cid%253Damazon%255Fgateway%255Ffpbox%26token%3D21513C47B07C95040C8597937CAB64718E97425C&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN
22. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Damb_link_29290301_30%2F014-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.amazon.com%2Fgp%2Fsearch.html%2F%253Fnode%253D3888811%2526keywords%253DDesigner%2526index%253Djewelry%2526rank%253D-price%2526x%253D7%2526y%253D14%26token%3D20975AF20B5B70DB8D6C01314A59A3FFC56364D7&pf_rd_p=162964201&pf_rd_s=center-8&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN
23. <http://ec1.images-amazon.com/images/G/01/x-locale/common/transparent-pixel.gif>

For the links from 1 to 17, we have a copy in the local and just differ with them. For the others links, we have nothing so that we have to create new records for them. Finally, we do the same thing to each link in C until the working pool is empty. After we finish the update process, we have to update the records in the database, too. Unlike the tracking algorithm before, we replace the new session id with the old one in all links and then store them for their corresponding files. This is because we want to keep the remote URLs in the database with a uniform id. At first we query the visited links last time from the database, they all have the old id; at last we update the visited links this time to the database, they all still have the old id. It would be more convenient for next time we do the tracking.

Thought the example, we explain how the algorithm works and we can see that we finished the update process correctly. By the tracking mechanism, we believe we can handle the situation on HTTP session with URL rewriting.

Chapter 5 Implementation

After discussing architecture and design issues, we will go into details to implementation. This chapter would introduce the detail of implementation with our Smart Crawler. First, we introduce the running environment, and then we introduce each unit in Smart Crawler one by one.

5.1. Environment

We have introduced already that there are three main units in Smart Crawler: network crawling units, user interaction units, and content publisher units. And there is still one internal database we use in Smart Crawler. Figure 5-1 introduces a normal scenario of Smart Crawler running environment.

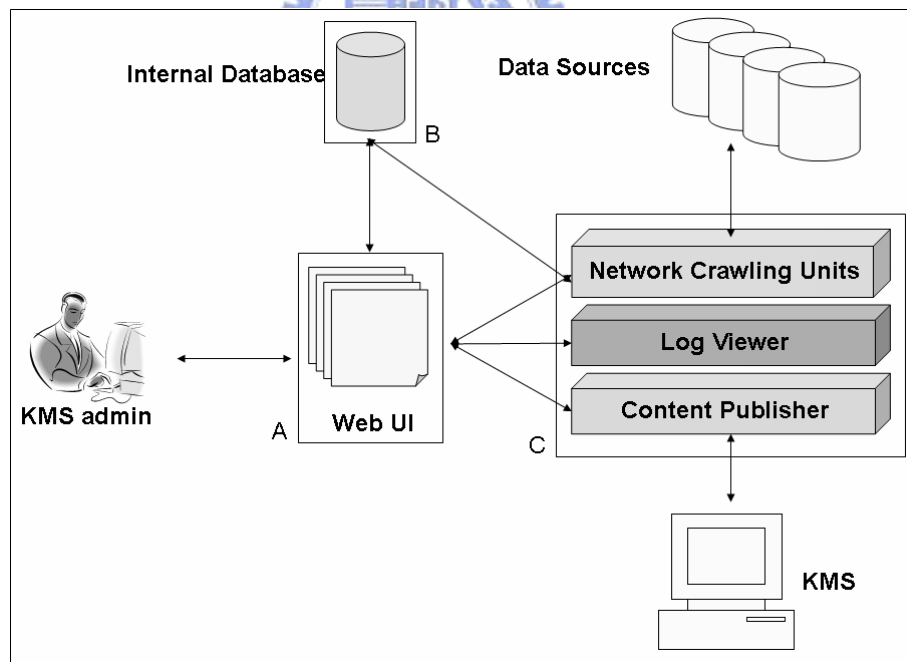


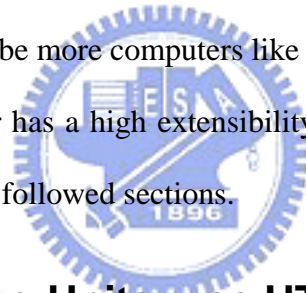
Figure 5-1 the scenario of Smart Crawler environment

The scenario describes KMS admin who can input or change the settings into the internal database through the Web UI. He also uses the Web UI to run the crawling project, and view the result log. When the network crawling units are got the signal from the Web UI, they can

get the project information from the internal database and do what they have to do. KMS can access the crawled contents through the content publisher. KMS admin can access the crawled data through the Web UI which would connect to the content publisher units.

From the scenario, we understand the big picture of Smart Crawler. For the advantages of the architecture, we can put these units in the different machines. In the figure, the Web UI is placed in the computer A which runs a web server. In B computer, the internal database is running there. In C computer, we put the core units, such like the network crawling units, the log viewer, and the content publisher. Smart Crawler runs in different machines so that the computation loading is shared by these computers. The resources needed in all units are separated in several independent computers, so the competition won't happen. In the other view; if there is some unit crash, the other unit might still finish their works. Beside, as we discuss in Chapter 3, there can be more computers like C computer which runs the core units.

We believe Smart Crawler has a high extensibility and flexibility. We will introduce the implementation of each unit as followed sections.



5.2. Network Crawling Units: use HTTP and FTP

Figure 5-2 is the architecture of the network crawling unit. There are some parts shown in Chapter 3. The `FtpObservable`, `HttpObservable` is designed from the common crawling interface. The structure of the common crawling interface is introduced in Chapter 3. So we introduce the other components first. `FtpObservable` uses the `Ftp` class design by `JvFTP` to negotiate with the remote FTP servers. In the other word, `JvFTP` is the crawler originally and we repack it and integrate into Smart Crawler. In the `HttpObservable`, we choose the spider `JoBo` as our original crawler. The `WebRobot` class provides many functions to negotiate with the web server and a lot of rules to filter what should be downloaded and what is not. By repacking followed by the common crawling interface, we integrate it, too. The `DatabaseObservable` provides the abilities to connect to the internal database. In the database

which will be introduced later, there are many tables used. So the DatabaseObservable integrates the classes which handle the communication with tables, like ProjectTable, SorceTable, etc. The three components is controlled by the ControlObserver.

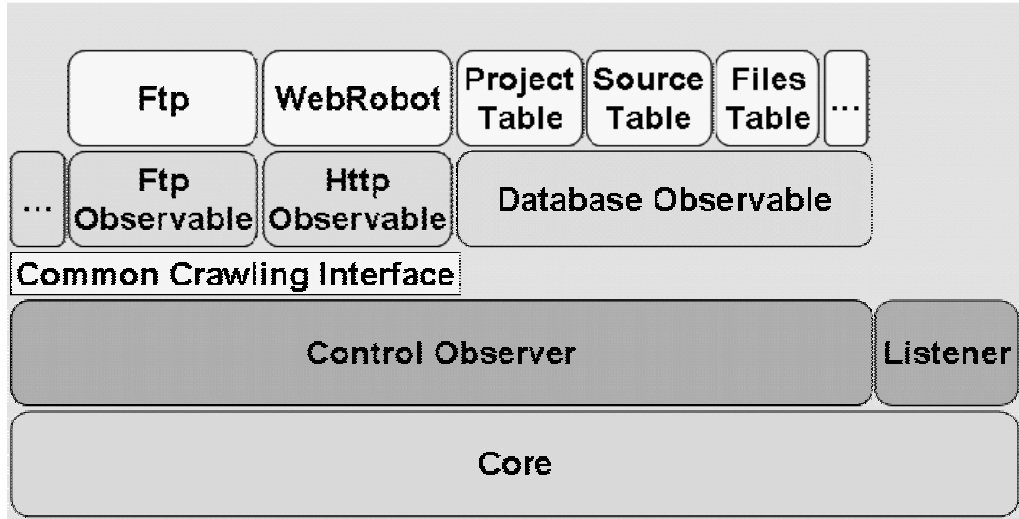


Figure 5-2 the architecture of the network crawling unit

The ControlObserver is the control center in the unit. By publish-subscribe mechanism, it manages the states between the three components discussed just now. In FtpObservable and HttpObservable, we implement these states introduced in Chapter 3: START, INIT, MIRROR, UPDATE, and CLOSE. For integrating the DatabaseObservable, we also design some states in it. The Figure 5-3 shows these states.

| FtpObservable | HttpObservable | DatabaseObservable |
|-----------------|-----------------|------------------------|
| START | START | INIT |
| START_COMPLETE | START_COMPLETE | INIT_COMPLETE |
| INIT | INIT | INSERT |
| INIT_COMPLETE | INIT_COMPLETE | INSERT_COMPLETE |
| MIRROR | MIRROR | UPDATE |
| MIRROR_COMPLETE | MIRROR_COMPLETE | UPDATE_COMPLETE |
| LIST | LIST | |
| LIST_COMPLETE | LIST_COMPLETE | ControlObserver |
| UPDATE | UPDATE | GO |
| UPDATE_COMPLETE | UPDATE_COMPLETE | REPORT |
| CLOSE | CLOSE | EXIT |
| CLOSE_COMPLETE | CLOSE_COMPLETE | |

Figure 5-3 the states in network crawling unit component

Clearly, there are two type states in the FtpObservable and HttpObservable. There are the one type states followed by the common crawling interface and the others states is report the previous states is finished. For example, after the MIRROR state is finished, the MIRROR_COMPLETE would be triggered. The DatabaseObservable has these two type states, too. In the DatabaseObservable, there are three action states. One is INIT representing the program gets the database information and connect to. And then the ControlObserver can query the project information and other data. Another is INSERT. It means we insert the Mirrored list into the database. The other is UPDATE. It means we update the records in the database according the UpdateList.

In Figure 5-3, we also introduce the states of the ControlObserver. After getting the project running signal, the GO state is triggered. First, the ControlObserver would change the state to DatabaseObservable.INIT for the project settings. According the queried information, ControlObserver will change the state to the START state of the right crawling component. After finishing crawling project, it will change the state to the REPORT. It will generate the crawling log according the Mirrored list or UpdateList. Finally, the program goes to the ControlObserver.EXIT state and closes. Figure 5-4 shows a state sequence of the mirror project of FTP.

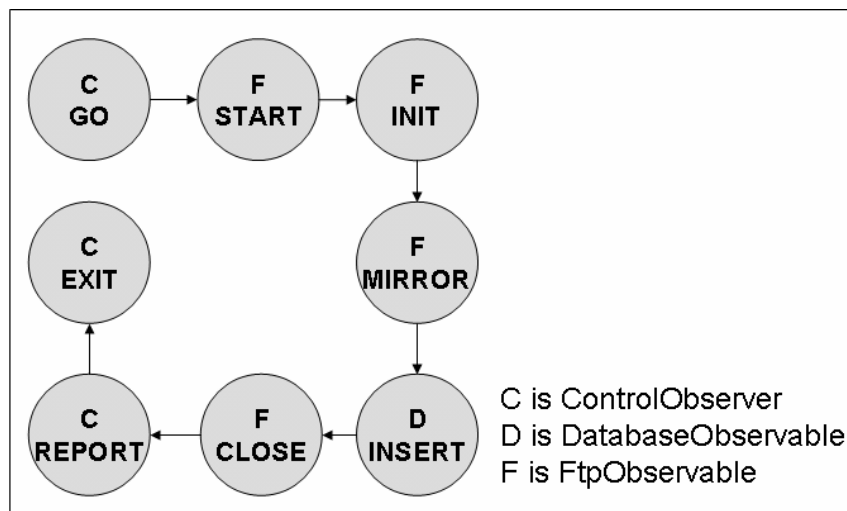


Figure 5-4 the state sequence of a FTP mirror project

The class Listener is always listening on the network socket and waiting for the connection from the Web UI. It is bind on the port 3000. The Web UI will send the commands from KMS admin to the socket. The Listener will parse the message and trigger the ControlObserver to the state GO and then work for the projects. The last class Core is the start class. In Figure 5-1, we realize the network crawling units, the log viewer, and the content publisher running in the same machine. So the main job of the class Core is generating the thread for them.

All in the section discussed is implemented with Java. So it would be easily port to various hardware architecture computers. We use the version is 1.4.2. The JoBo we used is 1.3. The JvFTP is 0.72. The JDBC here is mysql-connector-java-3.1.11.

5.3. Internal Database

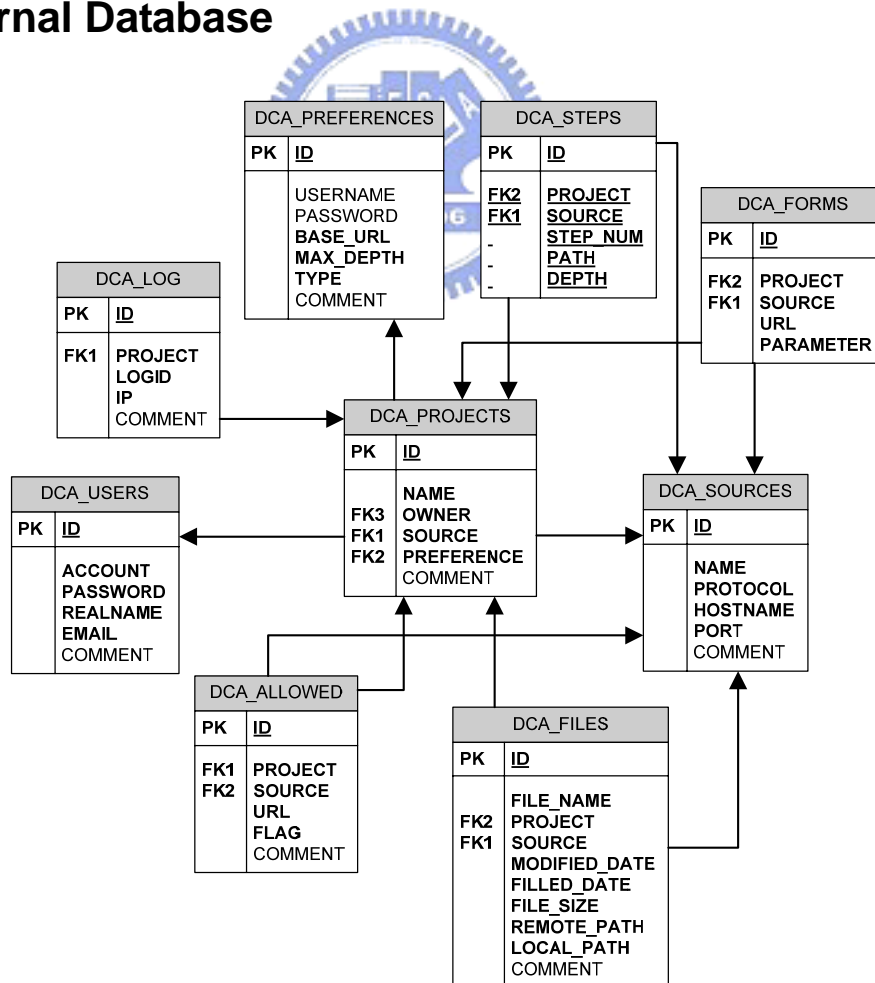


Figure 5-5 the ER model of the internal database

Figure 5-5 shows the schema designed in the internal database. The string “DCA” is just a table prefix and has no special meaning. We explain each field in the tables and their meanings.

DCA_PROJECTS

The table stores the basic project settings. It stores the unique id that will be generated automatically, the name, and the comment. It also refers the owner, the source and preference in the project. The detail of each field is as followed.

- ID: the unique id is generated automatically by the database identifying each project.
- NAME: the project name that is specified by the owner.
- OWNER: It refers DCA_OWNER and point out the project creator.
- SOURCE: It refers DCA_SORUCE and point out the target source.
- PREFERENCE: It refers DCA_PREFERENCE and point out the configuration.
- COMMENT: The values here are only Checked and Unchecked. The former the data crawled last time is checked by KMS admin and the data is ready to delete or update. The later is not. So the project won't be executed until the value is changed to Checked.

DCA_SOURCES

The table stores the settings about the remote data sources including the name, the protocol, the hostname, the port number, and the comment.

- ID: the unique id is generated by the database identifying each data sources.
- NAME: the readable name of the data sources.
- PROTOCOL: the protocol type of the data source, like HTTP, or FTP.
- HOSTNAME: the hostname of the data source.
- PORT: the port number of the data source, like 80 or 21.
- COMMENT: we can store some description to explain what the data source is.

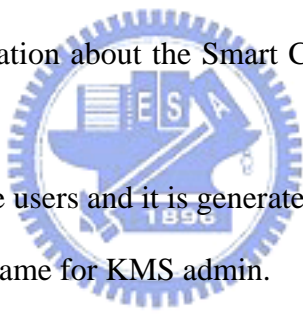
DCA_PREFERENCES

The table stores the preference used in the crawling operation. The information provides the network crawling unit to do crawling in the normal condition. The setting includes the username, the password, the base URL, the type, the max depth, and the comment.

- ID: the id identified the preferences and it is generated automatically.
- USERNAME: if the data source needs authentication, put the username here.
- PASSWORD: if the data source needs authentication, put the password here.
- BASE_URL: store the start point of the crawling.
- MAXDEPTH: store the max recursive level of crawling.
- TYPE: if the value is not 0, it stands for the session id length of URL rewriting.
- COMMENT: describe the main idea of the preference.

DCA_USERS

The table stores the information about the Smart Crawler user. In another word, it stores the data of KMS admin.

- 
- ID: the id identified the users and it is generated automatically.
 - ACCOUNT: account name for KMS admin.
 - PASSWORD: the corresponding password.
 - REAL_NAME: the truly name of KMS admin.
 - EMAIL: the email address of KMS admin.
 - COMMENT: some readable explanation.

DCA_FILES

The table stores the metadata of the downloaded data. In this table, it will refer the project and the source pair and identifies the corresponding file that is crawled from.

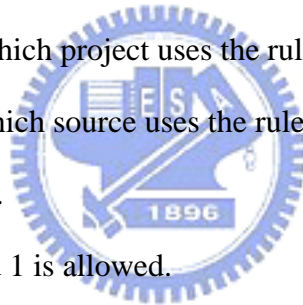
- ID: the id identified the rules and it is generated automatically.
- FILE_NAME: the file name.
- PROEJCT: store the id of the project that downloads the file.
- SOURCE: store the id of the source where the file comes from.

- **MODIFIED_DATE**: the modified date of the file.
- **FILLED_DATE**: the crawled date of the file.
- **FILE_SIZE**: the size of the file.
- **REMOTE_PATH**: the file's URL on the remote server.
- **LOCAL_PATH**: the local path stored the file.
- **COMMENT**: the file's description.

DCA_ALLOWEDS

The table stores the URL patterns and their permission. They will be used to determine the URL crawled is allowed or not. In this table, it will refer the project and the source pair and identifies the target that would be used by.

- **ID**: the id identified the rules and it is generated automatically.
- **PROEJCT**: it means which project uses the rule record.
- **SOURCE**: it means which source uses the rule record.
- **URL**: the URL pattern.
- **FLAG**: 0 is denied and 1 is allowed.
- **COMMENT**: the purpose of the rule can be filled here.



DCA_STEPS

The table stores the settings of Multi-Steps.

- **ID**: the id identified the steps and it is generated automatically.
- **PROEJCT**: it means which project uses the step record.
- **SOURCE**: it means which source uses the step record.
- **STEP_NUM**: it stands for the order of the steps in the same project.
- **PATH**: the URL in this step.
- **DEPTH**: the corresponding depth.

DCA_FORMS

The table stores the information about HTTP forms. According the information stored

here, HttpObservable will fill them automatically and submit. So we need to store the form action path and variable values.

- ID: the id identified the form setting and it is generated automatically.
- PROEJCT: it means which project uses the form record.
- SOURCE: it means which source uses the form record.
- URL: the form URL. It can be found in the attribute “ACTION” in the tag “FORM”
- PARAMETER: the form variable values. The format is introduced in Chapter 3.

DCA_LOG

The table stores the information about the log files. In the table, we can query the log file according the logid that is generated in each executing project and the IP where to run the project.

- ID: the id identified the form setting and it is generated automatically.
- PROEJCT: it means which project uses the step record.
- LOGID: it is a unique string representing the log file.
- IP: the machine generates the log file and stores it.
- COMMENT: the readable created date of the log file.

We use MySQL 4.1.12 that is running with the Mandriva Linux 2006 in a Pentium 4 2.8G computer.

5.4. Web Server with PHP for User Interaction Units

In the section, we introduce the implementation about the Web UI. We use PHP 4 to implement Web pages to interact with KMS admin and the network crawling units. Through these Web pages, KMS admin can do many works. We summarize all as follows:

1. New or modify the project, the data source, and the preference.
2. New or modify the allow rules, the form parameters, and the Multi-Steps configuration for the specified project.

3. Execute the specified project on the target machine and view its result report.
4. Explore the downloaded contents and download them.

Figure 5-6 shows the tree structure of these Web Pages. The root node is `index.php`. We depart the tree into five sub-trees in different blocks. The functions and members of each blocks is discussed as followed.

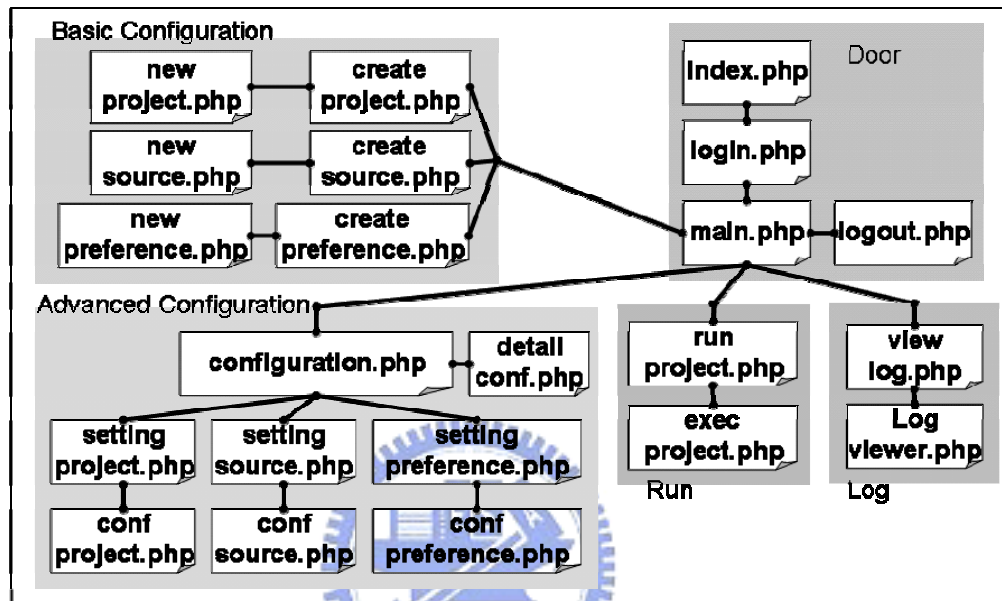


Figure 5-6 the tree structure of web pages

Door Block

When KMS want to use Smart Crawler, the block is the first content he accesses. The man function of the block is authentication. If the login was success, the `main.php` would show the links to the other blocks.

index.php: show login form

login.php: verify the login information.

logout.php: logout the user.

main.php: provide the links to the other blocks after login.

Basic Configuration Block

The block provides the function for KMS admin to create the project, the source, and the preference. KMS admin could fill the information and request for the data source and the

preference, first. And then, he could create a project by choose the right data source configuration and preference.

create_proejct.php: provide the form to select the source and the preference.

create_source.php: provide the form to fill the data of the data source.

create_preference.php: provide the form to fill the preference.

new_project.php: process the data passed from the create_project.php and new a record in the internal database.

new_source.php: process the data passed from the create_source.php and new a record in the internal database.

new_preference.php: process the data passwd from the create_preference.php and new a record in the internal database.

Advanced Configuration Block

There two main functions in the block. One is modifying the exist settings. It provides KMS admin to change the setting content of the project, the source, and the preference. The other is editing the project specified rules. For example, the form parameters, the allow rules, and the Multi-Steps information.

configuration.php: KMS admin can choose the setting of project, source, or preference to edit the exist setting. It will call the corresponding php file.

setting_project.php: provide the form to edit the project setting.

setting_source.php: provide the form to edit the data source setting.

setting_preference.php: provide the form to edit the preference setting.

conf_project.php: process the setting from setting_project.php and update the record in the internal database.

conf_source.php: process the setting from setting_source.php and update the record in the internal database.

conf_preference.php: process the setting from setting_preference.php and update the

record in the internal database.

detail_conf.php: provide the form for the form parameters, the allow rules, and the Multi-Step configuration and update the records in the database. It also provides KMS admin to new/delete each record and synchronize with the records in the internal database.

Run Block

KMS admin can use these pages in the block to execute a project in the specified computer that runs the network crawling units.

run_project.php: provide a form for KMS admin to select the project id and the target destination to execute the specified project.

exec_project.php: process the data from *run_project.php* and connect to the specified machine for notifying to run the specified project.

Log Block

The block provides KMS admin to access and explore the result report. The reports are stored in the remote computer that ran the project and generated the report. Hence, the Web pages in the block will connect to the Log Viewer and show the report content in the browser.

view_log.php: browser the project result report list and let KMS admin to choose one.

log_viewer.php: connect to the Log Viewer and show the result content.

We will use an example and screen shots to show how to use the user interaction units.

5.5. Content Publisher Units

In the section, we introduce our design of the content publisher units. There are two part in it, one is the log viewer that provides result reports, and the other is the content publisher that provides data contents.

5.5.1. Log Viewer

We introduce the Log Viewer in the section. The main job of the component is to be a communicator between the network crawling units and the user interaction units for transferring the log result report. The Log Viewer is executed with the network crawling units but it does not crawl. Hence we discuss it separately. It starts by the Core class of the network crawling units. Log Viewer runs as a Java thread and listens on the port 3001. We program the Log Viewer to a tiny web server but it is not a really web server. Unlike the standard HTTP request, the Log Viewer only takes the form request: `GET /log-id HTTP/http_version` or response 404 File Not Found. If the file of the `log-id` is found, the Log Viewer would send response and the file content, otherwise response “404 File Not Found”. Figure 5-7 shows its architecture.

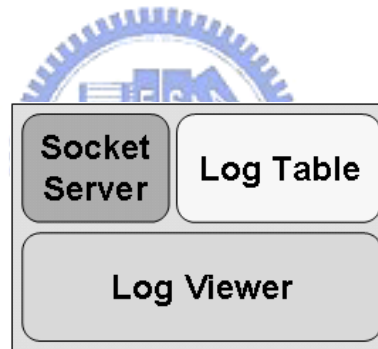


Figure 5-7 the architecture of the Log Viewer

First, the Log Viewer would be ran as a Java thread by the Core class of the network crawling units. The Socket Server class will bind and listen from the port 3001. If there is a request coming, Log Viewer will check the request form first. If the form is legal, the Log Viewer will find the requested report file according the log-id. It will use the Log Table class to query the logid is exist or not. And then search the file in the local file system. If the file is found, it will send a response with the file content and pack it with the HTTP format. Otherwise, response “404 File Not Found”.

5.5.2.Content Publish

Like the Log Viewer, the content publish will run as a Java thread by the Core class of the network crawling units. The unit would bind and listen in the port 3002. The mechanism is also the same with the Log Viewer replaced to the file-id instead of the log-id, otherwise response “404 File Not Found”. Figure 5-8 shows the architecture.



Figure 5-8 the architecture of the Content Publisher

In the beginning, the content publisher is evoked by the network crawling units and runs as a Java thread. The Socket Server class binds and listens from the port 3002. If there is request coming, it would check the format of the request first. If legal, the content publisher will query the local path of the request file-id from the internal database by the class Files Table. If the path is found and the file is found, pack the file content with the HTTP response as response. Otherwise, response “404 File Not Found”.

By the simple mechanism, KMS can easily and automatically access the downloaded files as long as parse the result report and collect the wanted file-ids. The KMS admin also can explore the downloaded file by the browser append the file-id to the URL.

5.6.Create Projects, Crawl, Track, and Explore Reports

In the section, we introduce the usage of Smart Crawler. We will take an example and show how to use step by step to show to use Smart Crawler. The scenario is we use Smart Crawler to crawl the articles about topic “Technology” and “Finance” on the website Taiwan Yahoo! news <http://tw.news.yahoo.com/>. Besides, we also want to collect news about Google. So first

we have known something: the hostname is tw.news.yahoo.com, the protocol is HTTP, the port number is 80, and the base URL is /.

Login to Smart Crawler

Before we use Smart Crawler, we should login first. Figure 5-9 shows the login page and Figure 5-10 shows the page after login successfully.

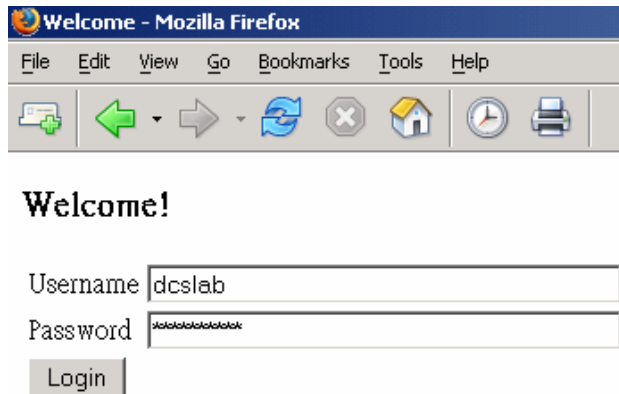


Figure 5-9 login page

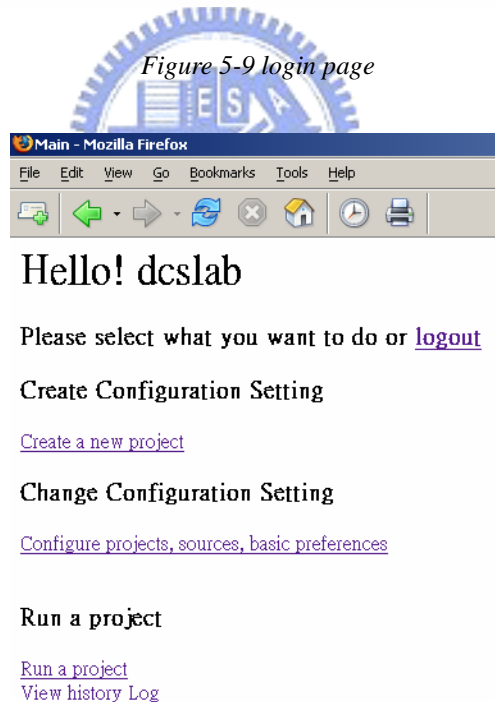


Figure 5-10 the main page after login successfully

Create a new Project, Source, Basic Preference

Now, we have to new a new project. So press the link “Create a new project”. We will see the page shown in Figure 5-11.

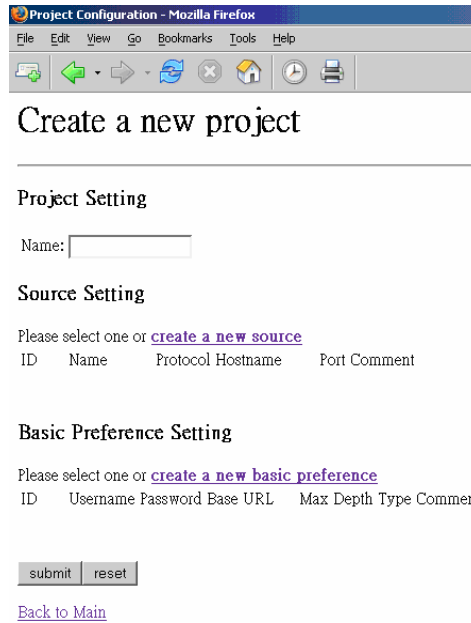


Figure 5-11 create a new project

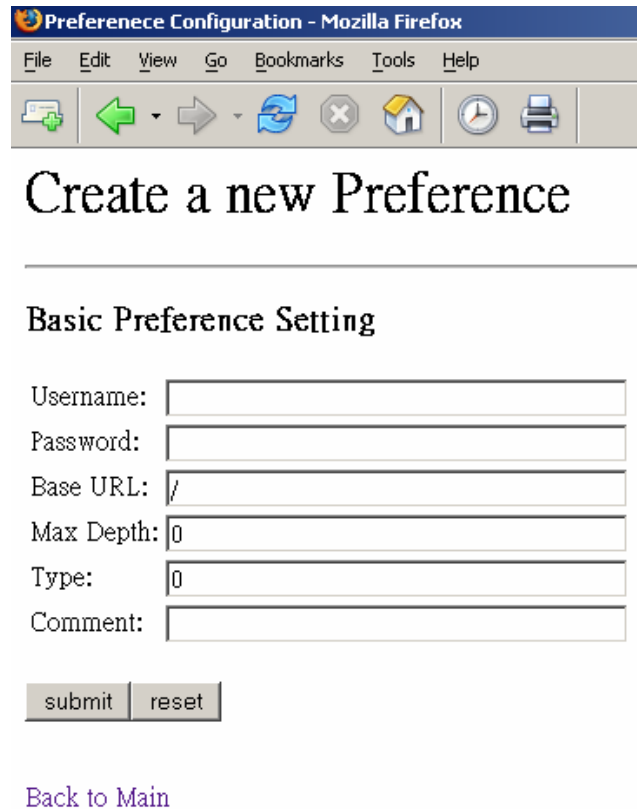
Before we actually create a new project, we have to create the data source and the preference first. We can see the both links in Figure 5-11. The Figure 5-12 and Figure 5-13 will show the pages just click the links “create a new source” and “create a new basic preference” and we will fill the right information.



Figure 5-12 create new sources

In the data source setting form, we name the data source “Yahoo News” and comment it

“Taiwan Yahoo News”. The protocol HTTP is selected and the other information is filled correctly. In the basic preference setting form, we set max depth 0 because we do not want Smart Crawler extracting links here instead of Multi-Steps setting. Another variable we set is type. The website is not implemented by URL rewriting. So the session id length is 0.



The screenshot shows a Mozilla Firefox window titled "Preference Configuration - Mozilla Firefox". The menu bar includes File, Edit, View, Go, Bookmarks, Tools, and Help. The toolbar contains icons for adding, navigating, and printing. The main content area displays "Create a new Preference" in a large serif font. Below this is a section titled "Basic Preference Setting" containing a form with the following fields: Username, Password, Base URL (with a slash in the input), Max Depth (with the value 0), Type (with the value 0), and Comment. At the bottom of the form are "submit" and "reset" buttons. A link labeled "Back to Main" is positioned below the form.

Figure 5-13 create new basic preference

After that, we can choose them when we actually create new project. We show that in Figure 5-14. We name the project is “Yahoo News”. In this time, we can see the new configuration in the source and the basic preference that we just add. Just select them and press submit button to create the new project.

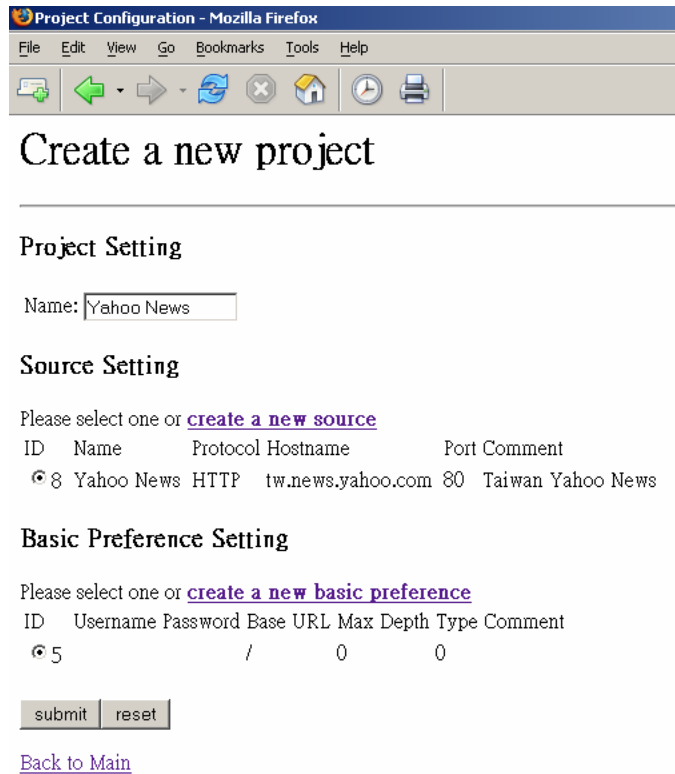


Figure 5-14 create the project

Setting Advanced Configurations

After we finish creating the new project, we have to add more settings. In Figure 5-10, there is a link named “Configure projects, sources, basic preferences”. It links to the configuration page that we show in Figure 5-15. There are four major parts. Three of these parts are changing the setting of the project, source, and basic preference or delete the settings. The fourth part is the link located after each project setting named detailed. The link links the page that can edit the advance settings including allow rules, form handler, and the most important Multi-Steps configuration.

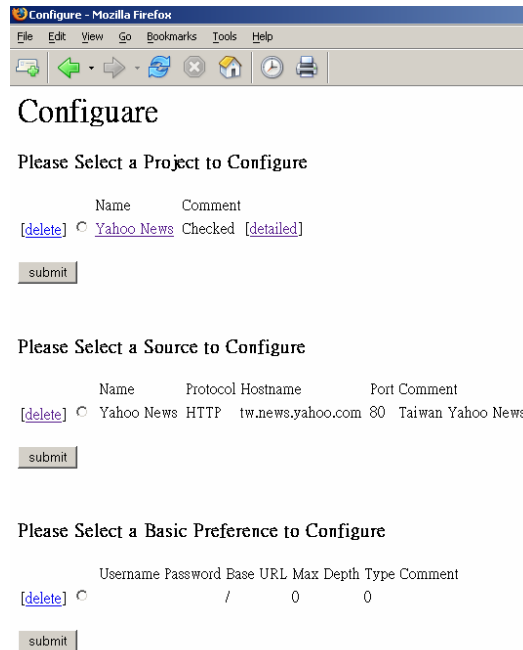


Figure 5-15 the configuration page

The detail configuration is shown in Figure 5-16. There are three parts for URL allowed rules, Multi-Steps set, and Form Filter. The way to add a new rule is just simple. That is filling the URL pattern in regular expression, selecting allowed (true) or denied (false), and filling the optional comment. In our scenario, we have already added eight rules. Here are the explanations.

False ****default****

The string pattern is for the default rules. The value false means that everything is denied if there are no other rules satisfied.

True `http://tw.news.yahoo.com/technology/`

We need to download the technology news. The link is the main introducing page.

True `http://tw.news.yahoo.com/finance/`

We need to download the finance news. The link is the main introducing page.

True `http.+/tw.yahoo.com.tw/[0-9]+/[0-9]+/[a-zA-Z0-9]+.html`

The regular expression pattern describes the URLs set of the news documents. Take an example: `http://tw.news.yahoo.com/060525/195/36cky.html`.

True `more.php?newsclass=technology`

There pattern describes we allow the links to show more articles of the class technology.

True `more.php?newsclass=finance`

There pattern describes we allow the links to show more articles of the class finance.

True `http://tw.news.yahoo.com/search.php`

It is the URL of the attribute action of the search form. We have to allow it so that Smart Crawler will fill them automatically and submit. And then we can download the result.
 True http://tw.search.news.yahoo.com/

It is the prefix of the URL that will show the search result. So we have to permit it.

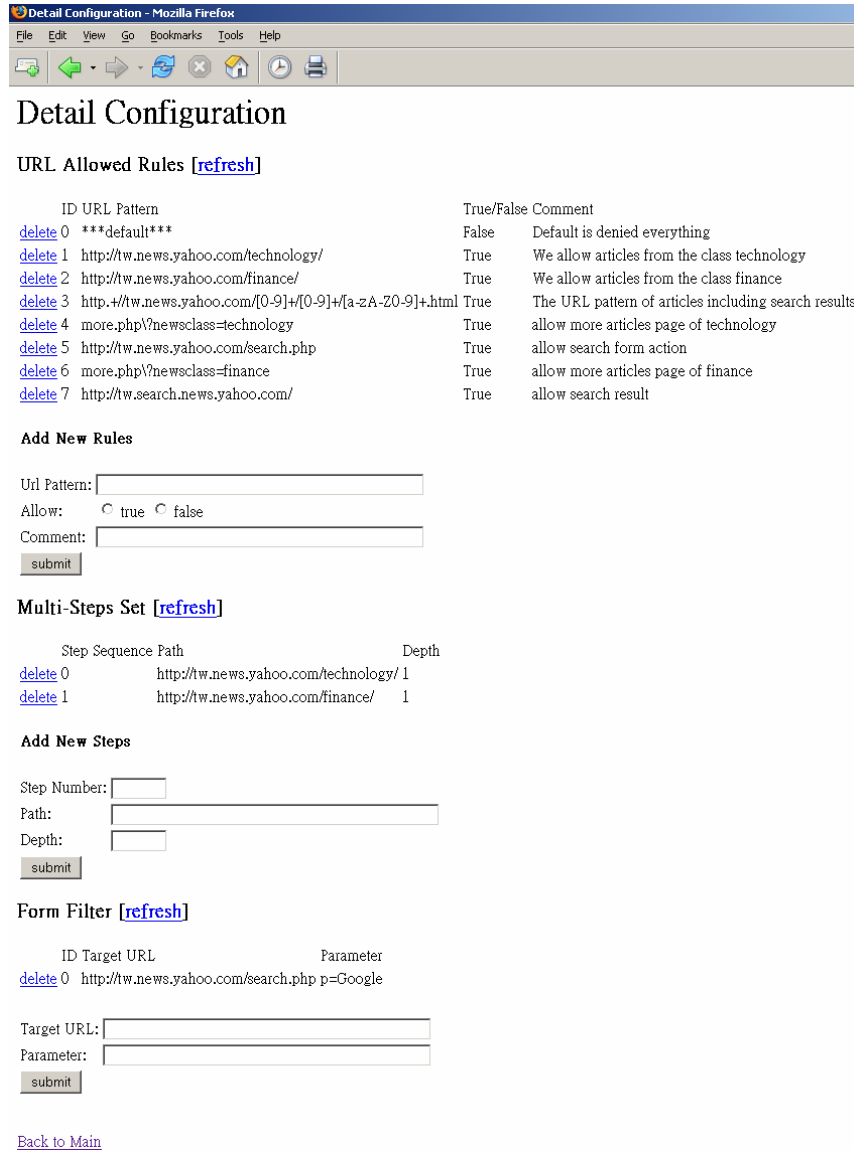


Figure 5-16 the detail configuration

After all, we finish our customized download rules. The next step is configuring the Multi-Steps set. The way is very thing that we just fill the step number first. And then we fill the URL and its depth of the step. After pressing submit, we finish adding a new step. Here we add two steps: one is for technology news and the other is for technology news. They all have the same depth 3. The last configuration is filling the form information so that Smart Crawler will fill them automatically. Here is the only form we use is to search articles about

Google. Hence, we fill the URL of the attribute action in the form from the HTML source code and then fill the variable p of the corresponding value Google. The variable p is the text field for input. After all, we complete our configurations. The whole information of the process can be got by going through manually at the same time.

Crawling & Tracking

Now we can run the project to collect data we want. In Figure 5-10, there is a link named “Run a project” bellow the section “Run a project”. The link links to the page that will execute the selected project. The operation behind we have discussed before. All we need is filling the target machine IP, selecting project and choosing the operation. Figure 5-17 shows that. Here we would like to download only, so we only choose the mirror operation.

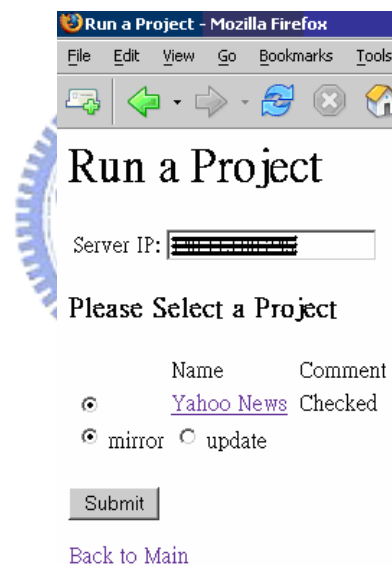


Figure 5-17 run a selected project

The Figure 5-18 shows the execution result. It will return the Log number. We can use it to identify the log in this time running. The track operation is quite the same except choosing the update operation. Another thing should be concerned is the comment of the project. The comment must and only be “Checked” and then the project will run normally. Otherwise, it means we have not check and verify the downloaded data. If we have done, use the link shown in Figure 5-15 to change the setting.

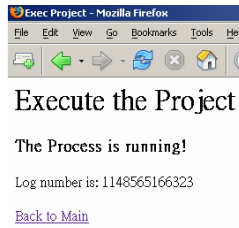


Figure 5-18 the execute result

View result report and downloaded files

We press the link “View history log” shown in Figure 5-10 and then we can choose the project that we want to the report of. The query result shows in Figure 5-19.

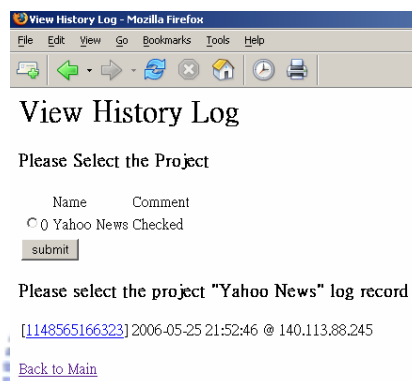


Figure 5-19 the report query result

Obviously, just press the link named with the log number we mentioned in Figure 5-18 and then we can explore the result report. In the background, the browser will send a request to the log viewer in the target machine and then show the response on the screen.

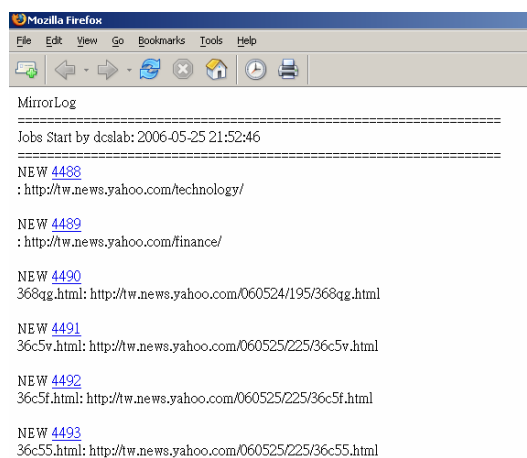


Figure 5-20 the result report

In the report, we mark the file id in the database as a hyperlink. If we press it, the browser will send the request to our content publisher unit. The content publisher unit will query the file id and find the file instance. If the target file is exist and found, the content publisher unit will send the content as the response. The detail operation is discussed in previous sections. The Figure 5-21 shows the content returning.



Figure 5-21 the content of downloaded data

In the example scenario, we show how to use Smart Crawler with step one by one. Smart Crawler is quite easy to use and with great power. Not only it can collect data from the remote server but it also track the data and report the changing status of each file. Besides, we also show the great power of Multi-Steps. In the traditional spider, there is only a base URL given. It is very difficult to describe the data clearly in our example. The downloaded data must be mixed and disorder. Through Multi-Steps, we can easily describe what we really care about.

Chapter 6 Evaluation

In the section, we introduce some evaluation about our proposition. One is to compare the performance of the finding the fixed i -length common substring in two known string. Second, we do a simple test to evaluate the correction rate of the algorithms with three finding fixed i -length common string algorithms. And then, we compare the Web UI with the current spiders' usage. Finally, we show the actually execution results in the real world.

6.1. Performance of Find Fixed i -length CS Algorithm

In the section, we do a simple test for three fixed i -length CS finding algorithm. First we introduce the computer we use:

CPU: Pentium M 1300MHz

Memory: 768 MB

Operation System: Microsoft Windows XP

Java Version: 1.4.2_09-b05



The test method is first we choose one pair long link strings. And then we find all the session id candidates totally running 10000 times and record the cost time for three algorithms.

The test link strings are as followed and we mark the actual session id which length is 19:

A: http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Dgf_gw_wine%2F104-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.wine.com%2Fpromos%2Famazonwine.asp%253Fan%253Damazon%2526s%253Damazon%2526cid%253Damazon%255Fgateway%255Ffpbox%26token%3D21513C47B07C95040C8597937CAB64718E97425C&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN

B: http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Damb_link_29290301_30%2F104-5791018-2027935%3Flocation%3Dhttp%3A%2F%2Fwww.amazon.com%2Fgp%2Fsearch.html%2F%253Fnode%253D3888811%2526keywords%253DDesigner%2526index%253Djewelry%2526rank%253D-price%2526x%253D7%2526y%253D14%26token%3D20975AF20B5B70DB8D6C01314A59A3FFC56364D7&pf_rd_p=162964201&pf_rd_s=center-8&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=15BNH99VVKBB0CDBQGQN

KMP stands for the algorithm which uses KMP pattern match to find the candidates. BM

stands for the algorithm which uses Boyer-Boore pattern match to find the candidates. CS is the algorithm that finds candidates by strait through comparison.

| | KMP | BM | CS |
|------------------------------|--|--|--|
| Cost Total Time | 991 ms | 871 ms | 58014 ms |
| Cost Time per Finding | 0.0991 ms | 0.0871 ms | 5.8014 ms |
| Found Candidates | 104-5791018-2027935 5BNH99VVKBB0CDBQGQN | 104-5791018-2027935 5BNH99VVKBB0CDBQGQN | 104-5791018-2027935 5BNH99VVKBB0CDBQGQN |

Table 6-1 the performance of finding fixed i-length algorithms for website 1

We also do the same test for other pairs of the links for the average condition. The session id length is still 19.

A: http://www.amazon.co.jp/exec/obidos/flex-sign-in/503-3240550-1917500?opt=n&page=help/ya-sign-in-secure.html&method=GET&cont-type=recs&cont-page=recs/signed-in-continue&response=tag/recs/recs-post-login-dispatch/--/recs/ref=pd_fu_gw_fav_h

B: <http://www.amazon.co.jp/exec/obidos/flex-sign-in/503-3240550-1917500?opt=ab&page=help/ya-sign-in-secure.html&response=order-history-filtered&method=POST&ss-order-filter=wheres-my-stuff&return-url=order-history-filtered>

Table 6-2 shows the result:

| | KMP | BM | CS |
|------------------------------|--|--|--|
| Cost Total Time | 450 ms | 441 ms | 12227 ms |
| Cost Time per Finding | 0.045 ms | 0.0441 ms | 1.2226 ms |
| Found Candidates | 503-3240550-1917500 sign-in-secure.html | 503-3240550-1917500 sign-in-secure.html | 503-3240550-1917500 sign-in-secure.html |

Table 6-2 the performance of finding fixed i-length algorithms for website 2

The third pair we test. The session id length is 32.

a: <http://webcaspar.nsf.gov/MyWebCASPAR/UserInfoSetVariables.jsp;jsessionId=C9204C12657A779A96E8D9A9D4CC5712?operation=insert®isterNew=1&subHeader=AccountInfo>

b: <http://webcaspar.nsf.gov/WhatsNew/whatsNewController.jsp;jsessionId=C9204C12657A779A96E8D9A9D4CC5712?type=history&subHeader=WhatsNew>

| | KMP | BM | CS |
|------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| Cost Total Time | 220 ms | 280 ms | 4727 ms |
| Cost Time per Finding | 0.022 ms | 0.028 ms | 0.4727 ms |
| Found Candidates | C9204C12657A779A 96E8D9A9D4CC5712 | C9204C12657A779A 96E8D9A9D4CC5712 | C9204C12657A779A 96E8D9A9D4CC5712 |

Table 6-3 the performance of finding fixed i-length algorithms for website 3

According to these tables, they all can find the same number of candidates including the right session id. And we can see clearly and obviously that the performance of CS is far worse than the other two. The performance of BM is as good as the one of KMP. We also can say that the performance of all with the order: $KMP = BM > CS$. In some cases, CS costs more than 30 times longer than the others. If the strings are much longer, it might cost 90 times longer.

In the evaluation, we also prove that the time depends strongly on the compared string length, not the fixed length. And it also tells us we do not find all common substrings and then choose the strings with the specified length. It might cost more. We can determine the substring with the specified length to see if the pattern is matched in another string using the more efficient algorithm or not. That would be better and faster with the same accuracy.

The Figure 6-1 shows the summary of the three comparisons.

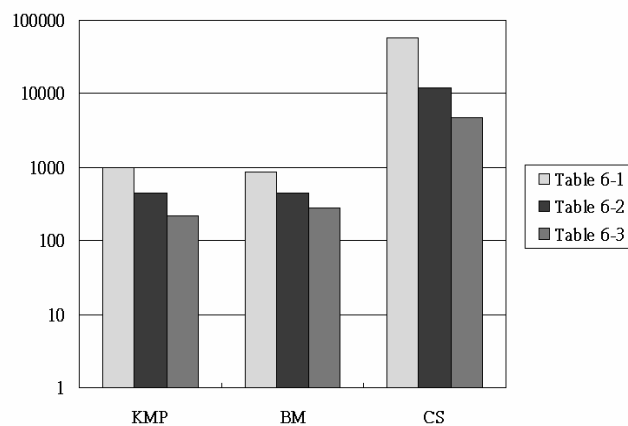


Figure 6-1 the result diagram of three comparisons.

6.2. Correctness Evaluation of Finding Session ID Algorithm

In the section, we do an evaluation about the correction and performance of finding session id algorithms that uses KMP, BM, and CS fixed i-length common string algorithms.

We run the experiment on the computer:

CPU: Pentium 4 2.8GHz

Memory: 1GB

Operation System: Mandriva Linux 2006

Java Version: 1.4.2_11-b06

We find four websites where the session id is embedded in the URLs on. They are <http://www.amazon.com/>, <http://www.amazon.co.jp/>, <http://webcaspar.nsf.gov/>, and <http://www.jesishpittsburgh.org/>. The first two websites have the same type session id and the same id length, 19. The id on the third website is embedded by the variable name `jsessionId` and its length is 32. The id on the last website is embedded by the variable name `PHPSESSIONID` and its length is 32, too. We make an example to explain the four websites. In our algorithm, we have to choose the two longest URLs first, called a, and b. The second step is finding the candidates by the right algorithm. The last step is counting the number of each candidate that appeared of all crawled URLs. The most one is the session id. So we list the detail information as followed and mark the session id in the special format.

Website 1: <http://www.amazon.com/>

```
a:http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Damb_link_32592101_14%2F104-0018288-3648739%3F%255Fencoding%3DUTF8%26location%3Dhttp%253A%252F%252Fwww.amazon.com%252Fgp%252Fsearch.html%252F%253Fnode%253D3407781%2526%2525255Fencoding%253DUTF8%2526keywords%253Damznclr05%2526index%253Dsporting%2526rank%253Dsalesrank%26token%3DCA472842B414C6A82DB974F7A7E14FE3CB2782C0&pf_rd_p=169076201&pf_rd_s=center-8&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=07F8ZVRY1X1F2XCAYGDH
b:http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Dgw_hp_ls_1_9%2F104-0018288-3648739%3Flocation%3Dhttp%3A%2F%2Fsl.amazon.com%2Fexec%2Fvarzea%2Fsubst%2Ffx%2Fhome.html%26token%3DDAA4717DDFF920A7B394398EC5886AEAF53EC766&pf_rd_p=181020701&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=07F8ZVRY1X1F2XCAYGDH
```

Candidates: 104-0018288-3648739 Count: 274

Candidates: 7F8ZVRY1X1F2XCAYGDH Count: 165

Website 2: <http://www.amazon.co.jp/>

a:http://www.amazon.co.jp/exec/obidos/flex-sign-in/503-8426031-4229552?opt=ab&page=help/ya-sign-in-secure.html&response=order-history-filtered&method=POST&ss-order-filter=wheres-my-stuff&return-url=order-history-filtered

b:http://www.amazon.co.jp/exec/obidos/redirect-to-external-url/503-8426031-4229552?path=http%3A//www.amazon.ca/exec/obidos/redirect-home%3Fsite%3Damazon%26tag%3Dintl-jphone-cahome

Candidates: 503-8426031-4229552 Count: 148

Website 3: <http://webcaspar.nsf.gov/>

a:http://webcaspar.nsf.gov/MyWebCASPAR/UserInfoSetVariables.jsp;jsessionid=17258A15BF7624A9AD510663867C80E2?operation=insert®isterNew=1&subHeader=AccountInfo

b:http://webcaspar.nsf.gov/WhatsNew/whatsNewController.jsp;jsessionid=17258A15BF7624A9AD510663867C80E2?type=history&subHeader=WhatsNew

Candidate: 17258A15BF7624A9AD510663867C80E2 Count: 8

Website 4: <http://www.jewishpittsburgh.org/>

a:http://www.jewishpittsburgh.org/cms/login.php?PHPSESSID=a9f771393bd485eb75638d9d77172335&PHPSESSID=a9f771393bd485eb75638d9d77172335

b:http://www.jewishpittsburgh.org/contact.php?PHPSESSID=a9f771393bd485eb75638d9d77172335

Candidate: a9f771393bd485eb75638d9d77172335 Count: 25

By these examples, we realize which type session id of these websites and their format. So we run our finding session id algorithm 100 times for each websites and evaluate the correction and performance between using the different algorithms finding fixed i-length. The result of the test is summarized in Table 6-4. It shows the times finding the correct session id and their cost time.

| | | KMP | BM | CS |
|------------------|-----------------------------|------------|-----------|-----------|
| Website 1 | Correctness Times | 100 | 100 | 100 |
| | Avg. Candidates | 2.35 | 2.47 | 2.68 |
| | Cost Total Time (ms) | 507381 | 496748 | 498849 |
| | Avg. Time (ms) | 5073.81 | 4967.48 | 4988.49 |
| Website 2 | Correctness Times | 100 | 100 | 100 |
| | Avg. Candidates | 1.04 | 1.02 | 1.02 |
| | Cost Total Time (ms) | 274862 | 270824 | 285865 |
| | Avg. Time (ms) | 2748.62 | 2708.24 | 2858.65 |
| Website 3 | Correctness Times | 100 | 100 | 100 |
| | Avg. Candidates | 1.00 | 1.00 | 1.00 |
| | Cost Total Time (ms) | 278812 | 279857 | 277363 |
| | Avg. Time (ms) | 2788.12 | 2798.57 | 2773.63 |
| Website 4 | Correctness Times | 100 | 100 | 100 |
| | Avg. Candidates | 1.00 | 1.00 | 1.00 |
| | Cost Total Time (ms) | 76594 | 82123 | 77829 |
| | Avg. Time (ms) | 765.94 | 821.23 | 778.29 |

Table 6-4 the evaluation result of calculating session id with networking

For the evaluation, first, we prove the algorithm that uses which one finding the fixed i -length common string algorithms; we can find the real session id if the id is exists. In the website 1, we find more than 2 candidates per finding. But we still find the truly session id no matter what algorithm we use. It also proves the candidate with biggest count number is the session id we want. Second we are told from the table, the performance is almost the same for the three algorithms for the average time. In the previous section, we show that there might more than 10 times time cost between these three algorithms. Even the algorithm that finds the fewest candidates with the best performance costs the most time in the website 1.

Why? The problem might because we only run 100 times for each algorithm. So the gap between these algorithms might be not so obvious that effect the cost time; the process of finding the session id includes connecting to the server, parsing links, calculating the candidates, and counting. The network of each connection might cost so much time in the whole process that the computation time seems be very small.

In order to prove the purpose, we take an experiment. We only connect to the website 1 once and then calculate the id 100, 200, 400, 800, 1600 and 3200 times for each calculation algorithm to evaluate their performance. We list them in Table 6-5.

| | | KMP | BM | CS |
|-------------|------------------------|------------|-----------|-----------|
| 100 | Total Time (ms) | 1361 | 1374 | 1977 |
| | Avg. Time (ms) | 13.61 | 13.74 | 19.77 |
| 200 | Total Time (ms) | 2525 | 2566 | 3742 |
| | Avg. Time (ms) | 12.63 | 12.83 | 18.71 |
| 400 | Total Time (ms) | 3547 | 3607 | 5042 |
| | Avg. Time (ms) | 8.87 | 9.02 | 12.61 |
| 800 | Total Time (ms) | 9843 | 9920 | 14711 |
| | Avg. Time (ms) | 12.30 | 12.40 | 18.39 |
| 1600 | Total Time (ms) | 14975 | 15121 | 20905 |
| | Avg. Time (ms) | 9.359 | 9.45 | 13.07 |
| 3200 | Total Time (ms) | 31915 | 32034 | 43857 |
| | Avg. Time (ms) | 9.97 | 10.01 | 13.71 |

Table 6-5 the evaluation result of calculating session id without networking

According the result, obviously the cost time of the network is much more than the computation time. Excluding the factor of the network, we prove again the performance of each algorithm is $KMP = BM > CS$. Figure 6-2 shows the difference of these conditions:

- A: Avg. time of whole algorithm process 100 times.
- B: Avg. time of calculation 100 times.
- C: Avg. time of calculation 400 times.

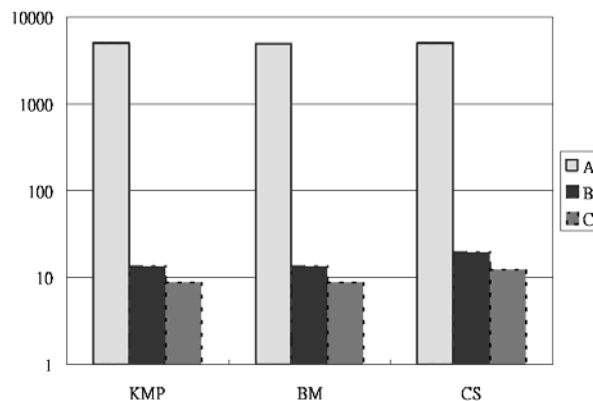


Table 6-6 the comparison of calculation time and network time

6.3. Evaluation of Multi-Steps Configuration

In the evaluation, we will use the scenario example used in the last section of Chapter 5. One reason is the scenario is used already. The other reason is there is only one competitor with FTP support but the program has only CLI. The competitors evaluated later here are JoBo, Teleport Ultra, Web Scraper Plus+ and PhoPicking. One of the common features of these is they all have the GUI and many configurations. Although the Where spider also has GUI but its purpose is a fully offline browser. So it does not provide enough customization functions. Open Spider is, too. The main purpose of the program is caching and indexing for users search later. Few customizations are provided. But we have to explain that if our target jobs is downloading images form the Internet, Smart Crawler might be not good enough. Later, we will compare each competitor with Smart Crawler one by one.

JoBo

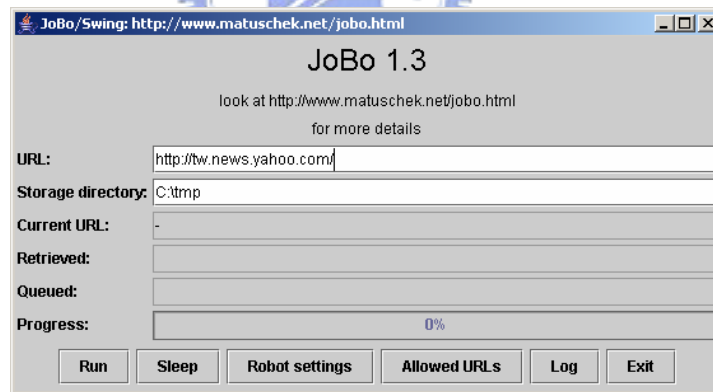


Figure 6-2 the configuration GUI of JoBo

Figure 6-2 shows the screenshot of JoBo. The start URL or seed is filled in the URL field. The strait way is filled the parent URL of <http://tw.news.yahoo.com/finance/> and <http://tw.news.yahoo.com/technology/>, <http://tw.news.yahoo.com/>. And then we can add the rules by pressing the Allowed URLs button. In our example, the depth of the two URLs is just the same. So setting in JoBo just adds one in the original depth. If the depth of each URL is

not the same, the downloaded data would be very chaotic and out of order by JoBo. Another problem; the GUI does not provide the interface for users to configure the needs form information. The only way is editing an xml file and put it at the right place. It is very inconvenient. Obviously, JoBo is not flexible and convenient enough in many cases, like our example.

Teleport Ultra

In teleport Ultra, there are many job purposes. The most right one is “Duplicate a website, including directory structure”. The selection “Create a browser copy of a website on my hard drive” is also the same. But the downloaded data will be sent to the KMS. Keeping the directory structure would be better. Figure 6-3 shows the next configuration screenshot.



Figure 6-3 the seed configuration screenshot of Teleport Ultra

Just like JoBo, there is only one URL required. So the same problem will happen here. The next step is shown in Figure 6-4. In the configuration, we can choose the type of the documents. In Smart Crawler, we also can do the same thing as long as we add the rules with the name of the file types. For example, “.+jpe?g\$” means the jpg or jpeg images. Obviously, the rules configure for not only images, sound or text but all file types. Smart Crawler has more flexible. After pressing the next button, we finish set the teleport Ultra project. Although

the configuration is simpler than Smart Crawler, it loses more flexible at the same time.



Figure 6-4 the download rules screenshot of Teleport Ultra

Web Scraper Plus+

In the program, its usage is quite very complicated. Before we add the new project, we can create the Datapages rules first. It describes the segment in the documents and the format how to save them in the target spreadsheets or a table in the database. Or, we can only choose download into the local hard drive as Smart Crawler as the same. And then, we can add new task. Figure 6-5 shows the screenshot. We choose the selection “Spider a website like Google does; start as a page and follow links that fit your criteria; download and categorize pages locally (Intermediate)”. We can add a new crawling task with start URLs.

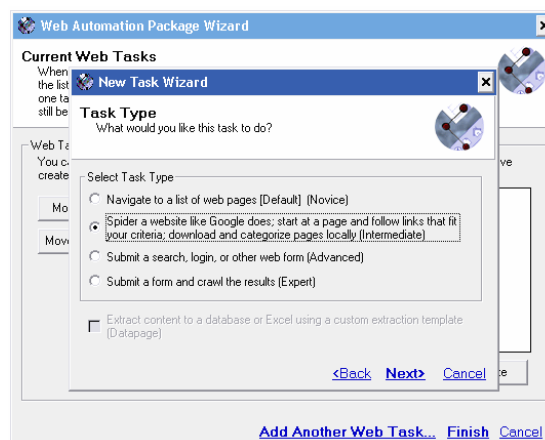


Figure 6-5 the adding new task screenshot of Web Scraper Plus+

After that, we add the two URLs as the start URLs shown in Figure 6-6. In Figure 6-7, we show the screenshot download rules and in Figure 6-8, we show the form handler UI.

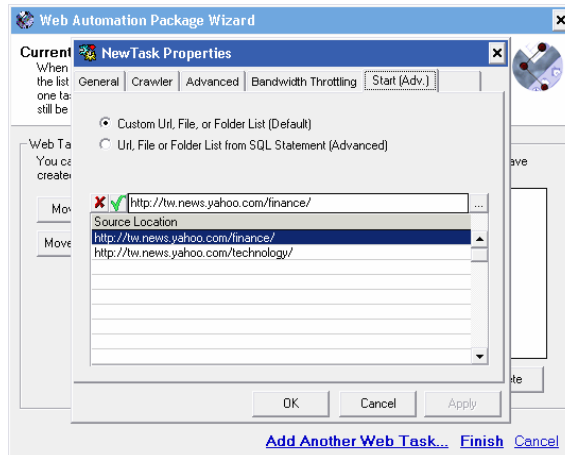


Figure 6-6 the seeds configuration screenshot of Web Scraper Plus+

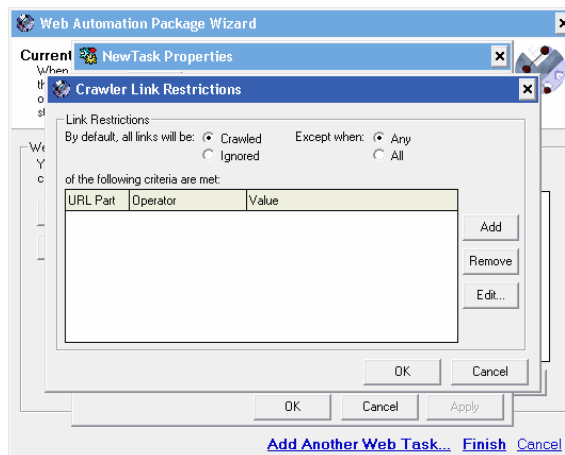


Figure 6-7 the download rules screenshot of Web Scraper Plus+

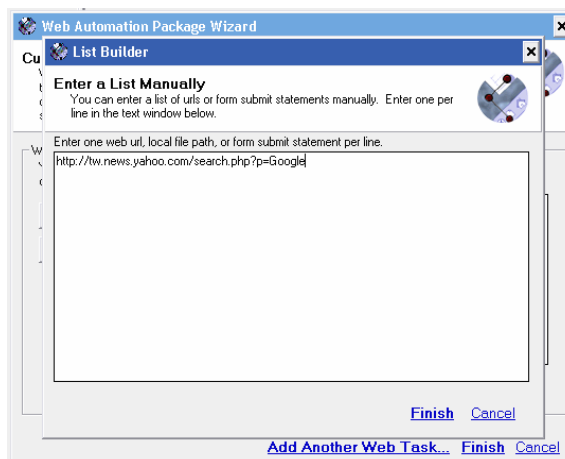


Figure 6-8 the form handler screenshot of Web Scraper Plus+

The GUI Web Scraper Plus+ seems have the same power with Smart Crawler. It has the function like Multi-Steps, the form handler, and the download rules. Besides, it has a very nice tool to help set the form information: Web Form Explorer. However, because of the wizard driven GUI, the user can't directly find where to set the setting soon. For Smart Crawler, the user can use many exist tools used in daily life to achieve the same function. Another disadvantage, it has to run on the special platform and only console configurations. The user has to use the program on the front of the computer. But, the user can use Smart Crawler as long as uses the any devices that can connect to the Internet and has a simple browser. In the other word, Smart Crawler can be used by the user anywhere. Besides, the network crawler unit is programmed by Java. It has the cross platform feature. In the previous evaluations, we prove our Smart Crawler can run on different platforms.

PhoPicking

The program's main target is different with Smart Crawler. So we can't use the example in Chapter 5 here. We will use another example for the program and Smart Crawler. The example is we use the tow programs to crawl all images on the PhoPicking exercise web page <http://www.phopicking.com/xpicweb/sample/sample1.htm>. For PhoPicking, Figure 6-9 shows the main configuration that sets the silver key's basic information.



Figure 6-9 the main setting screenshot of PhoPicking

In the page, we can set the name, the URL, the attributes and the authentication methods.

In the example, the page is two level recursive tables. The first one is listing the photo's title and separate into two pages. The second level is showing the actually images with full information. In PhoPicking, how many levels we want to crawl is how many levels we need to set. In the case, we only set two levels. Figure 6-10 shows the first level.





Figure 6-11 the second level configuration

For the same example, Smart Crawler only set the seed we mention before: <http://www.phopicking.com/xpicweb/sample/sample1.htm> and set the depth is 2. And then Smart Crawler will download everything automatically. Although PhoPicking can only download the images exactly what we want without any extra data, we have to fill more information than Smart Crawler. For Smart Crawler, although it fetches something we don't want, we can exclusive them in the report list before we ask KMS to access the downloaded data. Another problem is the setting we input of PhoPicking is only used in the website and this time. If the website is updated for another look, PhoPicking should be reset. But Smart Crawler might change nothing. In the setting process, PhoPicking obviously is more complex than Smart Crawler, and the setting might be useless if the website was changes its look only.

After comparing with each competitor, we show the advantage of Smart Crawler and its power. Of course, these competitors are still very good and well programs. But the Multi-Steps configuration does be better and more flexible than the traditional setting way. Maybe Web Scraper Plus+ has the same power with Smart Crawler for configurations, but the advantages of Web UI makes Smart Crawler have more benefits than it.

6.4. Evaluation for Smart Crawler Tracking Actually

In the section, we take an experiment that we use Smart Crawler to execute actually two real websites in the real world and show the results see if that is expected or not for our tracking algorithm. The two websites we take the test here are <http://www.amazon.com/> and <http://tw.news.yahoo.com/>. The first one website is designed by HTTP URL rewriting. So we will test our URL rewriting tracking algorithm. The other one is used to test our normal tracking algorithm and the other functions in Smart Crawler.

Website 1: <http://www.amazon.com/>:

For the website, we will download everything from the root node with depth 1, record the number of mirrored documents, and then mark them “NEW”. Here, we note the number of “NEW” documents. Later, we track with our URL rewriting algorithm and we will refresh all documents we have and we don’t have in the beginning. Here, we will get the number of “SAM”, “NEW”, “UPD”, and “MIS” documents. We summarize the result in the table 6-7.

| Action | Date | Result | |
|----------|---------------------|----------|---------|
| Crawling | 2006-06-06 20:31:29 | NEW: 340 | |
| Tracking | 2006-06-06 20:55:25 | SAM: 76 | NEW: 54 |
| | | UPD: 264 | MIS: 0 |

Table 6-7 the crawling and tracking result of the website 1

Website 2: <http://tw.news.yahoo.com.tw/>:

For the website, we will download every news article of the category about the technology and the finance with maximum depth 2 from <http://tw.news.yahoo.com.tw/technology/> and <http://tw.news.yahoo.com.tw/finance/>. Besides, we set Smart Crawler to fill the search form automatically for searching articles about the keyword “Google”. Here we will record the

number of “NEW” documents. After that, we track the website with the previous settings by our tracking algorithm. Here we will have the number of “SAM”, “UPD”, “NEW”, and “MIS” documents. We show the result in table 6-8.

| Action | Date | Result | |
|----------|---------------------|-----------------|---------------|
| Crawling | 2006-06-07 14:26:26 | NEW: 139 | |
| Tracking | 2006-06-07 14:40:49 | SAM: 19 | NEW: 3 |
| | | UPD: 120 | MIS: 0 |

Table 6-8 the crawling and tracking result of the website 2

Discuss

Before we discuss the result, we have to know one purpose of the execution time of crawling and the tracking. In the two websites, we take the two operations closely and nearly. We expect that the content of the two website would not change much: the removed links is few and the new links is, too. So we can reduce the link number of the “MIS” and evaluate our algorithm soon and quickly. If we get some links marked “MIS”, we can manually connect them by our browser see if it is removed or not. If we do not have any “MIS” link, it means all links are tracked successfully.

From the results of the two websites, we can find several things interesting. First one is we prove the tracking algorithm works well. Before we introduce the marks in the crawling and tracking algorithm, we know the “NEW” document of the crawling would be and only be one of “SAM”, “UPD”, or “MIS”. In the result of the website 1, we see that $264 + 76 + 0 = 340$. It means that Smart Crawler success to track every link got from the crawling without anything missing. The algorithm for URL rewriting that we propose can track every link successfully. In the website 2, we see that $19 + 120 + 0 = 139$. From the result, the algorithm we propose for normal cases can track every link successfully, too.

There is another thing we get from the result. The number of the “SAM” seems to be

unreasonable. It is too small. As we talk in the first paragraph, the content of these websites should not be changed heavily. So what's wrong with them? First, we compare the size of the document old and new copies. Obviously, they are not the same. For the website 1, every link has its own session id corresponding its session. But the session id has the fixed length so that the saved document in the disk will occupy the same space. In the other word, the different session id won't change the size of the document. In the website 2, the situation won't be happened. Hence, we have to compare the content of the old and new copies of the same link. It is caused by the different embedded advertisements in these copies. We show some differences of the website 1 in the Figure 6-12 and the website 2 in the Figure 6-13.

| |
|---|
| <p>File ID: 6035 (79860 bytes vs. 79859 bytes) # 926: the old one has an extra space #1293: different ad ids with the same length <!-- whfhr7X8WqegL5KJm2fmzqcqhCLvfvQNXsj8T --> <!-- whfha1iy0iz3eXXYpbQ3btGN0DRj0KZWHveI --></p> |
|---|

Figure 6-12 the difference between copies of the website 1

In Figure 6-12, we find two type differences. One is there is an extra space character in the old copy. So it causes the difference size with the new copy. Another type difference is the embedded advertisement ids between copies are not the same.

| | | |
|---|---|--|
| <p>File ID: 10106 (43814 bytes vs. 43804 bytes)</p> <table border="1"> <tr> <td> <p>if(window.yzq_p)yzq_p('P=on54Fcorw6zH2wocRIZvySBfjHfY9USGcC8AA_Qc&T=13u58kpsm%2fX%3d1149661231%2fE%3d152952142%2fR%3dtw_knews%2fK%3d5%2fV%3d1.1%2fW%3dJR%2fY%3dKIMO%2fF%3d2550867441%2fS%3d1%2fJ%3dDDC32BCA');</p> </td> </tr> </table> <table border="1"> <tr> <td> <p>if(window.yzq_p)yzq_p('P=1KARVMorw6yLfON_RIZw0ymLjHfY9USGcQMADvZ6&T=13tniacgs%2fX%3d1149661443%2fE%3d152952142%2fR%3dtw_knews%2fK%3d5%2fV%3d1.1%2fW%3dJR%2fY%3dKIMO%2fF%3d655722308%2fS%3d1%2fJ%3dF8C32BCA');</p> </td> </tr> </table> | <p>if(window.yzq_p)yzq_p('P=on54Fcorw6zH2wocRIZvySBfjHfY9USGcC8AA_Qc&T=13u58kpsm%2fX%3d1149661231%2fE%3d152952142%2fR%3dtw_knews%2fK%3d5%2fV%3d1.1%2fW%3dJR%2fY%3dKIMO%2fF%3d2550867441%2fS%3d1%2fJ%3dDDC32BCA');</p> | <p>if(window.yzq_p)yzq_p('P=1KARVMorw6yLfON_RIZw0ymLjHfY9USGcQMADvZ6&T=13tniacgs%2fX%3d1149661443%2fE%3d152952142%2fR%3dtw_knews%2fK%3d5%2fV%3d1.1%2fW%3dJR%2fY%3dKIMO%2fF%3d655722308%2fS%3d1%2fJ%3dF8C32BCA');</p> |
| <p>if(window.yzq_p)yzq_p('P=on54Fcorw6zH2wocRIZvySBfjHfY9USGcC8AA_Qc&T=13u58kpsm%2fX%3d1149661231%2fE%3d152952142%2fR%3dtw_knews%2fK%3d5%2fV%3d1.1%2fW%3dJR%2fY%3dKIMO%2fF%3d2550867441%2fS%3d1%2fJ%3dDDC32BCA');</p> | | |
| <p>if(window.yzq_p)yzq_p('P=1KARVMorw6yLfON_RIZw0ymLjHfY9USGcQMADvZ6&T=13tniacgs%2fX%3d1149661443%2fE%3d152952142%2fR%3dtw_knews%2fK%3d5%2fV%3d1.1%2fW%3dJR%2fY%3dKIMO%2fF%3d655722308%2fS%3d1%2fJ%3dF8C32BCA');</p> | | |

Figure 6-13 the differences between copies of the website 2

In Figure 6-13, we can see that the different advertisement ids with different lengths cause

the different size of copies. So Smart Crawler will mark these content “UPD” although the main content the human beings care about actually is identical. Hence we think that Smart Crawler should be provide the ability that the user can add the plug-ins that provide the customized comparing mechanisms.



Chapter 7 Conclusion

7.1. Conclusion

So far, we have described our Smart Crawler, and how it achieves the objectives mentioned in Chapter 1. First, we propose the architecture that supports multiple protocols. Through the architecture, we can integrate spiders with different network protocol support in. We also propose the tracking mechanism besides the crawling. For the mechanism, the design complexity and loading of KMS is slower. KMS admin also can realize the situation of the downloaded data to change the crawling/tracking frequency to balance the load of the remote servers and local network bandwidth. We also provide a solution for tracking on the HTTP session with URL rewriting. Through that, Smart Crawler can find the correct session id in each connection and find the corresponding documents rightly. At last, the Multi-Steps provide more flexibility and precision for describing the wanted data. Smart Crawler can download information precisely but less unexpected one.

7.2. Future Works

Now we finish the prototype of Smart Crawler but there are still many issues that could make the program more mature. We propose some future works that enhance Smart Crawler.

- Support more network protocol

Now we have finished the most popular network protocols HTTP and FTP. To make the program more complete, we must add other network types or protocols, like newsgroup, network file system (NFS), and so on as soon as possible.

- Content Comparison Plug-in

Different data might have different comparison mechanism. The comparison result is still the same even only the advertisements are not the same between two HTML documents. So In Smart Crawler, we should provide the ability that the comparison mechanism can be customized as a plug-in integrated in.

- Complete users and authentication

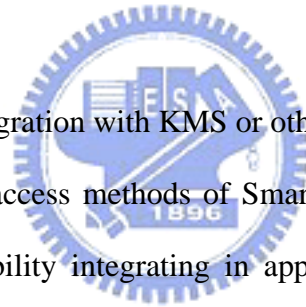
In Smart Crawler Web UI, now we just provide a simple account mechanism. The authentication is used for not only for users but the configurations and collected data. The idea way is that collected data is authorized as the same as its project user.

- Absent network usage

The main target is collecting data for the KMS, not exhaust all resources. It would be important that how to fine-grant collecting data by Smart Crawler without affecting the local or remote network resources.

- More automation and integration with KMS or other applications

We should provide more access methods of Smart Crawler for publishing the collected data. We also improve the ability integrating in applications like content filters in Smart Crawler. The more application can be integrated in together, the more automatic of KM process has.



Bibliography

- [1] Elias M. Awad, Hassan M. Ghazin, Knowledge Management, Pearson Prentice Hall.
- [2] 碩網資訊, 知識管理實務, 科技
- [3] 張吉成, 周談輝, 知識管理與創新, 全華
- [4] HTTP Session Management, Jan Newmarch,
<http://jan.netcomp.monash.edu.au/ecommerce/session.html>
- [5] Session Identification URI, W3C Working Draft WD-session-id-960221,
<http://www.w3.org/TR/WD-session-id-960221.html>
- [6] RFC 3986 – Uniform Resource Identifier (URI): Generic Syntax
- [7] RFC 2616 – Hypertext Transfer Protocol – HTTP/1.1
- [8] RFC 2109 – HTTP State Management Mechanism
- [9] RFC 959 – File Transfer Protocol
- [10] State in Web application design, David Orchard , BEA Systems, Inc. ,
<http://www.w3.org/2001/tag/doc/state.html>
- [11] Web Scraper Plus+ <http://www.velocityscape.com/>
- [12] Teleport Ultra <http://www.tenmax.com/teleport/ultra/home.htm>
- [13] Heritrix <http://crawler.archive.org/>
- [14] JoBo <http://www.matuschek.net/software/job/>
- [15] GNU wget <http://www.gnu.org/software/wget/wget.html>
- [16] OpenWebSpider <http://www.gnu.org/software/wget/wget.html>
- [17] Where Spider <http://wherespider.sourceforge.net/>
- [18] PhoPicking <http://www.phopicking.com/xpicweb/home/home.html>
- [19] Google Mini <http://www.google.com/enterprise/mini/index.html>
- [20] D. Knuth, J. Morris, and V. Pratt. Fast Pattern Matching in Strings. SIAM J. Computing,

6(2):323--350, 1977.

- [21] R.S. Boyer, J.S. Moore: A Fast String Searching Algorithm. Communications of the ACM, 20, 10, 762-772, 1977
- [22] James W. Cooper, The Design Patterns Java Companion, Addison-Wesley, 1998
- [23] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, Introduction to Algorithm, Second Edition, The MIT Press, 2001
- [24] Thierry Lecroq, Boyer-Moore Algorithm,
<http://www-igm.univ-mlv.fr/~lecroq/string/node14.html>
- [25] Thierry Lecroq, Kunth-Morris-Pratt Algorithm,
<http://www-igm.univ-mlv.fr/~lecroq/string/node8.html>
- [26] ICS 161: Design and Analysis of Algorithms Lecture notes for February 27, 1996
<http://www.ics.uci.edu/~eppstein/161/960227.html>
- [27] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, p293-p304 (1995)
- [28] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, Design Patterns, Addison-Wesley (1995)