ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

PUBLIC UNIVERSITY OF NAVARRA

Degree:

COMPUTER TECHNICAL ENGINEERING
(INDUSTRIALS AND TELECOMMUNICATIONS
ENGINEERING SCHOOL)

Title of the Diploma Thesis:

UPGRADE SYSTEM:
"Crawling Process: The Real Estate Case"

Student: Miren Arizaleta Arteaga

Tutor: Michalis Vazirgiannis

Athens, February 2011

INDEX

Chapter 1

INTRODUCTION

In this first chapter is going to be explained the purpose of the Diploma Thesis, detailing the process followed to do it.

1.1 Purpose of the Diploma Thesis

The aim of this Project is to put into gear again the system developing last year "Crawling Process: The Real Estate Case", changing or adding programs needed to make it works. This system consists basically in a compilation of information from five websites dedicated to selling and renting houses and plots, for the following realization of a statistic study.

For this, has been realized a study of the technics used in the system for capturing the data as well as its possible uses.

1.2 Description of the Diploma Thesis

Nowadays everybody uses or has used the web for different reasons. Everybody is aware that the web constitutes an architecture to access information and retrieves data in the form of interconnected documents which are distributed in millions of machines through the internet.

The most commonly used protocol for the retrieval of such documents is the http (Hypertext Transfer Protocol). When a user "demands" to retrieve some document or some information in the web, the use of this protocol is enough to do so. In this way the user can move through websites retrieving each piece of information or document which are useful to them at any given time. The question raised here is what happens in the case that one demands to retrieve millions and billions of documents or to retrieve a large volume of information either for future processing or for a simple reading. With the constant increase of the volume of data in the web as well as the daily renewal of the contents of the various websites, it is understandable that it is impossible for such a vast volume of data to be collected by the user, and therefore it is imperative the need to create mechanisms to automate this data retrieval procedure.

This is exactly the purpose of the present project. The design and implementation of a complete data collection system (Web Crawler), which is applicable in the field of Real Estate in Greece.

More specifically, the system implemented concerns the data (advertisements) retrieval by the five most popular property sites. The ultimate goal of this implementation is the collection of the advertisements from the above mentioned websites, so as to extract conclusions and statistical data for the overall picture of the property market in Greece. In addition to the above system and with the purpose to meet its demands, a database to store the retrieved data by the crawling process of the advertisements was designed and implemented.

Chapter 2

WEB CRAWLING

In this chapter is gonna be introduced the concept of "Web Crawling", explaining the most common techniques and policies and their possible uses.

2.1 Definition of Web Crawling

A Web Crawling (or Web Spider) is a program that inspects the World Wide Web pages in a methodical and automated way. One of the most common uses is to create a copy of all visited websites for the subsequent processing by a search engine that indexes pages providing a quick search system. Web spiders usually are bots (the most used type of these).

Web Spiders begins visiting a list of URLs, it identifies the hyperlinks on these pages and adds them to the list of URLs to visit on a recurring way under a certain set of rules. Normal operation is that it gives the program a group of addresses, the spider downloads these addresses, parses the pages and looks for links to new pages. Then downloads these new pages, analyzes their links, and so on.

The most common tasks of Web Spiders are:

- Create the index of a search engine.
- Analyze the links of a site to find broken links.
- Gather information from a certain type, such as product prices to compile a catalog.

To sum up, a Web Crawler can be defined by these ideas:

- A Web crawler is a computer program that browses the World Wide Web in a methodical, automated manner.

- A Web indexing program that builds an index by following hyperlinks continuously from web page to web page.

- A Web crawler is an automated program that accesses a web site and traverses through the site by following the links present on the pages.

2.2 Crawling Policies-Techniques

There are important characteristics of the Web that make crawling very difficult:

- its large volume,
- its fast rate of change, and
- dynamic page generation.

The large volume implies that the crawler can only download a fraction of the Web pages within a given time, so it needs to prioritize its downloads. The high rate of change implies that by the time the crawler is downloading the last pages from a site, it is very likely that new pages have been added to the site, or that pages have already been updated or even deleted.

The number of possible crawlable URLs being generated by server-side software has also made it difficult for web crawlers to avoid retrieving duplicate content. Endless combinations of HTTP GET (URL-based) parameters exist, of which only a small selection will actually return unique content. For example, a simple online photo gallery may offer three options to users, as specified through HTTP GET parameters in the URL. If there exist four ways to sort images, three choices of thumbnail size, two file formats, and an option to disable user-provided content, then the same set of content can be accessed with 48 different URLs, all of which may be linked on the site. This mathematical combination creates a problem for crawlers, as they must sort through endless combinations of relatively minor scripted changes in order to retrieve unique content.

As Edwards noted, "Given that the bandwidth for conducting crawls is neither infinite nor free, it is becoming essential to crawl the Web in not only a scalable, but efficient way, if some reasonable measure of quality or freshness is to be maintained." A crawler must carefully choose at each step which pages to visit next.

The behavior of a Web crawler is the outcome of a combination of policies:

- a *selection policy* that states which pages to download,
- a *re-visit policy* that states when to check for changes to the pages,
- a *politeness policy* that states how to avoid overloading Web sites, and
- a *parallelization policy* that states how to coordinate distributed Web crawlers.

Selection policy

Given the current size of the Web, even large search engines cover only a portion of the publicly-available part. A 2005 study showed that large-scale search engines index no more than 40%-70% of the indexable Web; a previous study by Dr. Steve Lawrence and Lee Giles showed that no search engine indexed more than 16% of the Web in 1999. As a crawler always downloads just a fraction of the Web pages, it is highly desirable that the downloaded fraction contains the most relevant pages and not just a random sample of the Web.

This requires a metric of importance for prioritizing Web pages. The importance of a page is a function of its intrinsic quality, its popularity in terms of links or visits, and even of its URL (the latter is the case of vertical search engines restricted to a single top-level domain, or search engines restricted to a fixed Web site). Designing a good selection policy has an added difficulty: it must work with partial information, as the complete set of Web pages is not known during crawling.

Cho made the first study on policies for crawling scheduling. Their data set was an 180,000-pages crawl from the stanford.edu domain, in which a crawling simulation was done with different strategies. The ordering metrics tested were breadth-first, backlink-count and partial Pagerank calculations. One of the conclusions was that if the crawler wants to download pages with high Pagerank early during the crawling process, then the partial Pagerank strategy is the better, followed by breadth-first and backlink-count. However, these results are for just a single domain. Cho also wrote his Ph.D. dissertation at Stanford on web crawling.

Najork and Wiener performed an actual crawl on 328 million pages, using breadth-first ordering. They found that a breadth-first crawl captures pages with high Pagerank early in the crawl (but they did not compare this strategy against other strategies). The explanation given by the authors for this result is that "the most important pages have many links to them from numerous hosts, and those links will be found early, regardless of on which host or page the crawl originates".

Abiteboul designed a crawling strategy based on an algorithm called OPIC (On-line Page Importance Computation). In OPIC, each page is given an initial sum of "cash" that is distributed equally among the pages it points to. It is similar to a Pagerank computation, but it is faster and is only done in one step. An OPIC-driven crawler downloads first the pages in the crawling frontier with higher amounts of "cash". Experiments were carried in a 100,000-pages synthetic graph with a power-law distribution of in-links. However, there was no comparison with other strategies nor experiments in the real Web.

Boldi used simulation on subsets of the Web of 40 million pages from the .it domain and 100 million pages from the WebBase crawl, testing breadth-first against depth-first, random ordering and an omniscient strategy. The comparison was based on how well PageRank computed on a partial crawl approximates the true PageRank value. Surprisingly, some visits that accumulate PageRank very quickly (most notably, breadth-first and the omniscent visit) provide very poor progressive approximations. Baeza-Yates used simulation on two subsets of the Web of 3 million pages from the .gr and .cl domain, testing several crawling strategies. They showed that both the OPIC strategy and a strategy that uses the length of the per-site queues are better than breadth-first crawling, and that it is also very effective to use a previous crawl, when it is available, to guide the current one.

Daneshpajouh designed a community based algorithm for discovering good seeds. Their method crawls web pages with high PageRank from different communities in less iteration in comparison with crawl starting from random seeds. One can extract good seed from a previously-crawled-Web graph using this new method. Using these seeds a new crawl can be very effective.

<u>Focused crawling</u>

The importance of a page for a crawler can also be expressed as a function of the similarity of a page to a given query. Web crawlers that attempt to download pages that are similar to each other are called focused crawler or topical crawlers. The concepts of topical and focused crawling were first introduced by Menczer and by Chakrabarti.

The main problem in focused crawling is that in the context of a Web crawler, we would like to be able to predict the similarity of the text of a given page to the query before actually downloading the page. A possible predictor is the anchor text of links; this was the approach taken by Pinkerton in a crawler developed in the early days of the Web. Diligenti propose to use the complete content of the pages already visited to infer the similarity between the driving query and the pages that have not been visited yet. The performance of a focused crawling depends mostly on the richness of links in the specific topic being searched, and a focused crawling usually relies on a general Web search engine for providing starting points.

URL normalization

Crawlers usually perform some type of URL normalization in order to avoid crawling the same resource more than once. The term URL normalization, also called URL canonicalization, refers to the process of modifying and standardizing a URL in a consistent manner. There are several types of normalization that may be performed including conversion of URLs to lowercase, removal of "." and ".." segments, and adding trailing slashes to the non-empty path component.

Path-ascending crawling

Some crawlers intend to download as many resources as possible from a particular web site. So path-ascending crawler was introduced that would ascend to every path in each URL that it intends to crawl. For example, when given a seed URL of http://llama.org/hamster/monkey/page.html, it will attempt to crawl /hamster/monkey/, /hamster/, and /. Cothey found that a path-ascending crawler was very effective in finding isolated resources, or resources for which no inbound link would have been found in regular crawling.

Many path-ascending crawlers are also known as Web harvesting software, because they're used to "harvest" or collect all the content from a specific page or host.

Re-visit policy

The Web has a very dynamic nature, and crawling a fraction of the Web can take weeks or months. By the time a Web crawler has finished its crawl, many events could have happened, including creations, updates and deletions.

From the search engine's point of view, there is a cost associated with not detecting an event, and thus having an outdated copy of a resource. The most-used cost functions are freshness and age.

Freshness: This is a binary measure that indicates whether the local copy is accurate or not. The freshness of a page p in the repository at time t is defined as:

$$F_p(t) = \begin{cases} 1 & \text{if } p \text{ is equal to the local copy at time } t \\ 0 & \text{otherwise} \end{cases}$$

Age: This is a measure that indicates how outdated the local copy is. The age of a page p in the repository, at time t is defined as:

$$A_p(t) = \begin{cases} 0 & \text{if } p \text{ is not modified at time } t \\ t - \text{modification time of } p & \text{otherwise} \end{cases}$$

Coffman worked with a definition of the objective of a Web crawler that is equivalent to freshness, but use a different wording: they propose that a crawler must minimize the fraction of time pages remain outdated. They also noted that the problem of Web crawling can be modeled as a multiple-queue, single-server polling system, on which the Web crawler is the server and the Web sites are the queues. Page modifications are the arrival of the customers, and switch-over times are the interval between page accesses to a single Web site. Under this model, mean waiting time for a customer in the polling system is equivalent to the average age for the Web crawler.

The objective of the crawler is to keep the average freshness of pages in its collection as high as possible, or to keep the average age of pages as low as possible. These objectives are not equivalent: in the first case, the crawler is just concerned with how many pages are out-dated, while in the second case, the crawler is concerned with how old the local copies of pages are.

Two simple re-visiting policies were studied by Cho and Garcia-Molina:
Uniform policy: This involves re-visiting all pages in the collection with the same frequency, regardless of their rates of change.

Proportional policy: This involves re-visiting more often the pages that change more frequently. The visiting frequency is directly proportional to the (estimated) change frequency.

(In both cases, the repeated crawling order of pages can be done either in a random or a fixed order.)

Cho and Garcia-Molina proved the surprising result that, in terms of average freshness, the uniform policy outperforms the proportional policy in both a simulated Web and a real Web crawl. The explanation for this result comes from the fact that, when a page changes too often, the crawler will waste time by trying to re-crawl it too fast and still will not be able to keep its copy of the page fresh.

To improve freshness, the crawler should penalize the elements that change too often. The optimal re-visiting policy is neither the uniform policy nor the proportional policy. The optimal method for keeping average freshness high includes ignoring the pages that change too often, and the optimal for keeping average age low is to use access frequencies that monotonically (and sub-linearly) increase with the rate of change of each page. In both cases, the optimal is closer to the uniform policy than to the proportional policy: as Coffman note, "in order to minimize the expected obsolescence time, the accesses to any particular page should be kept as evenly spaced as possible". Explicit formulas for the re-visit policy are not attainable in general, but they are obtained numerically, as they depend on the distribution of page changes. Cho and Garcia-Molina show that the exponential distribution is a good fit for describing page changes, while Ipeirotis show how to use statistical tools to discover parameters that affect this distribution. Note that the re-visiting policies considered here regard all pages as homogeneous in terms of quality ("all pages on the Web are worth the same"), something that is not a realistic scenario, so further information about the Web page quality should be included to achieve a better crawling policy.

Politeness policy

Crawlers can retrieve data much quicker and in greater depth than human searchers, so they can have a crippling impact on the performance of a site. Needless to say, if a single crawler is performing multiple requests per second and/or downloading large files, a server would have a hard time keeping up with requests from multiple crawlers.

As noted by Koster, the use of Web crawlers is useful for a number of tasks, but comes with a price for the general community. The costs of using Web crawlers include:
- network resources, as crawlers require considerable bandwidth and operate with a high degree of parallelism during a long period of time;
- server overload, especially if the frequency of accesses to a given server is too high;
- poorly-written crawlers, which can crash servers or routers, or which download pages they cannot handle; and
- personal crawlers that, if deployed by too many users, can disrupt networks and Web servers.

A partial solution to these problems is the robots exclusion protocol, also known as the robots.txt protocol that is a standard for administrators to indicate which parts of their Web servers should not be accessed by crawlers. This standard does not include a suggestion for the interval of visits to the same server, even though this interval is the most effective way of avoiding server overload. Recently commercial search engines like Ask Jeeves, MSN and Yahoo are able to use an extra "Crawl-delay:" parameter in the robots.txt file to indicate the number of seconds to delay between requests.

The first proposal for the interval between connections was given in and was 60 seconds. However, if pages were downloaded at this rate from a website with more than 100,000 pages over a perfect connection with zero latency and infinite bandwidth, it would take more than 2 months to download only that entire Web site; also, only a fraction of the resources from that Web server would be used. This does not seem acceptable.

Cho uses 10 seconds as an interval for accesses, and the WIRE crawler uses 15 seconds as the default. The MercatorWeb crawler follows an adaptive politeness policy: if it took t seconds to download a document from a given server, the crawler waits for 10t seconds before downloading the next page.

Anecdotal evidence from access logs shows that access intervals from known crawlers vary between 20 seconds and 3–4 minutes. It is worth noticing that even when being very polite, and taking all the safeguards to avoid overloading Web servers, some complaints from Web server administrators are received. Brin and Page note that: "... running a crawler which connects to more than half a million servers (...) generates a fair amount of e-mail and phone calls. Because of the vast number of people coming on line, there are always those who do not know what a crawler is, because this is the first one they have seen."

Parallelization policy

A parallel crawler is a crawler that runs multiple processes in parallel. The goal is to maximize the download rate while minimizing the overhead from parallelization and to avoid repeated downloads of the same page. To avoid downloading the same page more than once, the crawling system requires a policy for assigning the new URLs discovered during the crawling process, as the same URL can be found by two different crawling processes.

2.3 Web Crawler Architectures

A crawler must not only have a good crawling strategy, as noted in the previous section, but it should also have a highly optimized architecture.

Shkapenyuk and Suel noted that: "While it is fairly easy to build a slow crawler that downloads a few pages per second for a short period of time, building a high-performance system that can download hundreds of millions of pages over several weeks presents a number of challenges in system design, I/O and network efficiency, and robustness and manageability."

Web crawlers are a central part of search engines, and details on their algorithms and architecture are kept as business secrets. When crawler designs are published, there is often an important lack of detail that prevents others from reproducing the work. There are also emerging concerns about "search engine spamming", which prevent major search engines from publishing their ranking algorithms.

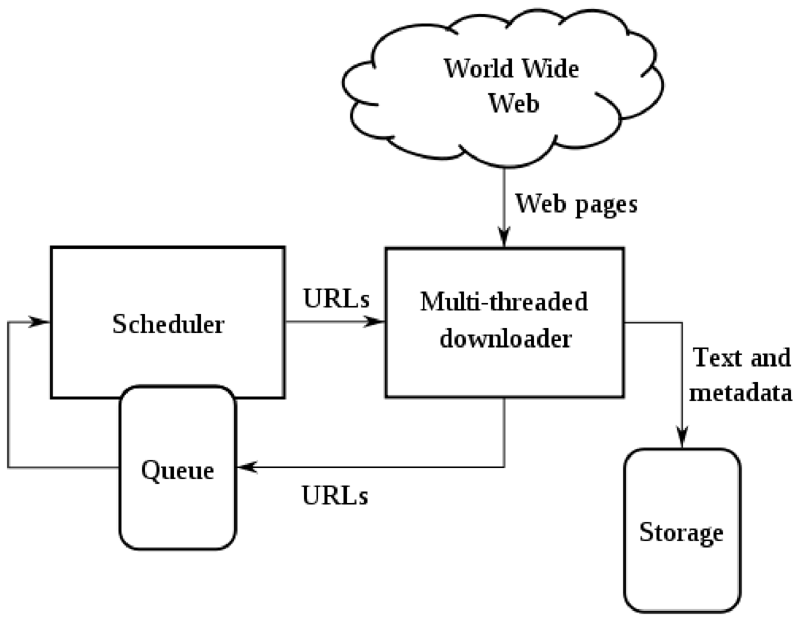


Figure 2.1: High-level architecture of a standard Web crawler

Chapter 3

SYSTEM DESCRIPTION

This chapter consists in a deep vision of the system developed for this project. All that appears in this chapter is based in the original project "*Crawling Process: The Real Estate Case*".

3.1 General description of the System

The Web Crawler developed in this Project is shown by the following diagram, where appear all the parts of the structure.

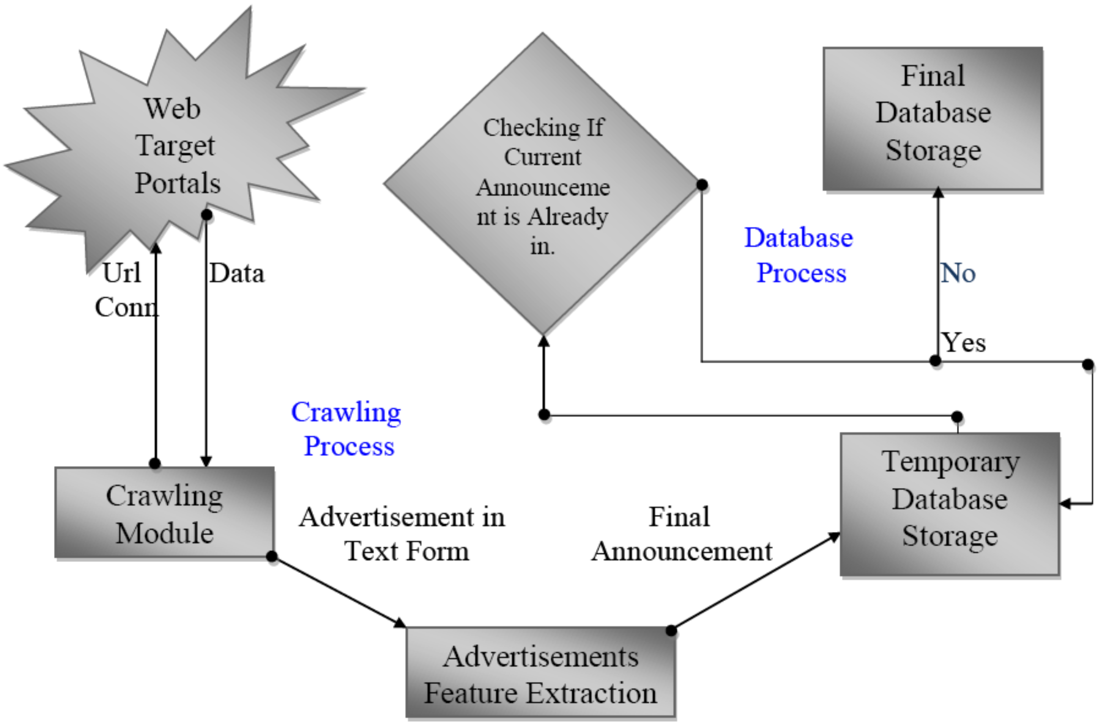

Web Target Portals
Url Conn
Data
Checking If Current Announcement is Already in.
Final Database Storage
Database Process
No
Yes
Crawling Process
Crawling Module
Advertisement in Text Form
Final Announcement
Temporary Database Storage
Advertisements Feature Extraction

Figure 3.1 Structure of the Web Crawler

As shown in the previous diagram, the types of processes that implement the system are twofold: the process of data collection (Crawling Process) and transport and storage process in the final database (Database Process). Let's analyze each one separately emphasizing the important points and problems encountered in each phase.

3.2 Crawling Process

This process performs the data collection web pages, and export of key features for subsequent introduction into the database. There are two units that carry out this process: The base unit Crawling (Crawling Module), and the extraction unit of the nature of advertising (Advertisements Feature Extraction module). The ads have been selected to be exported are those related to the sale and rental of residential and land.

3.2.1 Crawling Module

This unit performs the data collection process (Crawling Process). The operation is as follows:

- First of all, insert the URL of the target portal, i.e., the sites that we are interested to collect data.
- Then, knowing the structure of each one separately, are connected to each site, and source code is exported from each web page.
- After recovery of the source, then using a user-specific query (Focused Crawling technique), we focus on a specific piece of information to be retrieved. At this point the information has been recovered from the final announcement, which may be a new link that leads to the discovery of one or more final announcements. In the first case we take the ad or the final ads. In the second case, after making an extract from a new connection, reconnect to the new URL now and gives a new user query to retrieve new information that is relevant. The process is repeated until we have taken out all the ads that interest us.
- Let's see a concrete example of this procedure to retrieve data from the site (URL: http://www.xe.gr)
- This example shows the contents (index) generated during the process of collecting the ads until the final movement. This directory is only a path of links followed to reach the final of the ads.

Note: For purposes of this example, a prefix is used, which is used for the construction of the new URL.

Pattern= http://www.xe.gr/pages/

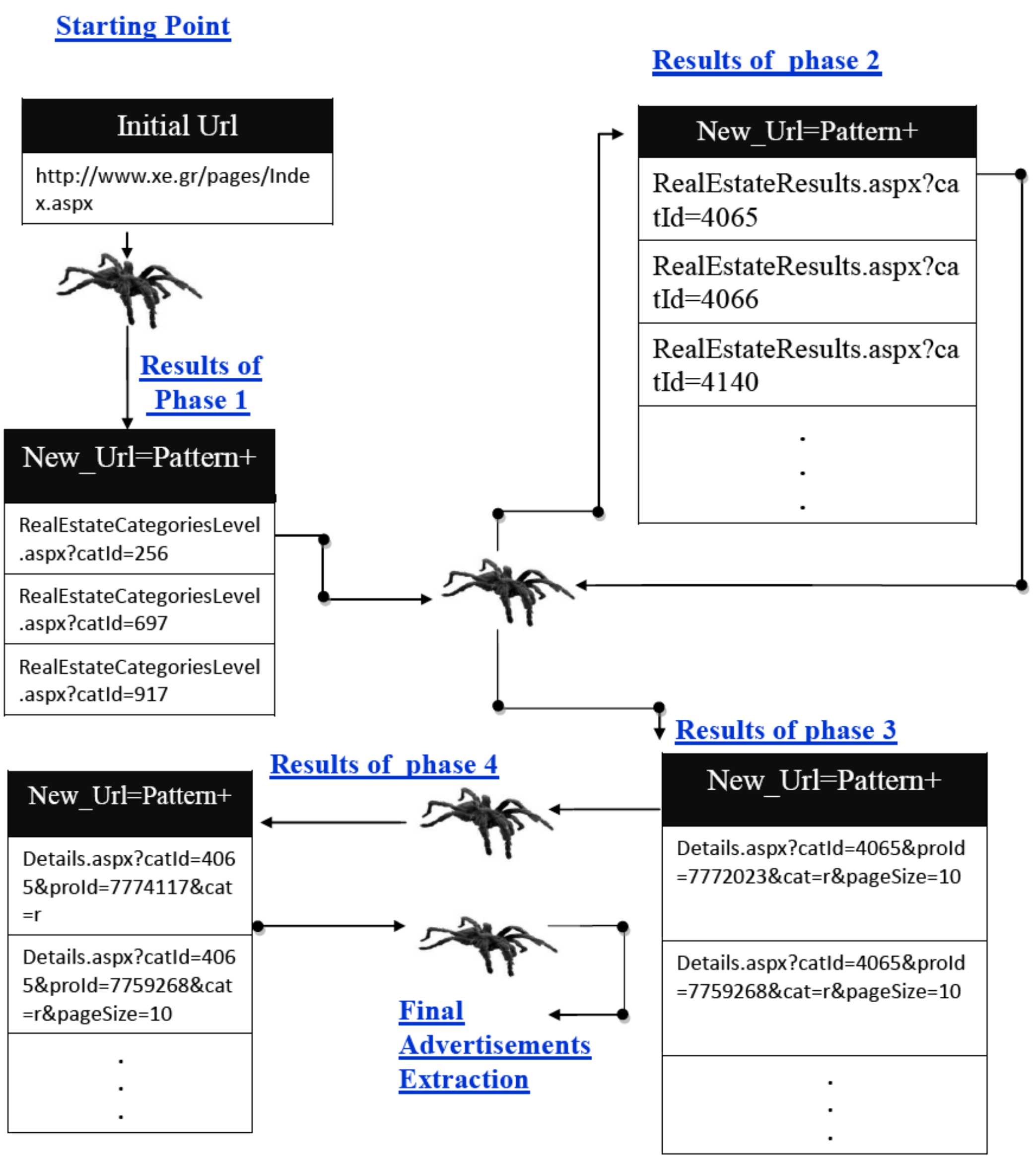

Starting Point
Initial Url
http://www.xe.gr/pages/Index.aspx
Results of Phase 1
New_Url=Pattern+
RealEstateCategoriesLevel.aspx?catId=256
RealEstateCategoriesLevel.aspx?catId=697
RealEstateCategoriesLevel.aspx?catId=917
Results of phase 2
New_Url=Pattern+
RealEstateResults.aspx?catId=4065
RealEstateResults.aspx?catId=4066
RealEstateResults.aspx?catId=4140
Results of phase 3
New_Url=Pattern+
Details.aspx?catId=4065&proId=7772023&cat=r&pageSize=10
Details.aspx?catId=4065&proId=7759268&cat=r&pageSize=10
Results of phase 4
New_Url=Pattern+
Details.aspx?catId=4065&proId=7774117&cat=r
Details.aspx?catId=4065&proId=7759268&cat=r&pageSize=10
Final Advertisements Extraction

Figure 3.2. Example

<u>Comments-Remarks</u>: As shown in the diagram above, each advertisement is the result of the compilation process (Crawling Process) in each phase.
Initially, the starting point used by default is the original direction of each site and a user query, which in this example corresponds to the tags (one for start and other for end) of which we are enclosing the text to retrieve. Therefore run a list of URLs (Results of phase 1). In this example, the links of such links are for sale, rental housing and land sales. So, taking the first item in the list of home sales and repeating all the details to create a new series: ***URL=Pattern+current_element_of_the_list.***
To rerun the data collection process with the new URL, and a new start tag and end tag, shows a list of new data as a result (Results of Phase 2). Note that these results relate to residential sales in specific areas. Start again, therefore, with the first element of the

new list for each item, create a new URL, which is derived as follows: ***URL=pattern+current_list_element.***
Repeating this process as we have all the elements of the list first, we have completed the data collection process for each site.

- In this example, the advertising recovery is carried out in 4 phases. This showed the structure of the advertising site. In other cases, it may take more or less action. Thus we see that when this is completed, it formed a complete index on the procedure to the extraction site for each ad. For a better understanding of the above process it can be thought a graphic representation ***G = (V,E)*** where:
- V: The set of vertices of the graph where each vertex represents a new link.
- E: All edges of the graph where each edge represents the relationship between each of the peaks (links).
- The root represents the initial step of each site.
- Each node represents a link of descendents who are the result of the crawling process.
- The leaves represent the final announcements.
- Each path from root to leaf represents a sequence of links until it recovers each advertisement.

3.2.2 Advertisements Feature Extraction Module

This unit carries out the export of the key features of each ad before its entry into the database. More specifically, because the crawler has exported ads by category and by region in the form of text, this unit is responsible for exporting the particular characteristics of each ad, depending on the kind of it.

For each type of ad are two types of features. The general characteristics, which are relate to the type, date of export, the source from which are exported, and the department, the earldom in question. The above information is usually extracted from the information provided by the website through the ad text. The second type of characteristics of the individual characteristics of each listing is the size that is removed from the ad text. These features are picked up from the main site of the announcement, and secondarily if it is determined from the ad text. Costs, unless stated in the notice if the value stored and the area is a property, of the earth. In addition, some sites offer information about the period in which each advertisement was published and in many cases, always in contact the agency responsible for this listing. Then, depending on the type of advertising, whether land or property, more information is stored. If it is a house, keep the number of rooms and floor location. Similarly, in the case of land, stores information about whether inside or outside the plan. Also one of the most important export dates is the type of property. This changes the formula depending on whether it is property or land. More specifically, residential properties are divided into apartments, houses, flats, penthouses, apartments, villas and studios. Land types are as follows: Land, property, plot, agricultural, area, vineyard, field.

- In all these cases, the information is stored, as determined by the text of each ad. Also, any information can not be assigned to any of the above categories, stored in a different field (other).

Problems-solutions:

- In this process we have found some problems. In some cases a solution was found. In others, it was impossible to find a solution, so it was simply to restrict the size of the problem. To name a few examples:

- The initial problems are in the extraction of the main area of the property and the secondary. As the actual location of this information cannot be identified by a specific keyword, you could not ensure the success of export. It has been partially addressed the problem with the structure of their ads. I.e., they tend to put ads on a specific point on the main page, or highlighting them. The secondary areas are normally found on the main site and often accompanied by certain keywords. Several of them have been identified, but there is a margin of error. To further increase the success rate in identifying the main area of the property, we have a list with a big number of areas to help us to identified its.
- A general and unexpected problem was the extraction of other features. There was no standard by which the ad should be structured, so the extraction of other features was quite difficult. This problem has been solved in part by studying the various advertisements to find the characteristics of a set of keywords. For example, there is a clear structure for calculating the number of bedrooms. Sometimes the number was next to the word room, or bedroom, or, literally, or word in another set of words. All keywords can be improved even more if there are other words that identify the selected features.

- Here's an example for the extraction of the key features of an ad to understand what is actually done in this system.

- Input: Papadiamantis March 25, attic of 75 square meters, 2 bedrooms, fourth floor, lounge, kitchen, bathroom, toilet, heating and no lift, price 100,000 euros.
- Output: *MainArea_id , Second Area* : Papadiamantis; *Bedrooms*: 2; *Size*: 75; *Price*: 100000; *Floor*: 4; *Other*: bright living room, kitchen, bathroom, toilet, heating and without elevators.

- In the example, receives as input the above entry and perform the procedure described above, produces the final announcement for entry into the database. As can be seen in the export of primary and secondary site all the ads on this site have this structure (primary-secondary). The number of rooms is returned by the token (integer) which is a position before the keyword ‘υ/δ’. Similarly for the other features are used keywords to find the text in the ad.

- The discovery of the main area of each ad plays an important role in the subsequent extraction of these characteristics. To achieve a significant increase in the probability of finding a large number of areas are stored in order to identify areas with the help of this list.

3.3 Database Process

3.3.1 Temporary Database Storage

After completing the admission process for a final announcement starts the import process to the final database. However, before final storage in the previous database, as shown in the diagram of the system, must be stored in a temporary database. Therefore, we use a database to temporarily transfer control of the uniqueness of the ads in the process of transferring them from temporary to permanent. More specifically, in the process of importing the database temporary ad, there is no verification that the ad is already in the earlier crawling process. Therefore, it is interesting to accelerate the process of collection of the ads, if we think that the system load by using only one database and no evidence before introducing the announcement, because the rate of data recovery crawling process goes far beyond search completion rate and the introduction of the ad. As a result we find a long queue of ads that are awaiting to be storaged in the database, which would be disastrous for the entire system. Thus, the ads are stored in a temporary database and once completed, we can execute the transportation process to the final database.

3.3.2 Final Database Storage

The completion of this process offers the integration process of the database. As mentioned, this process is the transfer of files, which is the result of the collection of data from the temporary database to the final database. This process only includes the new ads. For other announcements only keeps the information coming from the new crawling process, which was renovated in the period of publication. The transfer of control to the final database is accomplished by controlling the characteristics of each advertisement. If these are the same as the existing final advertisements in the database, then the ad is already in the database and is not necessary to be transported. In addition, an important technique to speed up the transfer process is that each candidate for the transfer is compared only with ads that have some common characteristics. More specifically, the same type of ads, the county and the same source from which it was recovered. Therefore, avoiding the comparison with all the ads in the database and is matched only by those who satisfy the above criteria.

Chapter 4

SYSTEM CHANGES

This chapter is the main chapter of the Diploma Thesis because it explains every change made in the Java programs developed initially to get again the properly working of the system.

4.1 Introduction to the Java packages and classes

All the system which is explained in the previous chapters has been developed in Java. The packages used in the system are as follows:

- *crawling*
- *crawling.aggel*
- *crawling.database*
- *crawling.databasecleaner*
- *crawling.entities*
- *crawling.estate*
- *crawling.kathimerini*
- *crawling.neomesitikh*
- *crawling.xe*

Each of these packages contains classes that perform specific functions. For packets with the prefix of the word "crawling." followed by the names of the sites *kathimerinh*, *neomesitikh*, *xe*, *Aggelos* and *estate*, it is obvious that contain the code of ads from their respective websites. For large packages like *crawling*, *crawling.database*, *crawling.entities* and *crawling.databasecleaner*, which include general titles, are essential to the process of collecting ads for these websites.

Here, we have a summary of all the packages with their classes in Java:

- <u>crawling</u>: *AdressesFeatExtraction.java, AnnFeatExtraction.java, Annkeywords.java, DataBaseConstants.java, DhmoiExtracter.java, HouseFeatExtr.java, ReadSource.java, SellEarthFeatExtr.java, Test.java, Utilities.java, WordUtilities.java*

- <u>crawling.aggel</u>: *AggelAdvertisements.java, AggelEarthFeatExtraction.java, AggelFeatExtraction.java, AggelHouseFeatExtraction.java*

- <u>crawling.database</u>: *DatabaseChangeManager.java, DatabaseDeleteManager.java, DatabaseManager.java, DatabaseQueriesManager.java, DatabaseSelectManager.java, DatabaseUpdateManager.java, DistinctionNew.java*

- <u>crawling.databasecleaner</u>: *ChageDates.java, CleaningDatabase.java, DublicateEntriesCleaner.java, ErrorClearer.java, WrongRecordsCleaner.java*

- <u>crawling.entities</u>: *Address.java, Announcement.java, AreaDescriptionEntity.java, AreaNumPriceEntity.java, Auction.java, CategoryValueEntity.java, DoubleStringFormat.java, HouseAnnouncement.java, IdDescription.java, IdPrice.java, MainArea.java, Period.java, PlotSellingAnnouncement.java, SearchInfo.java, SecondAreaPriceOtherEntity.java*

- <u>crawling.estate</u>: EstateAdvertisements.java, EstateEarthFeatExtr.java, EstateHouseFeatExtr.java

- <u>crawling.kathimerini</u>: KathimeriniAdv.java, KathimeriniEarthFeatExtr.java, KathimeriniHouseFeatExtr.java, KathimeriniTest.java

- <u>crawling.neomesitikh</u>: NeomesitikhAdvs.java, NeomesitikhEarthFeatExtraction.java, NeomesitikhHouseFeatExtraction.java

- <u>crawling.xe</u>: Advertisments.java, AuctionFeatExtr.java, EkswxikaFeatExtr.java, final_test.java, testing.java, XEAdvertisments.java, Xetest.java

With the aim of understand the relation between the classes, we show a diagram with the inheritance.

<u>CLASSES DIAGRAM. INHERITANCE</u>

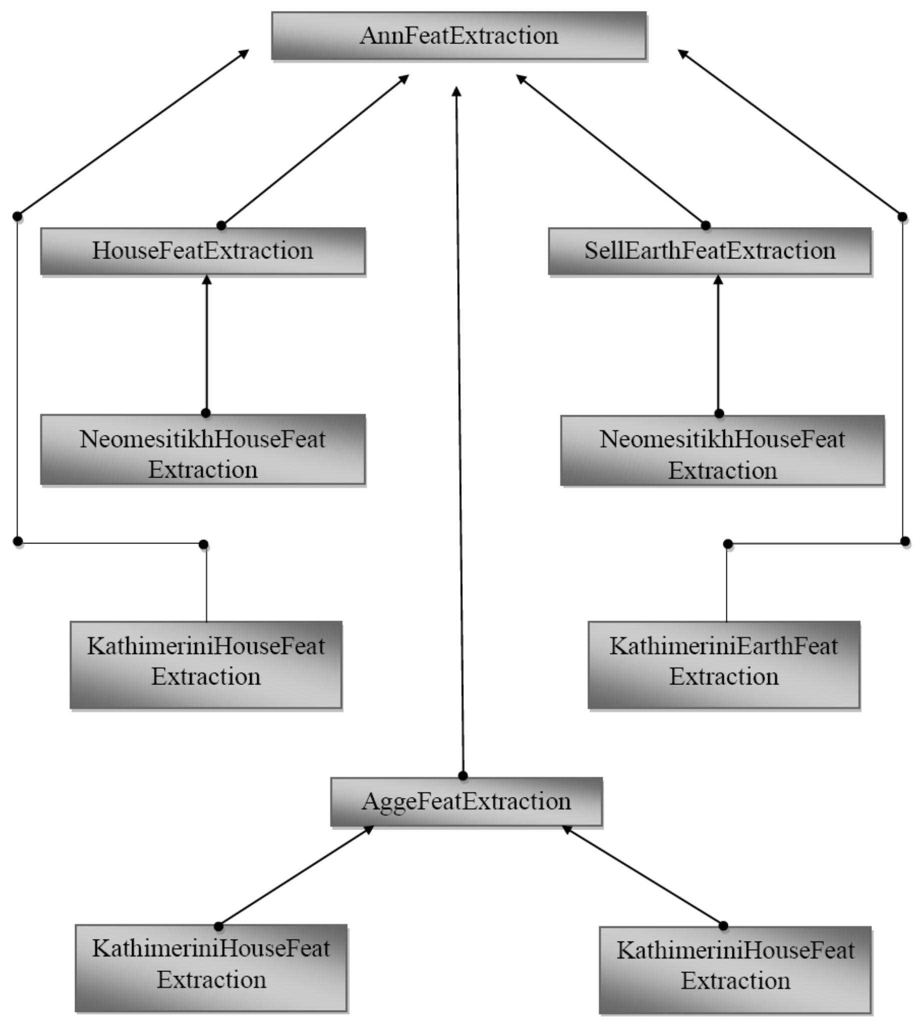

AnnFeatExtraction
HouseFeatExtraction
SellEarthFeatExtraction
NeomesitikhHouseFeat Extraction
NeomesitikhHouseFeat Extraction
KathimeriniHouseFeat Extraction
KathimeriniEarthFeat Extraction
AggeFeatExtraction
KathimeriniHouseFeat Extraction
KathimeriniHouseFeat Extraction

Figure 4.1: Classes Diagram. Inheritance between classes

4.2 Changes in the Classes

In this paragraph is going to be explained all the changes made in the Java classes to make the system work again.

4.2.1 Neomesitiki Website

The Website www.neomesitiki.gr is not available since last year. The web server doesn't work so, is impossible to work with this site and therefore, it can't be considered for the statistics.

4.2.2 e-state WebSite

This Website www.e-state.gr is available but there are no advertisements in the page. Trying with every different possibility for the search there is no result for it. This is the appearance of the website:

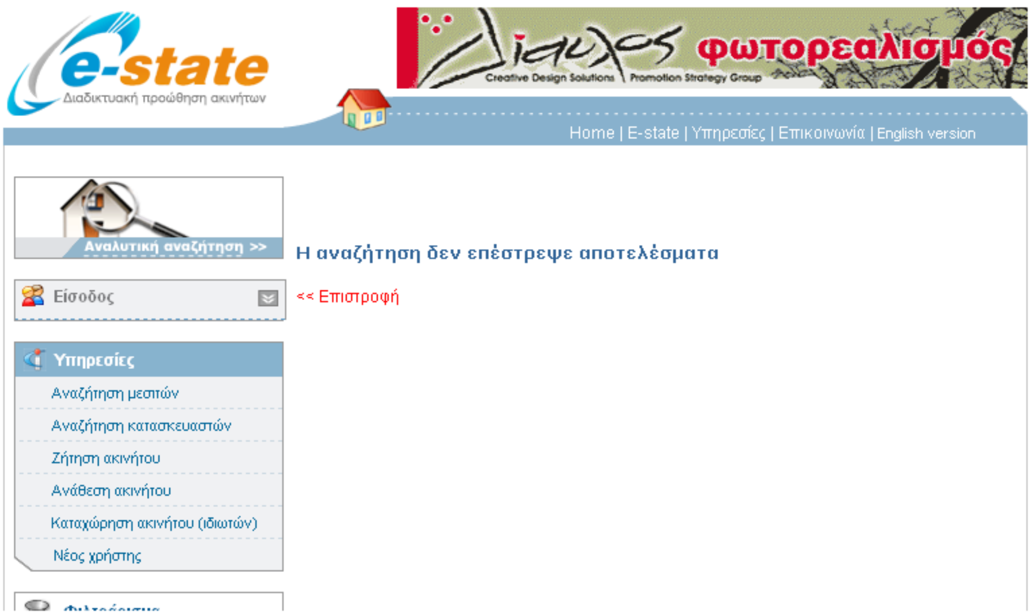

Η αναζήτηση δεν επέστρεψε αποτελέσματα

Figure 4.2: e-state website with no results for the search

4.2.3 Aggelioxoros WebSite

The website www.aggelioxoros.gr is working properly. Since last year the code of the web has changed and so, it has been necessary to change the Java programs to take the correct data from the website again.

The main class for this website is *AggelAdvertisements.java*. This class does everything that is necessary to get all the advertisements from the web: connect to the website to download the code, extract all the ads from this code, decides the kind of the ad (sell earth, sell house or rent house), connect to the database and change the information with the new one from each advertisement…

As it has been explained some sentences before, the code of the web has changed. Therefore, it was necessary to update the *AggelAdvertisemtents* class to take the correct information (advertisements) from de web code. In the following lines is going to be explained how we get it.

The first step is getting all the links that we want. For the rent house we include the advertisements for rent one room and the houses for holidays. To take all the links we use two tags, the start tag and the end tag. The code of each link is as follows:

```
<p class="hypcat"><a href="…">…</a></p>
```

Therefore, our tags are:
- Start tag for the link: “<p class=\"hypcat\"><a href=”
- End tag for the link: “</a></p>”

Using these tags we get all the links from the website but the problem is that we don’t need all of them. With the aim of take only the links that we want and separate them in types (rent house, sell house and sell earth) it has been necessary a piece of code like this:

```
if((text.contains("Ενοικιάσεις Κατοικιών")||text.contains("Ενοικιάσεις
Εποχικές")||text.contains("Συγκάτοικοι"))&&!(text.contains("Ζητήσεις")
)){
//This kind of advertisement is for rent houses
                        searchInfo.setOffer(1);
                        System.out.println("RENT_HOUSE");
                        editRentHouses(text,searchInfo);
}
if((text.contains("Πωλήσεις Κατοικιών")||text.contains("Πωλήσεις
Εξοχικών Κατοικιών"))&&!(text.contains("Ζητήσεις"))){
//This kind of advertisement is for sell houses
                        searchInfo.setSold(true);
                        System.out.println("SELL_HOUSE");
                        editSellHouses(text,searchInfo);

}
if((text.contains("Πωλήσεις
Οικοπέδων"))&&!(text.contains("Ζητήσεις"))){
//This kind of advertisement is for sell earth
                        searchInfo.setSold(true);
                        System.out.println("SELL_EARTH");
                        editSellEarth(text,searchInfo);
}
```

When we have all the links, we have to obtain the advertisements from each link. So, we use another two tags to get it. These tags are:
- Start tag for the advertisement: "<p class=\"p1"
- End tag for the advertisement: "</p>"

The next step (and the last one) is dividing all the information from the ad and put it into the database. This part of the system works properly except of the piece of the program that takes the price of the houses. So, the method mandated to do that has been modified as follows:

```
protected String getPrice(String[] text){

      for (int i=0; i < text.length; i++){
            String text_low =text[i].toLowerCase();
            String text_up = text[i].toUpperCase();

if((text[i].contains("0"))&&(text_low.equals(text_up))&&(((text[i].con
tains("."))&&(!text[i].endsWith(".00")))||((text[i].length()==3)&&(tex
t[i].endsWith("0"))))&&(!text[i].contains("-"))){
                        return text[i];
                  }
      }
      return "";
}
```

It was difficult to find a solution because the symbol "€" doesn't appear when the text from the website is taken. So, it was necessary to consider many cases and to be careful not to get the wrong information like dates, hours, number of the floor… Is because of that, the conditional sentence *if* is so long.

4.2.4 Kathimerini WebSite

The WebSite www.kathimerini.gr is working properly too. In this case, the code has changed as well, so it was necessary to rewrite the main program *KathimeriniAdv.java.*

First of all we have to choose the right links. Like in the previous paragraph, we get the links from the website using two tags. Each link has this form:

href='/4dcgi/classif/subcategorie_flag…frame…?cat?_id=…'>…</a></td></tr>

So, the tags are:
- Start tag for the link: "href=\'/4dcgi/classif/subcategorie_flag"
- End tag for the link: "</a></td></tr>"

Most of the links that we get with these tags are not necessary for us, so we have to select the right ones. For do that we use tree tables with specific words for each type of advertisement. These tables are:

```
public static final String[] sell_house_words = {"Ανατολικής
Αθήνας","Βόρειας Αθήνας","Δυτικής Αθήνας","Νότιας Αθήνας","Δήμου
Αθήνας","   Δήμου Πειραιά","Νομού Αττικής","Ακίνητα
Θεσσαλονίκης","Ακίνητα σε όλη τη χώρα"};

public static final String[] rent_house_words = {"Ακίνητα
Ενοικιάζονται", "Ακίνητα Θεσσαλονίκης"};
```

```
public static final String[] sell_earth_words = {"Ανατολικής Αθήνας",
"Βόρειας Αθήνας", "Δυτικής Αθήνας", "Νότιας Αθήνας", "Δήμου Αθήνας",
"Δήμου Πειραιά", "Νομού Αττικής", "Οικόπεδα Θεσσαλονίκης", "Οικόπεδα
σε όλη τη χώρα"};
```

For each link, we check the type of the advertisement (sell house, rent house or sell earth) with three methods:

```
public boolean is_sell_earth(String url, int c) {
            for(String i: sell_earth_words){
                  boolean b= i.contains(url);
                  if ((b)&&(c < 17)&&(c > 4)){
                        return true;
                  }
            }
            return false;
      }

public boolean is_sell_house(String url, int c){
            for(String i: sell_house_words){
                  boolean b= i.contains(url);
                  if ((b)&&(c >= 17)){
                        return true;
                  }
            }
            return false;
      }

      public boolean is_rent_house(String url, int c){
            for(String i: rent_house_words){
                  boolean b= i.contains(url);
                  if ((b)&&(c < 5)){
                        return true;
                  }
            }
            return false;
      }
```

The Website code is constantly changing. The way to handle this is by changing the value of the variable 'c' in the 'if' statement.

The next point is finding all the advertisements per page (link). There are two kinds of pages. The first ones that have links in this content and the others which have directly the advertisements in this page. In the first case we need tags to get these new links. The links are as follows:

<td class=small><a href='/4dcgi/claf_flag…frame…_id=…'>…</a>

Therefore, the tags are:
- Start tag for the new links: "<td class=small><a href="
- End tag for the new links: "</a>"

Is important to check in which case we are, with or without links. Is because of that, we use this piece of code:

```
if (urls.size()==0){
            String[] info = {"",""};
            info[0]=URL.substring(URL.indexOf("/4"));
            ArrayList<String> info2=Utilities.extractAllTexts(source,
"<div class='ctytle'><center>", "</center>");
            for(String i:info2){
                  info[1]=i;
            }
...
}//if (checking --> with links in the code)
//In this case, the page has got links
else {
…
}
```

If the condition is true we are in the first case, i.e. there is no links in the page. For the following steps, we need to store part of the link and the area in the array *info*. The rest of the code is the same for both cases.

Anyway, in both cases we need another two tags to extract the text of the advertisements. These advertisements are more or less like:

<TR><TD><span class="sidemenulinks"><P><B>…</B> … <HR><B>…</B>…</span>

For this reason, we use again two tags to find the ads in the pages. The tags are:

- Start tag for get code before the first advertisement: "<TR><TD><span class=\"sidemenulinks\"><P>"
- End tag for get code till the last advertisement: "</span>"

With this method we catch all the advertisements together, so, we have to separate them. In this case, the tag to do that is "<HR>". And the code is:

```
String[] advs = advsText.split(DEVIDE_TAG);
```

Where *DEVIDE_TAG* is "<HR>".

The next step (and the last one) is dividing all the information from the ad and put it into the database. Like in the previous Website, the method to get the prices has been modified. So, the method now is like that:

```
protected String getPrice2(String[] text){
      String ant = "";
      for (int i=0;i < text.length;i++){
            if (text[i].contains("€")){
                  return text[i].substring(0, text[i].indexOf("€"));
            }
            if
((text[i].contains("ευρώ"))||(text[i].contains("Ευρώ"))){
                  return text[i].substring(0, (text[i].indexOf("υρώ"))-
1);
            }
            if
((text[i].length()==4)&&((text[i].contains("Ευρώ"))||(text[i].contains
("ευρώ")))){
                  return ant;
            }
            ant=text[i];
      }
      return "";
}
```

There is a similar problem with the phones. Therefore, the new method is as follows:

```
      protected ArrayList<String> getPhones2(String[] text){
            ArrayList<String> phones = new ArrayList<String>(0);
            for (int i=0; i < text.length; i++){
                  String text_low =text[i].toLowerCase();
                  String text_up = text[i].toUpperCase();
                  if ((text[i].contains("-
"))&&(text[i].length()>=11)&&(text_low.equals(text_up))){
                        if (text[i].length()>13){

      phones.add(text[i].substring(text[i].indexOf("Τηλ.")+3));
                        }
                        else{
                              phones.add(text[i]);
                        }
                        return phones;
                  }
            }
            return null;
      }
```

4.2.5 XE WebSite

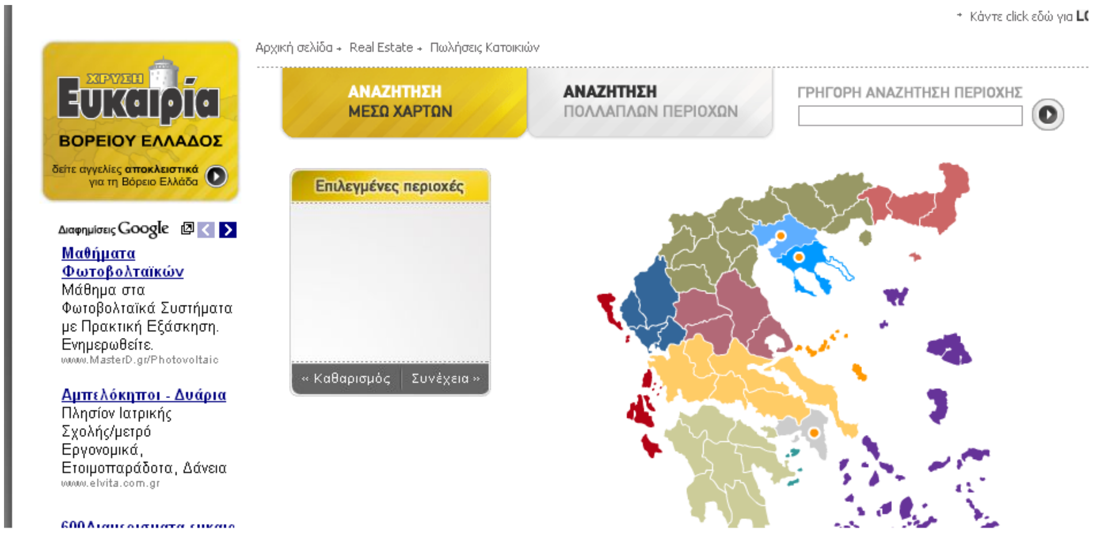
The Website www.xe.gr works but not for a long time. That means that the server works for a few minutes and then falls down. This makes the work more difficult because the programs need time to run and is impossible to test it in this way. Anyway we tried to fix all the problems and in the following lines are going to be explained all the changes that have been made although we are not sure of the result. Like in the other websites, the main program *Advertisements.java* has been changed.

The first step is getting all the links from the main page. This piece of code makes it possible:

```
if(URL.contains("Ενοικιάσεις Κατοικιών")||URL.contains("Ενοικιάσεις
Επιπλωμένων")||URL.contains("Ενοικιάσεις Εξοχικών Κατοικιών")){

        TypeOFAdv=AnnKeywords.RENT_HOUSE_CODE;
        editRent_Sell_Houses(URL);
        System.out.println("Finished Rent House Advertisments");
}
```

```
if(URL.contains("Πωλήσεις Κατοικιών")||URL.contains("Πωλήσεις Εξοχικών
Κατοικιών")){

        TypeOFAdv=AnnKeywords.SELL_HOUSE_CODE;
        editRent_Sell_Houses(URL);
        System.out.println("Finished Sell House Advertisments");
}
if (URL.contains("Πωλήσεις Οικοπέδων")){

        TypeOFAdv=AnnKeywords.SELL_EARTH_CODE;
        editSellEarth(URL);
        System.out.println("Finished Sell Earth Advertisments");
}
```

When we have all the links from the website, we need to take the advertisements of these pages. There are two kinds of links. The first ones lead us to a new page with advertisements and the others lead us to a page like the following one:



Μαθήματα Φωτοβολταϊκών
Μάθημα στα Φωτοβολταϊκά Συστήματα με Πρακτική Εξάσκηση. Ενημερωθείτε.
www.MasterD.gr/Photovoltaic

Αμπελόκηποι - Δυάρια
Πλησίον Ιατρικής Σχολής/μετρό Εργονομικά, Ετοιμοπαράδοτα, Δάνεια
www.elvita.com.gr

Figure 4.3: XE website without advertisements

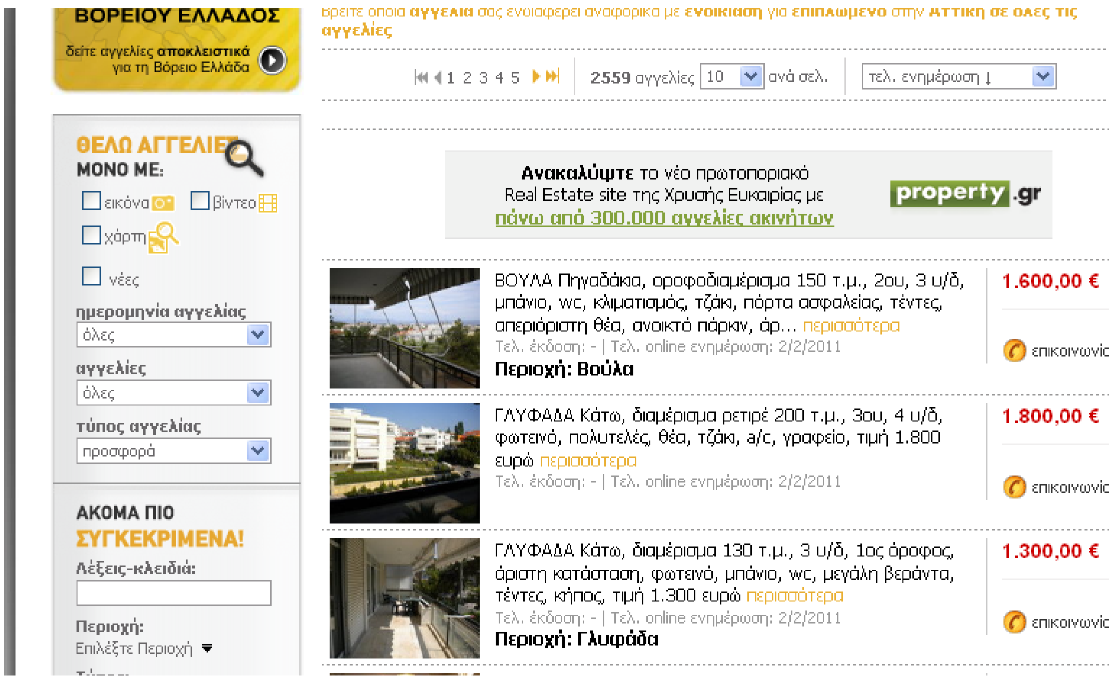
In this case the links for the advertisements are:

<br /><br /><a href="RealEstateResults.aspx?catId=…"><b>…</b></a>

Therefore, the tags for get the links are:

- Start tag for the link: "<br /> <a href=\""
- End tag for the link: "</b></a>"

The second kind of links is leading us directly to a page with advertisements like this:



βρείτε οποία **αγγελία** σας ενδιαφέρει αναφορικά με **ενοικίαση** για **επιπλωμένο** στην **ΑΤΤΙΚΗ σε ΟΛΕΣ τις αγγελίες**

1 2 3 4 5 | **2559** αγγελίες 10 ανά σελ. | τελ. ενημέρωση ↓

ημερομηνία αγγελίας: όλες
αγγελίες: όλες
τύπος αγγελίας: προσφορά

Ανακαλύψτε το νέο πρωτοποριακό Real Estate site της Χρυσής Ευκαιρίας με πάνω από 300.000 αγγελίες ακινήτων — property.gr

ΒΟΥΛΑ Πηγαδάκια, οροφοδιαμέρισμα 150 τ.μ., 2ου, 3 υ/δ, μπάνιο, wc, κλιματισμός, τζάκι, πόρτα ασφαλείας, τέντες, απεριόριστη θέα, ανοικτό πάρκιν, άρ... περισσότερα
Τελ. έκδοση: - | Τελ. online ενημέρωση: 2/2/2011
Περιοχή: Βούλα — **1.600,00 €**

ΓΛΥΦΑΔΑ Κάτω, διαμέρισμα ρετιρέ 200 τ.μ., 3ου, 4 υ/δ, φωτεινό, πολυτελές, θέα, τζάκι, a/c, γραφείο, τιμή 1.800 ευρώ περισσότερα
Τελ. έκδοση: - | Τελ. online ενημέρωση: 2/2/2011 — **1.800,00 €**

ΓΛΥΦΑΔΑ Κάτω, διαμέρισμα 130 τ.μ., 3 υ/δ, 1ος όροφος, άριστη κατάσταση, φωτεινό, μπάνιο, wc, μεγάλη βεράντα, τέντες, κήπος, τιμή 1.300 ευρώ περισσότερα
Τελ. έκδοση: - | Τελ. online ενημέρωση: 2/2/2011
Περιοχή: Γλυφάδα — **1.300,00 €**

Figure 4.4 XE Website with advertisements

The way to check in which case we are right now is this piece of code:

```
ArrayList<String> advs= new ArrayList();
ArrayList<String> links,ad;

if (url1.contains("CategoriesLevel")){
      links = extractAdvertisements(url1,Ann_start_tag,Ann_end_tag);
            for(String l:links){
                  String l2;
                  l2 = l.substring(0,l.indexOf("\">"));
                  l2 = Pattern.concat(l2);
                  ad =
extractAdvertisements(l,Ann_start_tag2,Ann_end_tag2);
                         advs.addAll(ad);
            }
}


else{
      advs = extractAdvertisements(url1,Ann_start_tag2,Ann_end_tag2);
}
```

When we have all the links for the different pages, we need the advertisements. They are like this:

</td> <td valign="top">…</td><td valign="top"> <div class="resultsSmalcol"> <nobr><span class="price">…</span></nobr>

So, the tags are as follows:

- Start tag for the advertisements: "</td> <td valign=\"top\">"

- End tag for the advertisements: "</span></nobr>"

NOTE: Is important to know that the last time that these programs have been checked was the day 11/02/11. Probably, the code of the websites has changed again after this day.

Chapter 5

SYSTEM USER MANUAL

The aim of this chapter is to explain how to run all the programs to make work the system.

5.1 Applications used

For this Diploma Thesis were necessary this applications or programs:

- Xamp Server & phpMyAdmin tool 2.5.8 to use Apache and MySql
- Eclipse SDK 3.4.2 to program all the classes in Java.
- mysql-connector-java-5.1.8
- J2SE-1.5 programming language

5.2 Main Classes

The main classes of the System and therefore, the necessary classes to run it are these ones:

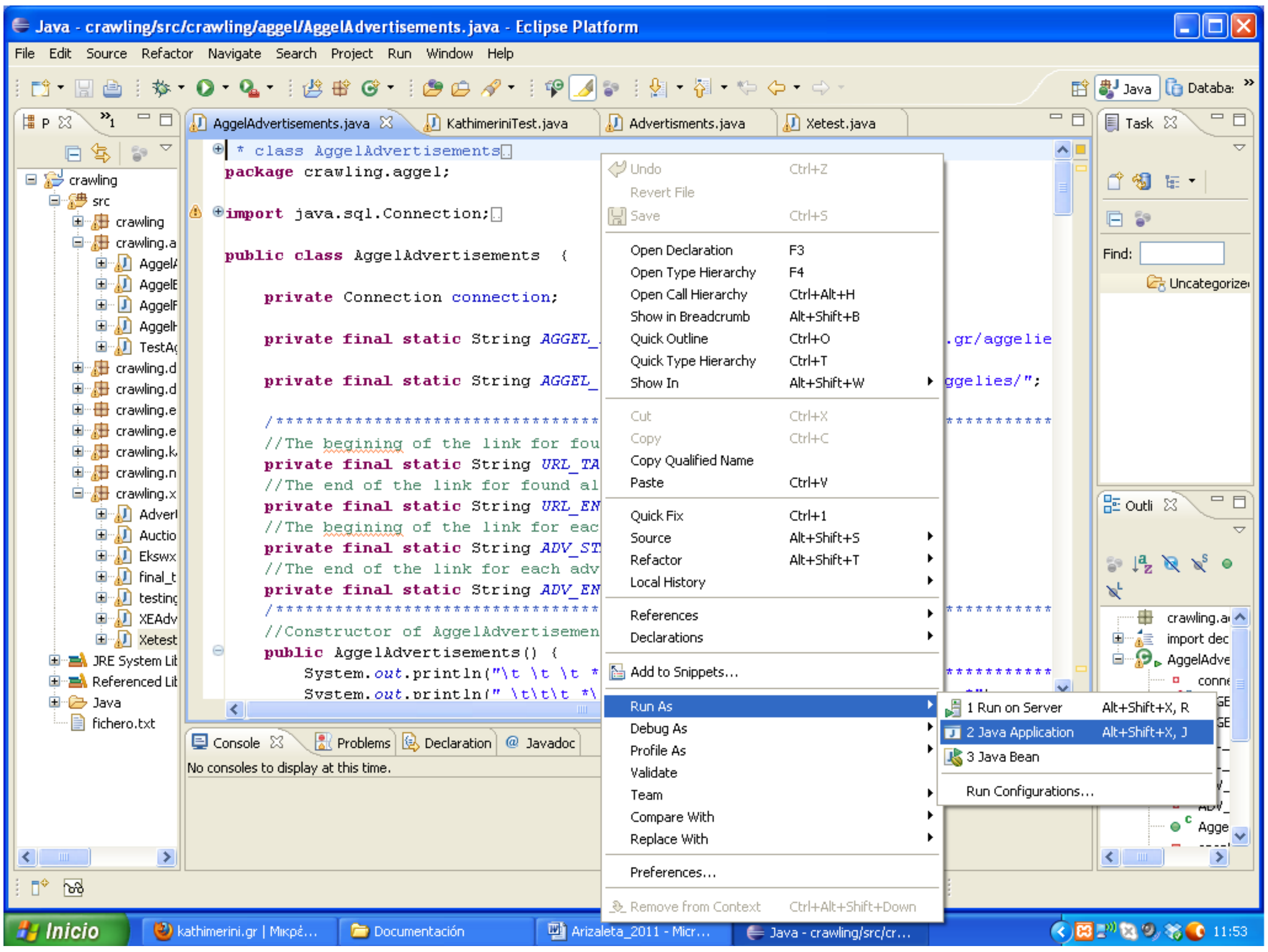
(To collect the information from the Websites)

- ***AggelAdvertisements.java*** from the package *crawling.aggel.*
- ***KathimeriniTest.java*** from the package *crawling.kathimerini*.
- ***Advertisements.java*** from the package *crawling.xe*.

The rest of the websites don't work, as it has been explained in the previous chapter.

To run these three classes first of all we need to have running Apache and MySql. In this case, we use the program Xamp, which includes these two applications.

With Eclipse opened (Eclipse is the application that we need to run the java programs), pressing the right button of the mouse above the main class and choosing *Run as/Java Application* the program will start to execute. In the next picture we can see it:



```
* class AggelAdvertisements
package crawling.aggel;

import java.sql.Connection;

public class AggelAdvertisements  {

    private Connection connection;

    private final static String AGGEL_                                  .gr/aggelie

    private final static String AGGEL_                                  ggelies/";

    /**********************************                                ***********
    //The begining of the link for fou
    private final static String URL_TA
    //The end of the link for found al
    private final static String URL_EN
    //The begining of the link for eac
    private final static String ADV_ST
    //The end of the link for each adv
    private final static String ADV_EN
    /**********************************                                ***********
    //Constructor of AggelAdvertisemen
    public AggelAdvertisements() {
        System.out.println("\t \t \t *                                 ***********
        System.out.println(" \t\t\t *\
```

No consoles to display at this time.

Figure 5.1: How to run a main class in Eclipse

(To clean the Database, all these classes are from the same package: *crawling.databasecleaner*)

- *ChangeDates.java*
- *CleaningDatabase.java*
- *DublicateEntriesCleaner.java*
- *ErrorCleaner.java*
- *WrongRecordsCleaner.java*

To run these classes we have to follow the same steps than in the previous paragraph.

Chapter 6

PROBLEMS AND FUTURE WORK

The last chapter of the project is dedicated to explain all the problems that have been founded during the development of the Diploma Thesis and a proposal of work for the future.

6.1 Problems

During the whole process of upgrading the system, have been found many problems.

Most of them were connected with the servers of the websites. In the Chapter 4 has been explained each case separately. Anyway, we will explain them in a short way.

The web server for *Neomestiki* doesn't work, therefore is impossible to work with this website. A similar problem exists with *e-state.* The server works but the website hasn't advertisements, so, there is no information to be considered in this page. In the case of *Xe*, the server works during a few seconds and then falls down during minutes or hours. Is because of that, the checking to test if the programs were working properly was really difficult. Even if we run the program during the whole day, it can't finish the tasks because it has to re-establish the connexion many times. This was one of the biggest problems that we found.

For the rest of the websites (*Aggelioxoros* and *kathimerini*) there was not any problem with the servers. They worked well. The difficulty was that the code of kathimerini has changed many times, once per 15 days more or less. So, we have had to change the programs every time that the code had been changed to fix it.

6.2 Future Work

In the following lines is gonna be explained a possible work for the future for the System of the Diploma Thesis.

It could be a possible work for the future creating this system but using different computers. It means that we will separate the tasks in PCs. For example, the web crawling will take place in one computer, the processing of the information in another one, the temporal warehouse in a different computer and the final warehouse in another one. Therefore we will have four different PCs (or more if we want one computer per website crawling) working communicated each other. To get this, we need to have all the computers in the same network and to use, for connect each other and send the information, RMI. Now, we are going to explain in a short way what RMI is.

6.2.1 Introduction to RMI

The Java Remote Method Invocation (RMI) system allows an object running in one Java virtual machine to invoke methods on an object running in another Java virtual machine. RMI provides for remote communication between programs written in the Java programming language.

A distributed application in Java need:

1. Locate the remote objects: With the ease *rmiregistry* Java or from remote objects as parameters or return values of remote method calls

2. Communicate with remote objects: RMI handles all details of the communication. For the programmer the remoting communication is equal to an invocation of a method standard local.

3. The download of kinds of objects passed as parameters or return value: RMI provides necessary mechanisms to download the code of both remote and local objects passed as parameters or return value, as well as passes their data.

6.2.2 Construction step by step

1. Write the RMI server code and Client
    - 1.1. Define the functions of the remote class as a remote interface
    - 1.2. Implement the remote class and the server
    - 1.3. Write a client program that uses the remote service

2. Compile the client code and server code

3. Start the RMI registry and server as well

6.2.3 Example: Hello World! With remote server

Step 1.1. Remote interface definition

```
package examples.hello;

/ / All remote interfaces must implement Remote interface
import java.rmi.Remote;

/ / All remote methods must have the ability to launch the exception RemoteException
import java.rmi.RemoteException;

public interface Hello extends Remote{
        String sayHello() throws RemoteException;
}
```

Step 1.2. Implementing Remote Object and Server

```
package examples.hello;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;
```

```
public class HelloImpl extends UnicastRemoteObject implements Hello {
        public HelloImpl() throws RemoteException {
              super();
        }
        public String sayHello() throws RemoteException{
              return "Hello World!";
        }

        public static void main(String args[]) {
              // Create and install a security manager
              if (System.getSecurityManager() == null)
                  System.setSecurityManager(new RMISecurityManager());
              try {
                  //Create a remote object
                  Hello obj = new HelloImpl();
                  // Register this instance with the name "HelloServer"
                  Naming.rebind("//IPmaquina:puerto/HelloServer", obj);
              } catch (Exception e) {}
        }
}
```

Step 1.3. Implement the client

```
package examples.hello;
import java.rmi.Naming;
import java.rmi.RemoteException;

public class HelloCliente {
        public static void main(String args[]) {
              try {
                   String message = "blank";
                  // In "obj" is stored the reference for the remote object
                  // This object has to be declared like the type of the remote interface of
                  // the object
                  Hello obj = null;
                  //Obtains a reference of the remote object
                  obj = (Hello)Naming.lookup("//" + getCodeBase().getHost() +
"/HelloServer");
                  //Le envía un mensaje al objeto remoto
                  message = obj.sayHello();
              } catch (Exception e) {}
        }
}
```

Step 2. Compiling the distributed application

- Compile the source files of the client and the server.
  Using the Java compiler *javac*

```
javac -d $HOME/public_html/myclasses Hello.java HelloImpl.java
        HelloApplet.java
```

- Generate stub and skeleton files.

  Using the Java *rmic* compiler

  ```
  rmic -d $HOME/public_html/myclasses examples.hello.HelloImpl
  ```

  Classes of *stub* and *skeleton* are generated

  ```
  examples.hello.HelloImpl_Stub.class
  examples.hello.HelloImpl_Skeleton.class
  ```

Step 3. Start the Java logging, Server and Client

- Start the Java logging

  The RMI registry is a simple name server in the side of the server that allows remote clients to obtain a reference to a remote object.

  To start the log on the server, the rmiregistry command of Java is running. This command produces no output and runs usually in the background. By default, the log runs on port 1099. To start recording at a port different, you have to specify in the command line.
- Start the RMI server

  ```
  java -Djava.rmi.server.codebase = http://myhost/~myusrname/myclases/
         -Djava.security.policy=$HOME/mysrc/policy
          examples.hello.HelloImpl
  ```

  codebase: the place to download the classes from

  policy: file that specifies the security policy to follow
- Start the RMI client

  ```
  appletviewer http://myhost/~myusrname/hello.html
  ```

BIBLIOGRAPHY

- Documentation for Web Crawling:

http://en.wikipedia.org

http://en.wikipedia.org/wiki/Web_crawler#URL_normalization

Diploma Thesis: "Crawling Process: The Real Estate Case"

Article: "SDMW 2008"

- Documentation for RMI:

http://www.lcc.uma.es/~pinto/apuntes/software%20de%20sistemas/Tema%204%20-%20Java%20RMI.pdf (Spanish)

http://en.wikipedia.org/wiki/Java_remote_method_invocation

- For programming in Java:

http://download.oracle.com/javase/1.5.0/docs/api/