CRAWLING SECOND HAND REAL ESTATE PROPERTIES IN BANGKOK

MR. TUNTIKORN ASAWAYUTHAPOL

MISS WILAI YAMCHUEN

MR. SONTHIPHONG TECHAKULTHAWORN

A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIRMENT FOR DEGREE OF BACHELOR OF SCIENCE

IN COMPUTER SCIENCE

FACULTY OF SCIENCE

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

ACADEMIC YEAR 2009

PROJECT TITLE:	Crawling Second Hand Real Estate Properties In Bangkok	
PERSONAL NAME:	Mr. Tuntikorn Asawayuthapol	49050378
	Miss Wilai Yamchuen	49050404
	Mr. Sonthiphong Techakulthaworn	49050411
DEGREE:	Bachelor's Degree of Science	
MAJOR PROGRAM:	Computer Science (International Programme)	
ACADEMIC YEAR	2009	
ADVISOR:	Mr. Suntana Oudomying	

ABSTRACT

We propose an engine which collects the information of the second hand real estate in Bangkok. Our application is based on the idea of the web crawler which we use java language to develop the application for crawling to collect information from web sites as much as possible as well as validating the accuracy of information about second hand real estate.

Acknowledgement

On the behalf of the author, we would like to express our thankfulness to all those who gave us the great suggestion and full support throughout the project.

With the deep honor, we wish to pay our gratitude to Mr. Suntana Oudomying, an advisor who dedicates his time for giving us recommendation as well as the possibility to complete this work.

Furthermore, the concerning of our family and friends can force us to pay attention to this special project.

Lastly, we would like to thank you all of the professor of the King Monkut's Institute of Technology Ladkrabang, Faculty of Science which give us valuable knowledge.

Mr. Tuntikorn Asawayuthapol

Miss Wilai Yamchuen

Mr. Sonthiphong Techakulthaworn

Table of Contents

Table of Contents (cont.)

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Rational and review literature

Searching second hand real estate in Bangkok via the internet offers advantages for users. For example, user is able to quickly narrow down listed properties to his desire in term of type, price, location, etc. User is also able to see images of the property. The internet is a powerful tool for creating list of interesting properties.

However, second hand real estate web sites in Bangkok are lack of rich content. Most sites offer only properties listed by a handful of their users. Buying a real estate property demands a careful search due to its price. We believe that a successful site for this business should accommodate as many properties as possible to draw users. One way of achieving that is to crawl and collect data from other sites. Such centralize model would also serve the analysis purposes other than merchandize such as hot areas.

1.2 Objective

Our application will create database for second hand real estate application and collect second hand real estate information from web sites in Bangkok. To do that involves

1. Obtaining URLs which contain such data as well as managing the queue containing the URLs.

2. Making database connection to heterogeneous databases. We use scrap and XML

3. Clean the content of various schemas as well as managing those data.

1.3 Scope of study

In this project, we will develop the web crawler application that can automatically collect the real estate information from internet. Our focus is on the crawler part; therefore, we will implement a very simple consolidation module for extension later, provided we have enough time.

1.4 Research Methodology

1. Study about how web crawler works.

2. Study the tools that we use to develop the application.

   - Java

   - My SQL

   - j2se

   - WebHavest

3. Design the database schema and the system.

4. Development Phrase.

5. Analyze the information obtained.

6. Make a conclusion.

Chapter 2

Background

This chapter focuses on basic knowledge and software required in this special project.

2.1 WebCrawler

WebCrawler is a Web service that assists users in their Web navigation by automating the task of link traversal, creating a searchable index of the web, and fulfilling searchers' queries from the index.

Conceptually, WebCrawler is a node in the Web graph that contains links to many sites on the net, shortening the path between users and their destinations. Such a simplification of the Web experience is important for several reasons. First, WebCrawler saves users time when they search instead of trying to guess at a path of links from page to page. Often, a user will see no obvious connection between the page he is viewing and the page he seeks. For example, he may be viewing a page on one topic and desire a page on a completely different topic, one that is not linked from his current location. In such cases, by jumping to WebCrawler — either using its address or a button on the browser — the searcher can easily locate his destination page. Such time savings is especially important given the increase in the size and scope of the Web: between 1994 and 2000, the Web grew in size by four orders of magnitude [MIDS 2000]. Second, WebCrawler's simplification of the Web experience makes the Web a more friendly and useful tool. Navigating the Web by using keyword searches is often more intuitive than trying to use a Uniform Resource Locator (URL) to identify a Web page directly. If users have a good experience, they are more likely to

continue to use the Web, and such repeat usage will continue to fuel the growth of the medium. Arguably, search engines like WebCrawler have contributed to the continued simplicity and growth of the Web. Finally, WebCrawler is useful because it can provide some context for a searcher's particular query by issuing a well-formed query; a searcher can find the breadth of information about that particular topic and can use that information to further refine his goal. Searchers frequently issue a broad query which they refine as they learn more about their intended subject.

Crawling is the means by which WebCrawler collects pages from the Web. The end result of crawling is a collection of Web pages at a central location. Given the continuous expansion of the Web, this crawled collection is guaranteed to be a subset of the Web and, indeed, it may be far smaller than the total size of the Web. By design, WebCrawler aims for a small, manageable collection that is representative of the entire Web. Crawling proceeds in much the same fashion as a person would if he were trying to build a collection of Web pages manually: WebCrawler starts with a single URL, downloads that page, retrieves the links from that page to others, and repeats the process with each of those pages. Before long, WebCrawler discovers links to most of the pages on the Web, although it takes some time to actually visit each of those pages.

In algorithmic terms, WebCrawler is performing a traversal of the Web graph using a modified breadth-first approach. As pages are retrieved from the Web, WebCrawler extracts the links for further crawling and feeds the contents of the page to the indexer. The indexer takes the full-text of the page and incorporates it into the full-text index. Indexing is a fairly straightforward process, and can easily keep up with the crawling when given the appropriate resources.

At first, indexing ran in step with crawling: each document was added to the index as it was retrieved from the Web. Later, indexing became a batch process that ran nightly. Inherent in the crawling process is the strategy for constructing the collection: which documents are worth indexing? which are not? At first, the entire Web would seem to be worth indexing. However, serving queries against such a large index is expensive and does not necessarily result in the best results all the time. Because of the large size of the Web, approximately one billion pages at the end of 1999, implementing the crawler efficiently is difficult. Furthermore, the Web's authors make the task of crawling even more difficult by constantly adding, deleting, modifying, and moving documents.

2.2 Web crawler vs. Search Engine

A web crawler is a program that downloads and stores Web pages, often for a Web search engine. Roughly, a crawler starts off by placing an initial set of URLs, in a queue, where all URLs to be retrieved are kept and prioritized. From this queue, the crawler gets a URL (in some order), downloads the page, extracts any URLs in the downloaded page, and puts the new URLs in the queue. This process is repeated until the crawler decides to stop. Collected pages are later used for other applications, such as a Web search engine or a Web cache. The most important measure for a search engine is the search performance, quality of the results and ability to crawl, and index the web efficiently. The primary goal is to provide high quality search results over a rapidly growing World Wide Web. Some of the efficient and recommended search engines are Google, and Yahoo, which share some common features and are standardized to some extent.

Web crawlers are an essential component to search engines; running a web crawler is a challenging task. There are tricky performance and reliability issues and even more importantly, there are social issues.

Crawling is the most fragile application since it involves interacting with hundreds of thousands of web servers and various name servers, which are all beyond the control of the system. Web crawling speed is governed not only by the speed of one's own Internet connection, but also by the speed of the sites that are to be crawled. Especially if one is a crawling site from multiple servers, the total crawling time can be significantly reduced, if many downloads are done in parallel. Despite the numerous applications for Web crawlers, at the core they are all fundamentally the same. Following is the process by which Web crawlers work.

1. Download the Web page.
2. Parse through the downloaded page and retrieve all the links.
3. For each link retrieved, repeat the process.

The Web crawler can be used for crawling through a whole site on the Internet and Intranet. You specify a start URL and the Crawler follows all links found in that HTML page. This usually leads to more links, which will be followed again, and so on. A site can be seen as a tree-structure, the root is the start-URL; all links in that root-HTML-page are direct sons of the root. Subsequent links are then sons of the previous sons.

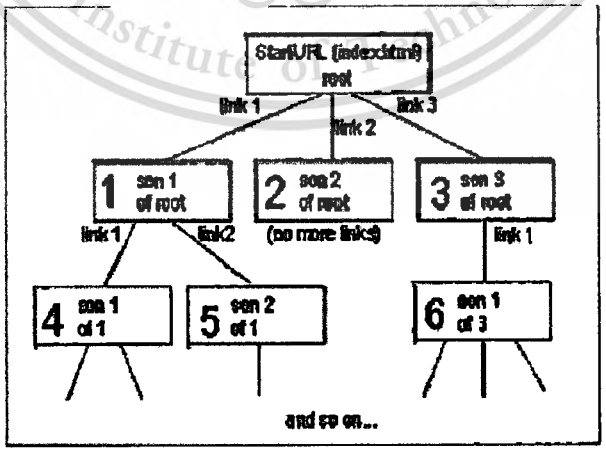


Figure 2.1 Shows the tree-structure of html pages of a web site.

A single URL Server serves lists of URLs to a number of crawlers. Web crawler starts by parsing a specified web page, noting any hypertext links on that page that point to other web pages. They then parse those pages for new links, and so on, recursively. Web crawler software doesn't actually move around to different computers on the Internet, as viruses or intelligent agents do. Each crawler keeps roughly 300 connections open at once. This is necessary to retrieve web pages at a fast enough pace. A crawler resides on a single machine. The crawler simply sends HTTP requests for documents to other machines on the Internet, just as a web browser does when the user clicks on links. All the crawler really does is to automate the process of following links. Web crawling can be regarded as processing items in a queue. When the crawler visits a web page, it extracts links to other web pages. So the crawler puts these URLs at the end of a queue, and continues crawling to a URL that it removes from the front of the queue.

Search engine is a challenging task. Search engines index tens to hundreds of millions of web pages involving a comparable number of distinct terms. They answer tens of millions of queries every day. Despite the importance of large-scale search engines on the web, very little academic research has been conducted on them. Furthermore, due to rapid advance in technology and web proliferation, creating a web search engine today is very different from three years ago. There are differences in the ways various search engines work, but they all perform three basic tasks:

1. They search the Internet or select pieces of the Internet based on important words.
2. They keep an index of the words they find, and where they find them.
3. They allow users to look for words or combinations of words found in that index.

A search engine finds information for its database by accepting listings sent in by authors who want exposure, or by getting the information from their "web crawlers," "spiders," or "robots," programs that roam the Internet storing links to and information about each page they visit.

2.3 WebCrawler's Implementation

WebCrawler's implementation is from the perspective of a Web user, entirely on the Web; no client side elements outside the browser are necessary. The service is composed of two fundamental parts: crawling, the process of finding documents and constructing the index; and serving, the process of receiving queries from searchers and using the index to determine the relevant results. This process is illustrated schematically.

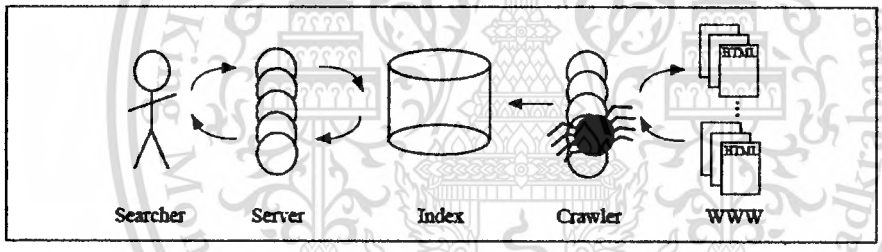


Figure 2.2 Web Crawler architecture.

Web Crawler's overall architecture. The Crawler retrieves and processes documents from the Web, creating an index that the server uses to answer queries.

Though WebCrawler's implementation has evolved over time, these basic components have remained the same. Only their scope, detail, and relative timing have changed. We briefly describe an early implementation of WebCrawler. We can assort Spider Queue into 4 categories.

We can assort into 4 categories.

- Waiting Queue

When we check the URLs that was collect from the web page and add them into waiting queue for parsing (Collect Link).

- Running Queue

  In process URL, we will bring out the URL from Waiting Queue. Then add that URL into Running Queue to showing this process URL is running.

- Complete Queue

  When we already visit the URL, we will remove it from Waiting Queue. After that we will add URL into complete instead for showing that URL was already visited and collected. After this we will not come back to check it again.

- Error Queue

  When the URL that we are visiting not available or cannot connect, it will remove the URL from Waiting Queue and we will add URL into Error Queue instead. For show that this URL has been visiting and not come back to check it again.

2.4 Web-Harvest

Web-Harvest is inspired by practical need for having right data at the right time. And very often, the Web is the only source that publicly provides wanted information.

The main goal behind Web-Harvest is to empower the usage of already existing extraction technologies. Its purpose is not to propose a new method, but to provide a way to easily use and combine the existing ones. Web-Harvest offers the set of processors for data handling and control flow. Each processor can be regarded as a function - it has zero or more input parameters and gives a result after execution. Processors

could be combined in a pipeline, making the chain of execution. For easier manipulation and data reuse Web-Harvest provides variable context where named variables are stored. Figure 2.4 describes one pipeline execution

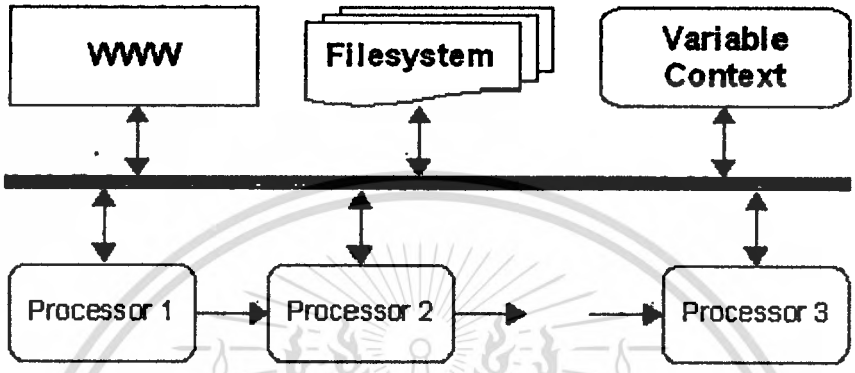


Figure 2.4 Basic concept of web-harvest

The result of extraction could be available in files created during execution or from the variable context if Web-Harvest is programmatically used.

2.5 MySQL

MySQL is open source of relational database management system. MySQL is the most commonly used for Web applications as well as embedded applications. It has become a popular alternative to proprietary database systems because of its speed and reliability.

MySQL distribution is used for an open source. This is all MySQL users can use and customize the functionality as needed. They are MySQL Enterprise, MySQL Cluster, MySQL Embedded and MySQL Community (opensource version). MySQL can be downloaded from the Internet and used without any cost.

2.6 NetBeans IDE

NetBeans is an integrated development environment (IDE) for developing applications. The Tools help programmers develop applications using Java, JavaScript, PHP, python, ruby, Groovy, C and C + +.

NetBeans project started in 1997 under the guidance of students of mathematics and physics at Charles University, Prague.Roman Stanek later formed a company around the project and produced commercial versions of the NetBeans IDE until it was bought by Sun Microsystems in 1999. Sun open sourced NetBeans IDE in June of the following year. NetBeans community has since continued to grow as individuals and companies using and contributing to the project.

Currently developers can use other languages (C / C + +, ruby, UML, SOA, Web Application, Java You, Mobility (Java ME), Java FX, Java Script, etc.) in developing a program to install the "Add. -On "from the netbeans website or center update.

Current version is NetBeans IDE 6.5, which launched in November 2008. NetBeans has won several products. Developer.com Year Awards 2009: Development Tools Development Utilities, Wireless / Mobile, Java tools and Open Source.

2.7 Extensible Markup Language (XML)

The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages. Its primary purpose is to help information systems share structured data, particularly via the Internet, and it is used both to encode documents and to serialize data. It is a related W3C (World Wide Web Consortium) standard, and is the basis for standard information interchange. It is classified as an extensible

language because it allows its users to define their own elements. XML is a generic framework for storing any amount of text or any data whose structure can be represented as a tree. The only indispensable syntactical requirement is that the document has exactly one root element (alternatively called the document element). This means that the text must be enclosed between a root start-tag and a corresponding end-tag.

2.8 Multithread

In computer science, a thread of execution results from a fork of a computer program into two or more concurrently running tasks. The implementation of threads and processes differs from one operating system to another, but in most cases, a thread is contained inside a process. Multiple threads can exist within the same process and share resources such as memory, while different processes do not share these resources.

On a single processor, multithreading generally occurs by time division multiplexing .The processor switches between different threads. This context switching generally happens frequently enough that the user perceives the threads or tasks as running at the same time. On a multiprocessor or multi-core system, the threads or tasks will generally run at the same time, with each processor or core running a particular thread or task. Support for threads in programming languages varies. A number of languages support multiple threads but do not allow them to execute at the same time.

Multithreading as a widespread programming and execution model allows multiple threads to exist within the context of a single process. These threads share the process' resources but are able to execute independently. The threaded programming model provides developers with a useful abstraction of concurrent

execution. However, perhaps the most interesting application of the technology is when it is applied to a single process to enable parallel execution on a multiprocessor system.

This advantage of a multithreaded program allows it to operate faster on computer systems that have multiple CPUs, CPUs with multiple cores, or across a cluster of machines because the threads of the program naturally lend themselves to truly concurrent execution. In such a case, the programmer needs to be careful to avoid race conditions, and other non-intuitive behaviors. In order for data to be correctly manipulated, threads will often need to rendezvous in time in order to process the data in the correct order. Threads may also require atomic operations in order to prevent common data from being simultaneously modified, or read while in the process of being modified. Careless use of such primitives can lead to deadlocks.

Another advantage of multithreading, even for single CPU systems, is the ability for an application to remain responsive to input. In a single threaded program, if the main executions thread blocks on a long running task, the entire application can appear to freeze. By moving such long running tasks to a worker thread that runs concurrently with the main execution thread, it is possible for the application to remain responsive to user input while executing tasks in the background.

Chapter 3

System Overview

This chapter will cover design that we have in this project. These design cover system analysis, application design, and database design.

3.1 Systems analysis

3.1.1 Problem Description

Second hand real estate web sites in Bangkok are lack of rich content most sites offer only properties listed by a handful of their users.

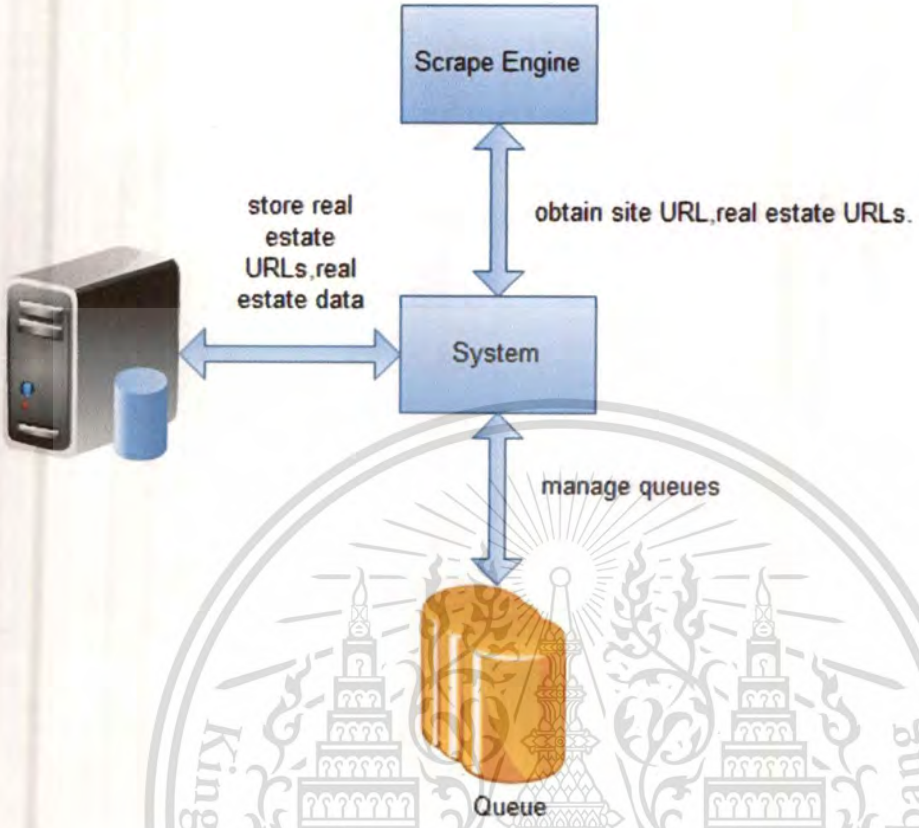
3.1.2 Solution

To develop the web crawler application that can automatically collect the real estate information from the internet.

3.2 System Requirement

3.2.1 System Architecture

The general idea of our application goes as follow. The system works by obtaining the URLs from Search Engines and collecting them into our database. Then we extract real estate information from stored URLs to collect real estate information on those pages into our database.



Figures 3.1 Overall System Design

Figure 3.1 shows the overall design of the application. The system sends URL of the initial page to scrape engine for scrape site. This process is repeated by collecting external links from scraped pages to extend the coverage of crawled sites. The stored URLs are immediately loaded from the database into a queue. Once the queue is not empty, the system needs to verify whether pages from URLs in the queue is scrap-able and migrates those scrap-able to another queue for scrapping real estate information. The process is done in parallel while collecting the URLs mentioned at the beginning of this paragraph. The scraped real estate URLs then are sent back to system and loaded in the last queue for collecting actual real estate information. System needs to manage queues for checking URLs that was collect from web page and add them into waiting queue for parsing then send back to system. System will store real estate URLs and real estate data into database.

The more detailed design is presented below.

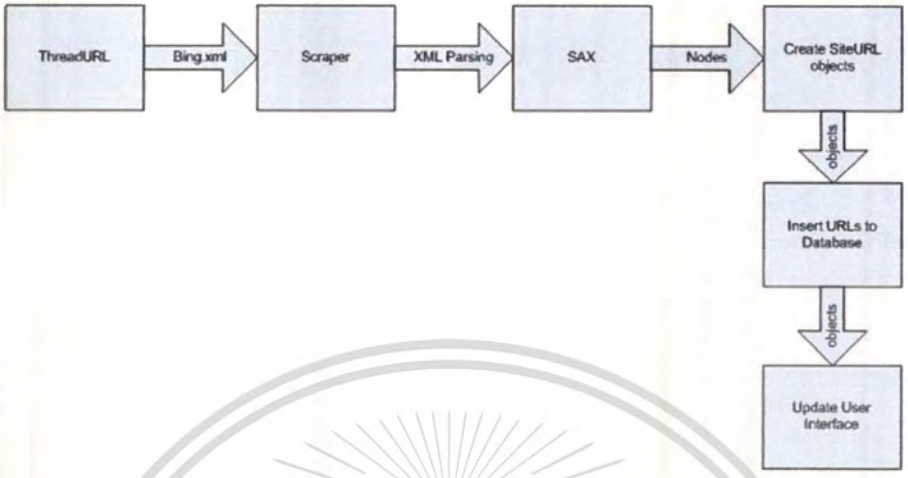


Figure 3.2 Thread URL process

From figure 3.2, the main thread activates thread url to call web-harvest API for scraping URLs of second hand real estate, after finished it will send URLs to queue1. Because some of these URLs may not be sites that contain second hand real estate, our application tries to select only ones that are and stores them in queue 2.

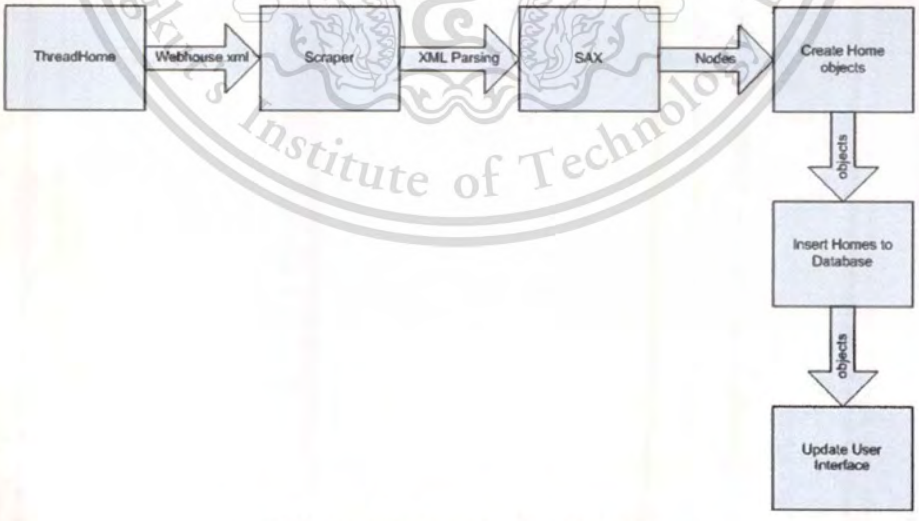


Figure 3.3 Thread Home process

From figure 3.3, legitimated URLs stored in queue 2 are scraped one by one for real estate URLs on those pages to be stored in queue 3. The system scrapes real estate information from queue 3 and stores them into database. Our issue here is that, because both parts of our application are independent, we implement our application with 2 independent threads.

3.2.2 Sequence Diagram

In this section, we present our application design with sequence diagram for formally explaining how it works. At the end of this section, we also take an opportunity to explain our design for maintaining data integrity of the application.

As mentioned in previous section, the queue 1, marked by label 1 in figure 3.4, is obtained by matching the structure of the input page to our xml configuration file. The configuration file let web harvest to obtain external links. The SAX is activated by web harvest implicitly.

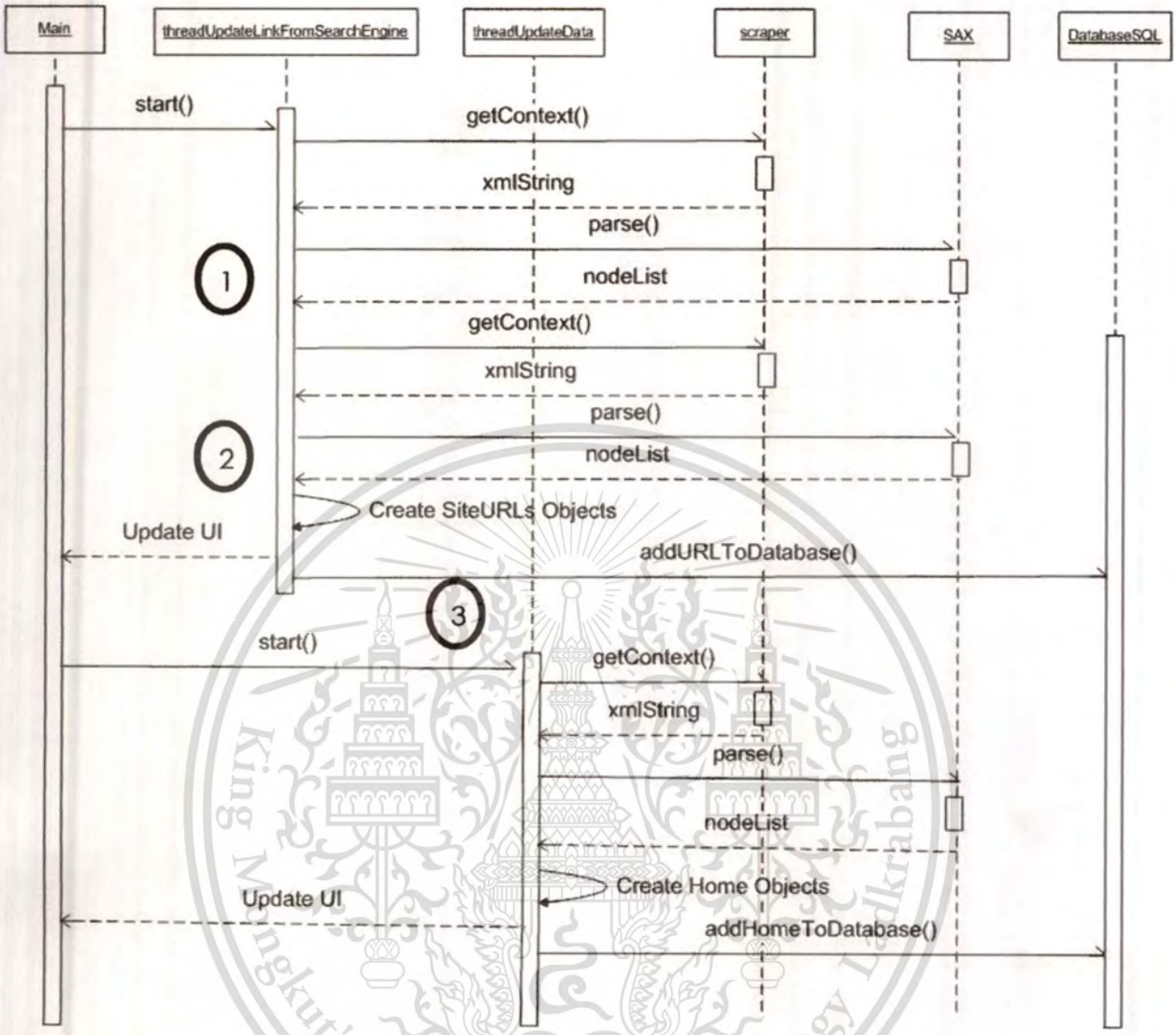


Figure 3.4 Sequence Diagram

The program filters only URLs which its web page structure is scrape-able by our predefined structure in another xml configuration file. Selected URLs are kept in Queue 2, labeled 2 in the same figure below. The same technique is used again for retrieving real estate information accompanied by the xml configuration file mentioned in this paragraph.

We finish this section by explaining our policy regarding maintaining the integrity of our real estate data. Our two concerns need to be managed are whether the real estate has been sold and whether any real estate from different sites is, in fact, the same. The following is how we manage each case.

- We flush the data belonged to every site we are scrapping. This way sold property will not be scraped and, as a result, will not be in our database.

- To handle redundant real estate information list on different sites, we depend on the real estate address stored in our database. Scraped real estate information is non-structured. Therefore, we use keywords to break the information for storing it into structured designed database schema. Any matched addresses are considered redundant. The system displays only a copy of the information.

3.3 XML configuration files

Figures 3.6 – 3.8 are used for explaining each part of the xml file in figure 3.5, which is required for scrapping URLs from Bing.com

```
<var-def name="startUrl">http://www.bing.com/search?q=%E0%B8%9A%E0%B9
    %89%E0%B8%B2%E0%B8%99%  E0%B8%A1%E0%B8%B7%E0%B8%AD%E0%B8%AA%E0%B
    8%AD%E0%B8%87&amp;go=&amp;form=QBRE&amp;filt=all</var-def>
  <var-def name="links">
      <xpath expression="//a[starts-with(@href, 'http://') ]/@href">
          </xpath>
      <body>
          <xquery>
              <xq-expression><![CDATA[

                  declare variable $articleUrl as xs:string external;

                let $id := $articleUrl

                   return
                        <link>
                              <url>{$articleUrl}</url>
                        </link>
              ]]></xq-expression>
          </xquery>
      </body>
```

Figure 3.5 bing.xml configuration file

From Figure 3.5, we will explain details in separate parts.

First we define the variable in var-def language that shown in Figure 3.6

```
<var-def name="startUrl">http://www.bing.com/search?q=%E0%B8%9A%E0%
B9%89%E0%B8%B2%E0%B8%99%E0%B8%A1%E0%B8%B7%E0%B8%AD%E0%B8%AA%E0%B8%A
D%E0%B8%87&amp;go=&amp;form=QBRE&amp;filt=all</var-def>
```

Figure 3.6 Started URLs (part of bing.xml)

Secondly, Uses an XPath language expression to stored URLs from tag "a" start from href with http:// that shown it Figure 3.7

```
<xpath expression="//a[starts-with(@href, 'http://') ]/@href">
        </xpath>
```

Figure 3.7 Find URLs (part of bing.xml)

Next, we use XQuery language express to query xml document and we use Xq-Expession language for print out URLs result in form of xml. That shown in Figure 3.8

```
<xquery>
    <xq-expression><![CDATA[

        declare variable $articleUrl as xs:string external;

        let $id := $articleUrl

           return
                <link>
                      <url>{$articleUrl}</url>
                </link>
   ]]></xq-expression>
</xquery>
```

Figure 3.8 Print URL in XML Form (part of bing.xml)

We explain how we extract information from web pages using figure 3.9 and 3.10,.

```
<var-def name="homes">
        <body>
            <xquery>
                <xq-expression><![CDATA[
                    let $detail := data($doc//div[@class="A_label"])
                    let $contact := data($doc//div[@class="BL_inborder clearfix"]/div[@class="E_label"])
                    return <home>
                                    <link>{$myUrl}{$articleUrl}</link>
                                    <detail>
                                    {data($detail)}
                                    </detail>
                                    <contact>
                                    {data($contact)}
                                    </contact>
                                </home>
                ]]></xq-expression>
            </xquery>
        </body>
 </var-def>
```

Figure 3.9 webhouse.xml configuration file

We use Xq-Express language for declare the variable detail and contact for scrape the Real estate Information from tag A_label, BL_inborder clearfix and E_label. Next, we return the information in form of xml that shown in Figure 3.10

```
<xq-expression><![CDATA[
    let $detail := data($doc//div[@class="A_label"])
    let $contact := data($doc//div[@class="BL_inborder clearfix"]/div[@class="E_label"]
    return
                <home>
                    <link>{$myUrl}{$articleUrl}</link>
                    <detail>
                    {data($detail)}
                    </detail>
                    <contact>
                    {data($contact)}
                    </contact>
                </home>
]]></xq-expression>
```

Figure 3.10 Scrape Real Estate Information (part of webhouse.xml)

3.4 Database Design

We create two tables for storing real estate information and URLs from Search Engine.

3.4.1 Homes Table

Homes Table shown below in Table 3.2 is for storing real estate information data. Its structure information of real estate (ID, URLs, Address, Phone, Description).

Table 3.1 Homes Table

Attribute	Data type	Description	Example
HomeID	INTEGER	Home Code	
URLs	VARCHAR(200)	URLs of Homes	http://www.greathome.co.th/de_home_2home.php?id=81
Address	VARCHAR(1000)	Address of Homes	หมู่บ้าน/ซอย European Thai House ถนน พัทยาใต้ 87 เขตบางละมุง จังหวัดชลบุรี
Phone	VARCHAR(1000)	Phone number of Homes	029033668
Description	VARCHAR(500)	Description of Homes	European Thai House- Pattaya ประเภทบ้านเดี่ยว ราคา :4900000 บาท เนื้อที่ดิน 0-1-0 พื้นที่ใช้สอย 257 ตรม. จำนวนชั้น 1 ห้องนอน3 ห้องน้ำ 3

3.4.2 URLs Table

URLs Table shown below in Table 3.3 is for storing URLs from Search Engine. Its structure contains ID, URLs.

Table 3.2 URLs Table

Attribute	Data type	Description	Example
urlID	INTEGER	url Code	
URL	VARCHAR(200)	URLs from Search Engine	http://www.greathome.co.th

Chapter 4

Results and Discussion

4.1 System Operation

4.1.1 Main Page

Figure 4.1 shows layout of our main interface. Queue 1 box displays URLs obtained from external links. Queue2 contains verified URLs that the system determines they are scrape-able. Queue 3 contains real estate URLs listed on the processed site URL from Queue 2.

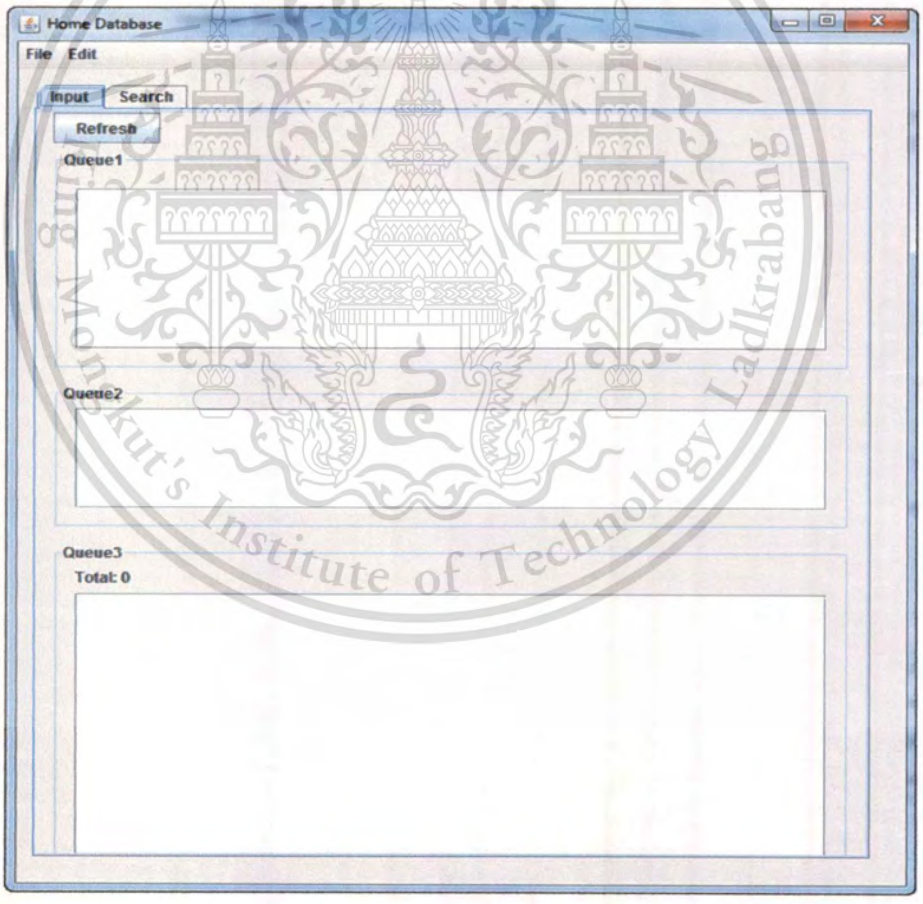


Figure 4.1 : Main page

4.1.2 Search page

If a user wants to search a keyword of second hand real estate, users can input their information that they want to find such as type of houses, area and location.

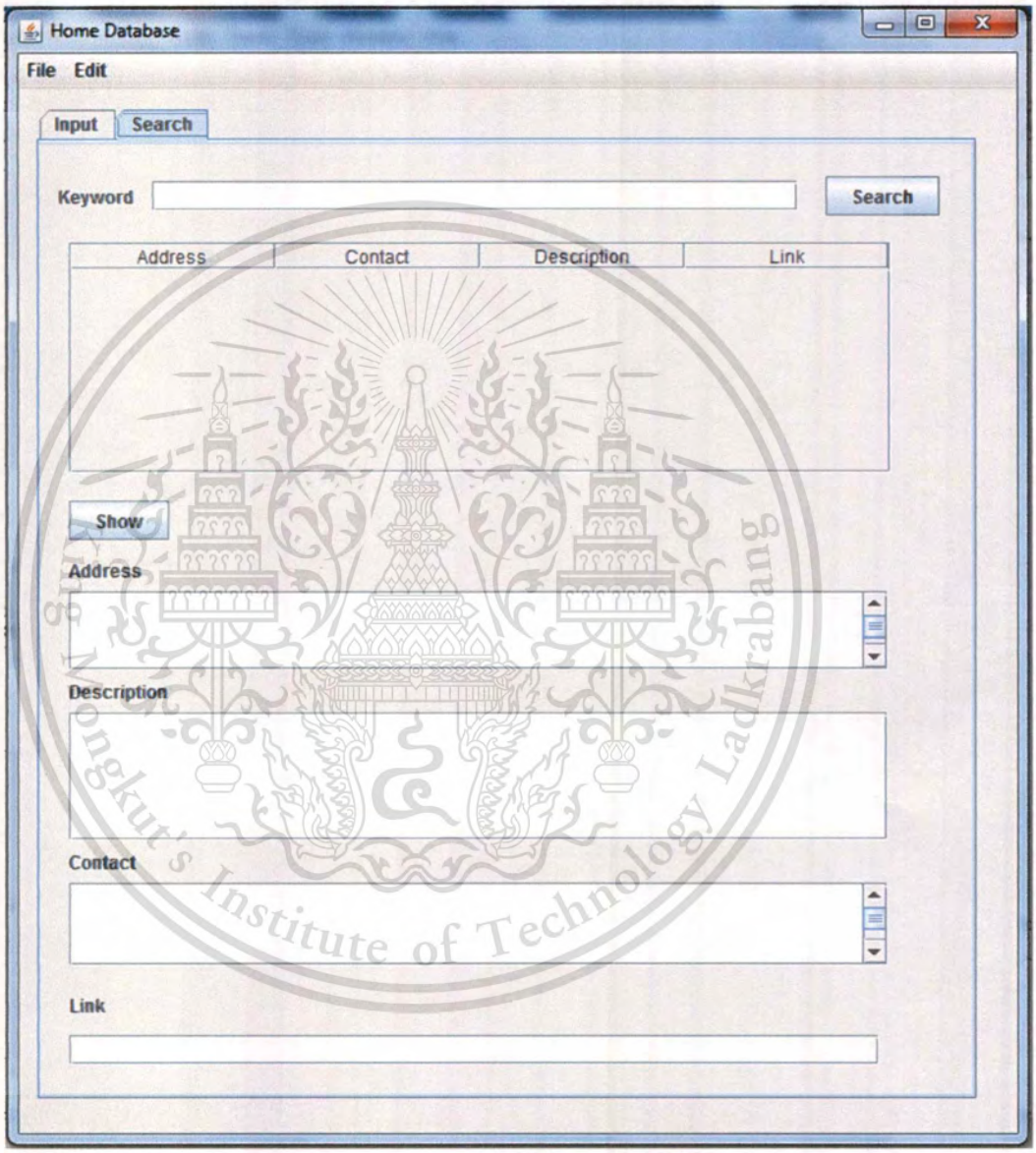


Figure 4.2 : Search page

4.2 Discussion

Our application have obtained about 15 sites in 30 minutes. The number of sites increases very slowly after that due to the lack of external link exchanged among URLs. On average, each site lists 20 pages of real estate properties where each page contains approximately 10 properties.

Figure 4.3(A) shows a screenshot of the application during program execution. The figure also shows an feature we have not mentioned – the refresh button. In input page, the refresh button is to refresh the information contained in each queue. Our current implementation suffers from displaying the information in those queue in real time.

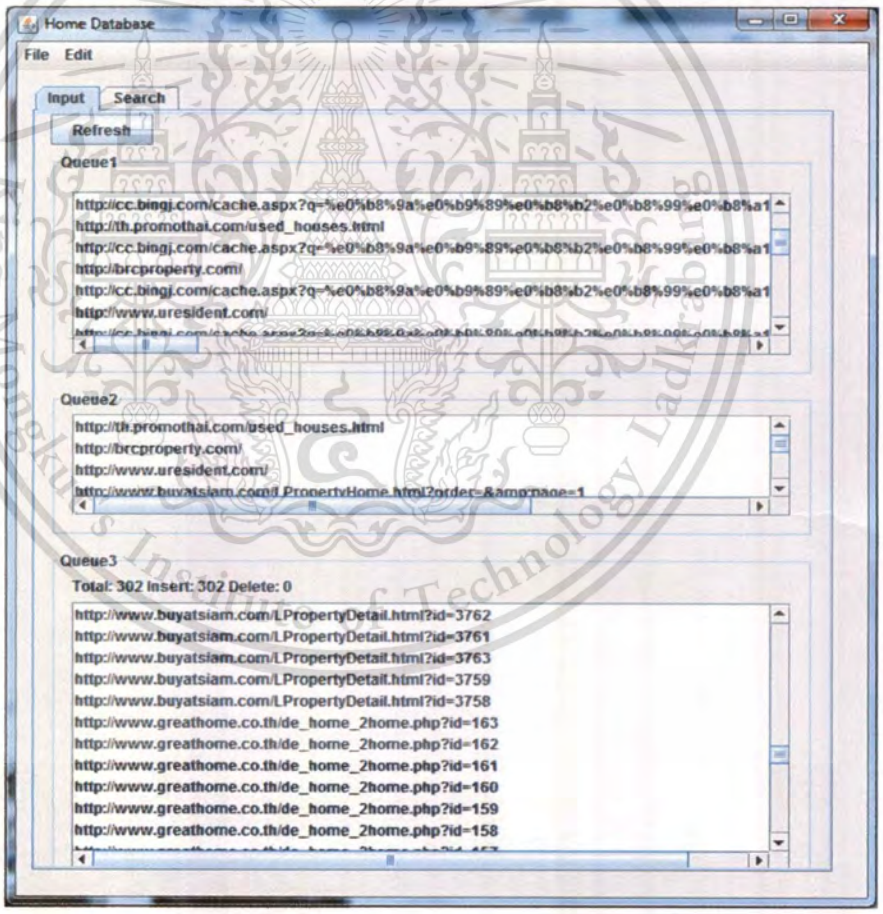


Figure 4.3(A) : Result page

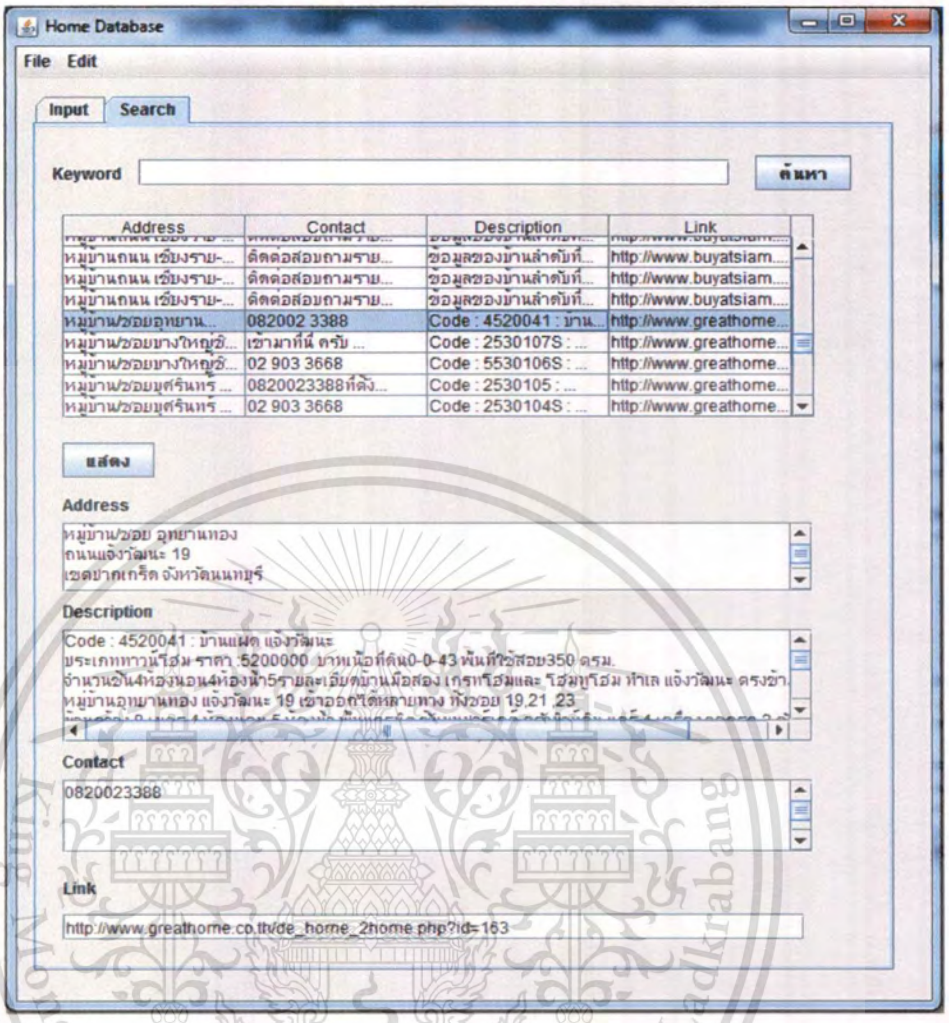


Figure 4.3(B) : Result page

From figure 4.3(B), if the user does not supply any keyword, the application displays all properties stored in the database. After that, the user can see the detail information of a property by pressing the show button.

Chapter 5

Conclusion and Recommendation

5.1 Conclusion

Our result shows that our database is a collection of second hand real estate information in Thailand from various sites. Policies and techniques for managing queues as well as consolidating data are described while we overviewed our system. The engine can be integrated to any second hand real estate business web application. Moreover we have implemented two user interfaces for using the information as a standalone application.

5.2 Recommendation

The following are areas where we believe our application can be improved.

1. Consolidating unstructured data and semi-structure data from a number of sites to structured data need to be more effective.
2. Our application can gather URLs only from Bing. This is because its scrapped content is text while other search engine's result pages are not.
3. Currently, our application can collect the data only from pages that match our defined structure. In reality, Pages from various sites are of different structure.

References

Brian Pinkerton. **WebCrawler: Finding What People Want**. [Online]. Available: thinkpink.com/bp/Thesis/Thesis.pdf.

How Search Engines Work. [Online]. Available: http://searchenginewatch.com/2168031

Thread. [Online]. Available: wapedia.mobi/en/Thread_(computer_science)

Web-harvest. [Online]. Available: http://web-harvest.sourceforge.net/.

Wikipedia. **Web scraping**. [Online]. Available: http://en.wikipedia.org/wiki/Web_scraping