INSTITUTE OF COMPUTER SCIENCE
Databases and Information Systems

Universitätsstr. 1, D–40225 Düsseldorf

HEINRICH HEINE
UNIVERSITÄT
DÜSSELDORF

Extraction and Storage of Web Structures

Maksims Abaļenkovs

Bachelor Thesis

Begin of Work:	15th December 2005
End of Work:	15th March 2006
Supervisors:	Prof. Dr. Stefan Conrad
	Prof. Dr. Martin Mauve

Declaration

Herewith I certify that this Bachelor Thesis was written by myself. No other sources and aids except the indicated were used.

Düsseldorf, 15th March 2006

Maksims Abaļenkovs

Abstract

The Internet is a multi-sided pulsating medium, which immediately reacts to every event happening in the world. Scientists are interested in studying inner processes taking place in the Web. Whereas investigation of the invisible architecture of the Web is one of important research fields. Understanding Web structure, which is one of the ground principles of the Internet, would result in a great number of improvements. That's why diverse range of technologies is developed to obtain the most precise data about the Web structure.

In this work we propose an algorithm to retrieve information about the inter-document bonds existing between Web resources. A compound extraction-storage tandem system will be suggested. At the beginning a short theoretical introduction will be given to supply the research basis. After that we will analyse and discuss the sort of information to be explored. Then we construct a database model and substantiate the selected data representation. To explain the model parallels among graph structures will be drawn. Based on our analysis of the expected value area an object-relational database will be created. Main purpose of the database lies in storage of the discovered data. Preservation of the extracted information is needed for the future analysis. The database will also provide more vivid and intuitive access to the information retrieved.

The second major part of the project is devoted to description of the program, intended to extract the Web structures. We will observe the program gradually. Starting from overall program's composition and division into packages, we will turn to smaller parts of the project. We will discuss the functionality of each important class and mention operation of vital methods. Then the main algorithm and the most interesting methods involved in the process of data extraction and storage will be presented. At the end we will illuminate prospects for future work and possible problem solutions.

Contents

List of Figures

List of Tables

1 Introduction

Nowadays the Internet growth sets another merits to its perception. It is a great phenomenon of people's creative power. The Internet could be observed not only as a wide area network but as a totally independent living organism. It offers a lot of possibilities to humans, beginning with entertainment, communication, booking tickets and electronic commerce, leading to comprehensive scientific surveys based on its unique properties.

It is a notorious fact that the World Wide Web (Web) became a tremendous source of information. The Web already ranks as one of the most popular information medium and it tends to replace common mass media in the future [CGJP03]. The role it plays in everyday life increases. Whereas it is possible to compare the Web with a large encyclopaedia, no one can assert that the information required, will be available on the Web at the time of request. Other vital question is, if the retrieved data is reliable.

In order to obtain the required data in most optimal way, deeper understanding of the underlying structure of the Web and especially spontaneous relationships arising between the component nodes is needed. Knowledge of the structure would entail considerable improvements to many fields concerned with the Web.

Work efficiency, popularity, and therefore commercial success of the search engines rely on good and frequently updated information about the Web framework and changes occurring to it. Precise information on how the largest and smallest Web nodes are combined and which document types are tied together would help to improve contemporary and give birth to new routing algorithms capable of faster data communications.

Knowledge of the structure would contribute for better deployment of application components and improvement of hardware allocation. This would reflect in higher quality of delivered services. Also security problems could be vanished if the detailed interconnection scheme between the local site and its neighbours is available.

Information on the Web structure would reflect in more directed and successful management, administration, and product design in sphere of IT business. Moreover meta-data of the Internet structure would supply a strong basis for statistical investigations and encourage fresh ideas and technological inventions.

The primary intent of our work is to explore the short- and long-term dependencies which exist between different Web nodes in the Internet. For collecting the useful data about the inter-site bonds we propose a Web crawler program, capable to operate in multi-threaded mode.

Since the extracted information is of a great importance and can not be examined right away we develop a database intended to preserve the discovered data. High emphasis was placed on determination of the value area we expect to obtain from the Web and implementation of efficient storing procedure.

2 Theoretical Background

In this section we will define the Web mining, review the problems which could be solved using its techniques, and discuss the general categories of the Web mining procedure. Then description of the Web structure mining will be given in more detail. After all information extraction and its major types will be presented. At the end we will speak about the scope of mining and the Object Exchange Model. This part of our work bases on the research [KB00] done by Kosala and Blockeel.

2.1 Web Mining Tasks

The Web became a huge source of information today. The rapidly increasing size of the Web results in an information overload experienced by an ordinary user. The most common problems while interacting with the Web are difficulty of finding relevant information due to low precision and low recall of modern search tools, creating new knowledge out of the information available, personalisation of the information, and learning about consumers or individual users.

The amount of information contained on the Web must be structured and indexed for the purposes of better utilisation. Data mining methods may be applied to find the information on a necessary subject. But since the Web offers new challenges to the traditional methods operating on flat data, the *Web mining* techniques have been developed further to Data mining.

Definition 1. Web mining is the use of data mining techniques to automatically discover and extract information from Web documents and services [Etz96].

Web mining stands at the intersection of several fields of computer science such as databases, information retrieval, and artificial intelligence, especially the sub-areas of machine learning and natural language processing. According to Etzioni [Etz96] Web mining could be decomposed into four sub-tasks:

Resource finding: the task of retrieving the intended Web documents.

Information selection and pre-processing: automatically selecting and pre-processing specific information from retrieved Web resources.

Generalisation: automatically discover general patterns at individual Web sites as well as across multiple sites.

Analysis: validation and/or interpretation of the mined patterns.

2.2 Web Mining Categories

The Web consists of many diverse kinds and types of data. Web mining is often divided into three categories, according to the investigated part of the Web. These are *Web content mining*, *Web structure mining*, and *Web usage mining*.

2.2.1 Web Content Mining

Web content mining is interested in discovering an intra-document structure of the Web, since its methods obtain effective data from text or hypertext of a single Web resource. Web content mining might be directed to exploring unstructured data like literary text, semi-structured data as HTML/XML documents, or more structured data, for example information contained in tables or Web sites generated automatically by means of a database.

2.2.2 Web Structure Mining

Contrary to Web content mining Web structure mining examines the inter-document structure. Its techniques are dedicated to reconstruction of the real topology of the Web, i.e. in comparison to Web content mining it operates on concepts of a higher order. The Web structure methods build up a model of links framework, which can be used for splitting the Web resources among different categories of interest. The derived model also helps to discover the similarities and relationships between diverse sites. Applying social networks theory and citation analysis various types of Web documents could be determined. For example *hubs* and *authorities* are found calculating numbers of incoming and outgoing links for a Web resource.

Hub is a Web site holding multitude hyperlinks pointing to pages of various subjects [Fin05]. Imagine a youth community Web site, that covers different cultural events such as theatre performances, photo-exhibitions, film festivals and concerts taking place all over the world.

Authority is an expert site, keeping a lot of information to one particular subject [Day99]. Thus it serves a target for many Web resources itself. Good example of an authority may be a Web resource observing solely the process of space discovery.

2.2.3 Web Usage Mining

Web usage mining collects the secondary data left by Web surfer's sessions and behaviours. Generally research on this topic is done by information providers, and is intended to learn the user's tastes and habits. The derived results might be used to improve the data presentation and customisation for various groups of users. Web design, management and marketing are other fields able to make profits on the gathered statistics.

2.2.4 Web Mining Scope

The Web mining categories may be also divided into two groups according to the *scope* of applying the mining techniques. Web content and Web usage mining are considered to operate within the local scope, since they work at the level of a single Web site. On the other hand IR (see Section 2.3) and Web structure mining are referred to the global scope, while their strategies employ the entire Web.

However it is not always possible to assign a Web mining algorithm to a specific category, since mining techniques may operate with methods from different categories. This is especially true for the modern applications where Web mining systems are frequently mixed to obtain the best results. For example Web content and Web structure mining are often used in tandem in case analyse of Web documents with contained links is needed.

2.3 Information Extraction

Speaking about *Information Extraction (IE)* we should mention the *Information Retrieval (IR)*.

Definition 2. Information retrieval is a process of automatic selection of the most relevant documents on a requested subject, while filtering as much as possible erroneous out.

Information extraction works on the primary data derived by means of IR. The general aim of IE is to represent the information in a more structured form such as database or a brief summary of a text, ready for further processing. One might say, that IR does the rough work and chooses the relevant documents, while IE applies more deeper and detailed techniques to extract relevant facts from these documents. A simple analogy may be drawn here. Realise that IR considers the data as a collection of unordered words, and IE is capable to discern parts of speech and build sentences. Whereas Web mining might constitute a part both of IR and IE, intended to improve the quality of delivered data.

Generally two types of IE are distinguished: IE from unstructured texts and IE from semi-structured data.

The first group also called classical or traditional IE works upon natural language texts and usually performs linguistic pre-processing of data prior to the actual extraction procedure. The pre-processing operation may include syntactic, semantic, or discourse analysis.

The IE from semi-structured data functions on basis of meta-information stored in the Web documents. This type of IE makes use of HTML tags, document's simple syntactics, or delimiters.

Machine learning algorithms and data mining technologies are often used to discover the data extraction patterns. These prototypes may be applied to accomplish the extraction procedure (semi-) automatically.

2.4 Object Exchange Model

Object Exchange Model (OEM) is a way to represent a semi-structured data as a labelled graph. In this model objects are described as vertices and labels are denoted as edges. Each member of the model is marked by an object identifier *oid* and holds a value, which might be either of atomic or complex character. In the latter case value is composed of a set of object references stored in (label, oid) pairs.

Generally usage of graph structures is widely spread in the sphere of Web mining. For example database view, that is one of possible representations of the Web content mining category, rests on the principles of the OEM. In addition data structuring in a graph form perfectly matches for application of machine learning algorithms.

3 Related Work

There exist a wide range of different algorithms intended to reflect the Web topology. Some of them are HITS, PageRank and extensions to HITS, which uses content information combined with the links structure and also applies filtering system to exclude outliers. These strategies perform calculation of quality rank and relevancy of each Web resource participating in the Web mining process. For example operation of Clever system and Google involves these techniques [Day99]. The above algorithms are also utilised for Web pages categorisation and discovery of micro communities.

Web warehousing is another research field for application of Web structure mining. The completeness of Web sites might be explored, measuring the popularity of local links located on a server. Estimation of Web documents replication across the warehouse permits to identify mirrored sites. And discovery of the hyperlinks hierarchy on the Web sites of a particular domain shows the effects of information flow on the sites design.

4 Tools and Preliminaries

4.1 Java

Java is the state-of-the-art object-oriented programming language containing diverse assortment of next generation features [Fla02]. It consists of three general components: *Programming Language*, *Virtual Machine*, and the *Platform*.

Virtual Machine enables execution of Java programs on different operating systems, as it serves a transitional unit converting intermediate byte codes created by the compiler into native machine-language instructions. Every following edition of JVM is tuned and optimised to provide a better performance, so programs' execution speed is not an issue any more and is comparable to native C and C++ applications.

Platform is a collection of predefined classes grouped in packages according to the kind of accomplished activity. This is the main ready to use programming instrument. Each program consists of a number of classes loaded dynamically in case of necessity. This kind of modular structure allows flexible extension and substitution of program's functionality.

Security principles were observed from the early stages of Java development. Restrictions may be assigned for any program launched and execution of untrusted code will cause no harm for the local system. Java also admits the importance of networking and a lot of attention is paid to deliver programming interfaces for working with network resources. Core support of 16-bit Unicode characters in Java makes it easier to write internationalised programs.

The utilisation of this language is becoming ubiquitous. Every popular Web browser has a built-in Java Virtual Machine. It is possible to run Java programs not only on desktop systems but also on television set-top boxes, PDAs and mobile phones.

The selection of Java as an implementation language for our project was mainly dictated by its wide variety of proposed capabilities, accents put on thread-safe concurrent programming, its prevailing status in programming sphere, and certainly platform independence and portability.

4.2 HTML Parser

HTML Parser is a Java library aimed for extraction and transformation of HTML data [OR05]. The package offers a wide spectrum of filters, visitors, custom tags and easy to use JavaBeans. The parsed information may be represented either in linear or nested fashion. Operation of the package is based on two general constructs: *Lexer* and *Parser*.

Lexer class contains methods to deliver extracted data in linear, sequential manner. It dissects the source document into a set of abstract nodes. Imagine a table with a single column where every row keeps a HTML tag or a string of text.

Parser class differentiates the embedding level of each explored tag and delivers structure of the analysed page. After the parsing process each opening tag has its own indent position according to the ending tag.

`Lexer` should be used where structure of the parsed document is not important and a developer is more interested in accessing individual, isolated nodes of the page. `Parser` constitutes a comprehensive alternative, capable of deeper understanding of document's composition. It is applied for processing of complex elements such as tables or frames.

HTML Parser ideally suites our main task of crawling through Web pages and extracting links. For the purposes of the project we have used class `Parser`, where tag attribute filters were applied to locate and collect maximum available hyperlinks held by a Web page. Also we have chosen `Parser` since we wanted to store the obtained source code in its original form as it was found in the Web.

4.3 DB2 Universal Database

For the purposes of our work the IBM DB2 Universal Database (DB2 UDB) was chosen. This is an object-relational database, where the database processor can represent data in form of tables, objects, or a combination of both [Moo04]. Objects symbolise cells located at the intersection of rows and columns. This kind of information representation provides access to the stored data in a way object-oriented programming languages do. That's why the universal object-relational databases are commonly used in the world of commercial applications. The UDB popularity gain is also promoted by the Structured Query Language (SQL), which embodies the principles of this sort of databases.

The DB2 UDB maintains a special directory called *catalog*.

Definition 3. Catalog is a hierarchically organised structure comprised of system tables containing the meta-data about information stored in the database.

The DB2 UDB may be managed using either graphic or command-line user interface, whereas the latter offers more detailed tuning possibilities like scripting.

4.4 Java Database Connectivity

The Java Database Connectivity (JDBC) is a programming interface intended to support communication between an application and a database [KE01]. JDBC is a variation of Open Database Connectivity (ODBC) initially designed for C and C++ programming languages.

JDBC provides the functionality for establishing connection to a database, requesting and updating the database by means of SQL statements, and obtaining the query results using standard Java classes `Connection`, `Statement` and `ResultSet` [EN02].

Since JDBC allows to represent the SQL calls to a database in form of dynamically generated strings, we have preferred it to another available interface SQLJ. Another circumstance contributed for the choice made was the relative JDBC universality. Because it is capable to interact with different database systems by means of one program, in case no database specific SQL extensions are used.

4.5 Web Mining Place

As we have seen in the previous sections Web mining appears in most fields of data mining and is employed for effective discovery of essential information in the Web. Web mining could be also viewed as part of the Web IR process or as an instrument of improving the IE process. It also absorbs the procedure of Knowledge Discovery in Databases (KDD). Because of existence of many standpoints about what place does a Web mining takes in a general data mining procedure, to extent of our work we will assume that Web mining is part of IE.

Other important question is, how do we define a Web structure? As long as we should analyse inter-document relationships we use the Web mining techniques at the global scope's level. In this case the Web structure could be regarded as the whole set of all accessible sites in the Web. But that would be too abstract for our research. That's why we will introduce the notion of a *minimum Web structure*. It will symbolise a single Web site with its first-level neighbours, i.e. with all sites to which we have hyperlink bonds from our source site.

The Web structures discovered in our work form a weighted digraph, where each Web site represents a vertex, and each hyperlink an edge [Wes01]. Since every hyperlink describes a one-way tie from source site to a target, and we want to retain the roles taken up by every Web site our graph is directed. For the reason that every source site may have a number of hyperlinks leading to the same target each edge in our graph has weight, intended to reflect the strength of inter-document relationship. The obtained graph will also contain loops because each Web site may keep hyperlinks referring to itself. The program developed within the framework of our research will operate on this graph.

5 Database Model

5.1 Database Structure

Figure 1 illustrates the Entity-Relationship Model. It represents both logical structure of the database and important information we want to store about Web documents. The database is built on basis of the weighted digraph described in Section 4.5.

Resource is the main entity type of the model. Objects of this type will contain the following nine attributes: resource identification *RID*, Uniform Resource Locator *URL*, resource type *RType*, character set *Charset*, discovery time *DTime*, discovery bot *DBot*, processing time *PTime*, processing bot *PBot*, and the source code *Source*. Resource identification is an entity key, since it is the only attribute containing a unique value for each entity. The resource stands in the inverse M to N relation "links" to itself. This means every resource may point to other resources.

Host is the second entity type. It has only two attributes: host identification *HID* and host name *HName*, where HID is an entity key. It lays in 1 to N relation "owns" to the entity resource. This way we are able to operate with a quantity of resources, which belong to a host.

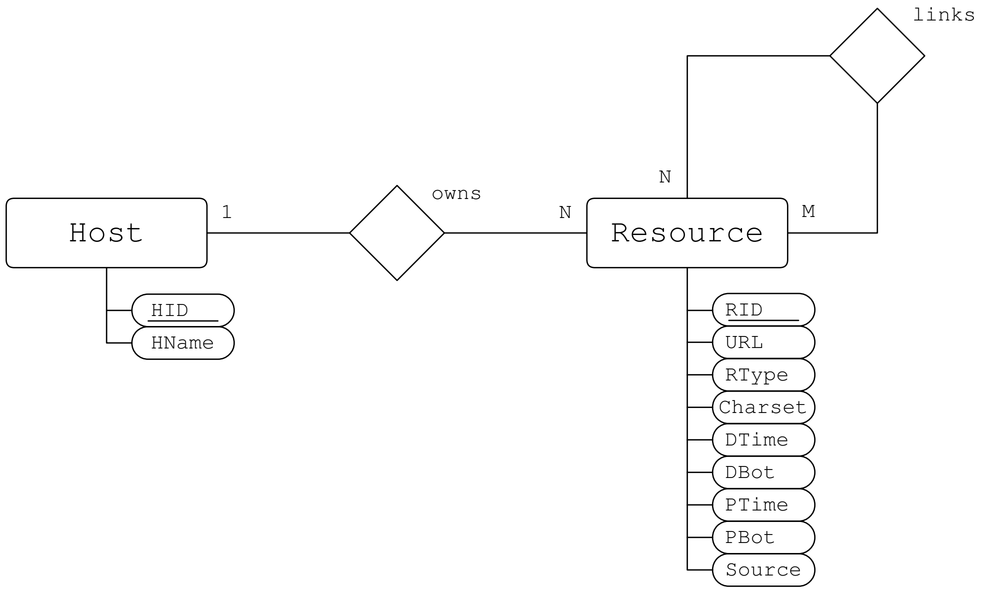

Figure 1: Entity-Relationship Diagram

5.2 Tables Design

In this section we will transfer the Entity-Relationship Model constructed earlier into Relational Model. We will also describe and substantiate the selection of data types for the values extracted and intended to be saved. There will be two primary and one auxiliary table in our database.

5.2.1 Table "Resources"

The general data about a Web resource will be stored in the table "Resources". Attributes of the resource entity are reflected in the Table 1.

Attribute	Type	Description	Constraint
RID	`CHAR(32)`	Resource Identification	$\neq 0$
URL	`VARCHAR(1024)`	Uniform Resource Locator	
RType	`VARCHAR(128)`	Resource Type	
Charset	`VARCHAR(40)`	Character Set	
DTime	`TIMESTAMP`	Discovery Time	
DBot	`CHAR(16)`	Discovery Bot	
PTime	`TIMESTAMP`	Processing Time	
PBot	`CHAR(16)`	Processing Bot	
Source	`CLOB`	Source Code	
HID	`BIGINT`	Host Identification	$\neq 0$

Table 1: Review of table "Resources"

Attributes RID, URL, RType, Charset, DBot, PBot, and Source have the `CHAR` data type. Column RID, DBot, and PBot values are represented by character strings of constant-length `CHAR`, whereas attributes URL, RType, and Charset are described by strings of variable-length `VARCHAR`. On account of the special case of Source value, Character Large Object `CLOB` was chosen. The selection of `CLOB` was also motivated by improved performance a DBS offers during the data manipulation. While the optimised operation on data is achieved over usage of *Locators*. Locators work with pointers to LOB data and accomplish logical changes to the information [KE01].

DB2-SQL differentiates between character types of constant and variable length. For storage and management of each constant-length character value will be used as much bytes as were noted in the definition of the column `CHAR` type, i.e. every entry will take up the same amount of space in the database, regardless of whether or not the entered string is as long as the set length – 1 byte for each symbol. In spite of that access to this data will cause less load for the processor.

For saving varying-length data strings DB2 will use only as many bytes, as the current data word needs. But management of a data word's length in RAM will require 2 bytes, because each data word's length is represented as a natural number of type `SMALLINT`. Besides handling of `VARCHAR` data increases the processor load [Moo04].

The selection of character length type for a column could be based on a simple formula. If it is supposed that an average character field's length will be above the mean value given, constant length should be chosen. Else it is more rational to use variable-length character type, especially when support of a rather big maximum value is required.

Resource Identification is a primary key in table "Resources" and must exist for each entry in the table. It contains a hexadecimal value of a MD5 hash code generated from the URL of a Web resource (see Section 7.2 for detailed description of Message-Digest Algorithm 5). We have selected the character type of constant 32 bytes length to store the symbols representing the digest. As long as we know that for string of any length we obtain 128-bit hash value, which hexadecimal representation will always occupy 32 characters.

Discovery and Processing Bot DBot is a bot's name which has discovered the resource and PBot is a name of the bot, which has processed the resource for the last time. Since we designate bots' names automatically in our program and none of them has a name longer than 8 characters, we set 16 as maximum allowed length for both of these fields.

Uniform Resource Locator As mentioned in [FGM$^+$99] the HTTP protocol does not place any a priori limit on the length of a URI. Servers must be able to handle the URI of any resource they serve, and should be able to handle URIs of unbounded length if they provide GET-based forms that could generate such URIs. A proper status code (414 Request-URI Too Long) should be returned to the user, if a URI is longer than the server can handle. But it is also noted that servers ought to be cautious about depending on URI lengths above 255 bytes, because some older client or proxy implementations might not properly support these lengths. In context of our project we consider URI and URL synonyms and allow URL lengths quadruple as long as older systems may recognise, namely 1024 bytes long.

Resource Type represents a MIME type of a Web resource. Although there exist no strict violations of any RFC limiting maximum length for a MIME type, Juniper Networks Inc. confine the length of the content-type header to 128 bytes, because they assume that valid headers tend to be short, and are unlikely to exceed this value [Ju0]. Also RITLabs programmers, the developers of "The Bat!" email client, claim, that it is always recommended to keep the maximum MIME line length at 72 characters [Ri0]. And since we await that MIME types of the majority of Web resources will be shorter than 64 bytes, we have constrained RType to maximum of 128 varying-length characters.

Character Set describes the character encoding of a Web resource that may be indicated in content-type field of a Web document. Each resource will have `NULL` value in this cell by default. The encoding will be set only in case it was found in the source code of a resource. According to [IAN05] the character set names may be up to 40 characters taken from the printable characters of US-ASCII, where no distinction will be made between use of upper and lower case letters. That's why for this column we have selected character type of varying-length with capability to store 40 symbols.

Discovery and Processing Time DTime is time indicating when the resource was first discovered, and PTime is time when the resource was completely processed by any of the bots. Both columns will contain values of `TIMESTAMP` data type.

Source Code value will hold source code of the HTML/XML document. Since it is impossible to predict the total size of the source code the `CLOB` type was chosen for this field. And the database was adjusted to pass through 32 KB data words, that is the maximum allowed size.

Host Identification is an identification number of host, to which a Web resource belongs. Since the Web consists of approximately 12 milliards of pages and tends to grow [GS05], the `BIGINT` type was selected for this attribute to support as much host names as possible. HID is a foreign key in table "Resources", which realises 1 to N relation with "Hosts" table, meaning that one host may "own" many Web resources. The HID value is not allowed to be `NULL` and has to be present for every entry in the table.

5.2.2 Table "Hosts"

"Hosts" is the auxiliary table introduced to save host names separately. We reckon separate placement of hosts will promote flexibility in further analysis and statistical research of Web structures. It will also adhere the second normal form [EN02]. This table has only two columns (see Table 2 for a review).

Attribute	Type	Description	Constraint
HID	`BIGINT`	Host Identification	$\neq 0$
HName	`VARCHAR(64)`	Host Name	

Table 2: Review of table "Hosts"

Host Identification is a primary key in this table and will be set automatically by the Database Management System (DBMS). The initial value for the first entry is 1 and others will receive current value incremented by 1. Other data type's properties are inherited from HID column in table "Resources" (see Section 5.2.1 for a description).

Host Name is a name of host, to which a Web resource belongs. According to [Moc87], the labels must follow the rules for ARPANET host names. They must start with a letter, end with a letter or digit, and have as interior characters only letters, digits, and hyphen. Length of the labels is restricted to 63 characters. And so host name column in our database is depicted by varying-length character string of 64 bytes.

5.2.3 Table "Links"

Very important information about dependencies between Web structures will be stored in the "Links" table. It describes the inverse M to N relation, where one resource "links" to the other. This table has two columns (see Table 3 for a review).

Attribute	Type	Description	Constraint
FRID	`CHAR(32)`	Father Resource Identification	$\neq 0$
RID	`CHAR(32)`	Resource Identification	$\neq 0$

Table 3: Review of table "Links"

Father Resource Identification holds the resource identification of the container Web structure. This is a normal RID of the page being analysed.

Resource Identification describes a containee Web document.

Both columns are of the same data type and contain fixed-length character strings 32 bytes long, where none of the cells are allowed to be `NULL`. FRID and RID values are foreign keys, since table "Links" is a classical case where problem of reflecting a M to N relation is solved by means of a subsidiary table containing key attributes both of represented entities [EN02]. Foreign keys also protect the table from registration of fortuitous values, not contained in the table "Resources".

An entry in table "Links" says that FRID resource holds a hyperlink pointing to RID resource, i.e. that HTML/XML source code of a FRID document has "HREF" or "SRC" attribute with URL of the RID document. For example, if we have found five URLs in FRID's source code, five entries we be inserted in the table "Links", where FRID will always be the same hash code string and RID will be a newly constructed value for every explored URL.

It was possible to introduce one more attribute of type `INTEGER` to store the counter value indicating the number of equal records encountered during the information extraction. The third column in table "Links" would show the strength of stored relationships (see Section 4.5 for an explanation). But insert of a value pair in this case would require an extra procedure to check, if this couple of values was already present in the table. We assume management of an additional column would reduce performance of the database too much. Moreover quantity of equal value pairs may be derived using a simple SQL query with an aggregate function `count()`.

5.3 Required Capacity Estimation

Let us proceed to the required capacity estimation for the whole database. We will start with a discussion how much place would require a single entry in each table and then turn to approximate capacity estimation for saving of one Web structure. Finally we draw a conclusion about the possible size of the filled database and its storage capability.

According to the reference given in [IBM05] `TIMESTAMP` value is stored internally as a packed string of 10 bytes and `BIGINT` value uses 8 bytes of storage space. As long as in our database we work with character string data in UTF-8 we should also remember that every ASCII character is one byte, but non-ASCII characters take two to four bytes each. Hence as mentioned earlier we need 1 byte of memory space to store one symbol of `CHAR` type and 2 bytes for each symbol of `VARCHAR` type. The `CLOB` type we use for saving source code of a Web page will be the only field where we may encounter international symbols. Let us suppose that 25 percent of the data saved in the Source column will contain Unicode characters and storage of every such character will require 3 bytes. According to known amount of memory needed to store each DB2 source data type we are now capable to make our calculations.

Resources		
Attribute	**Value, B**	**Formula**
RID	32	
URL	512	256 symbols × 2 B
RType	64	32 symbols × 2 B
Charset	40	
DTime	10	
DBot	16	
PTime	10	
PBot	16	
Source	112500	12500 symbols × 3 B + 37500 symbols × 2 B
HID	8	
One record	*113208*	
Hosts		
HID	8	
HName	64	32 symbols × 2 B
One record	*72*	
Links		
FRID	32	
RID	32	
One record	*64*	
Total	**113344**	

Table 4: Required Capacity Estimation

Take a look at the Table 4. One record in table "Resources" will totally occupy 113208 bytes. Also we have to take in account two values in table "Hosts". On average one record in there will reserve 72 bytes. And the last part left, is calculation of the database memory usage for saving hyperlink dependencies in the table "Links", where storing of one record will take up 64 bytes. Summing up all the results we deduce, that storage of a mean Web structure will require 113344 bytes.

We should mention, that the above calculation is feasible, only under assumption that average length of URL, RType, Source and HName fields are 256, 32, 50000, and 32 symbols correspondingly. Where the Euronews' Web page [Eu0] was taken for the reference.

6 Program Review

The main idea of the program was to implement a *Web crawler*. The crawler may consist of some number of simultaneously running bots, that fill up the underlying database with the discovered data. Most interesting moments of the work were elaboration and deployment of program's interface for interaction with the database. This is where development of program's structure for representation of extracted data comes in play. For better understanding of tasks and results of our project we will start our description with the most general aspects of program's design and will gradually immerse in implementation details.

6.1 Packages

Our program is divided into a number of packages. Every package forms a group of classes intended for execution of some special task. The five most important packages are shown at the Figure 2.

Capturer is the first package. It consists of three classes, which are `Capturer` itself, `BotManager` and `Bot`. `Capturer` is a collective class, which only holds the main method, initializing the whole program. Since we want to work with a number of bots a class `Bot` was introduced. This is our extension to standard `Thread` class in Java. The last class in this package is `BotManager`. It contains methods for controlling the functionality of different bots.

Database is another package composed of two classes: `Database` and `LoginInformation`. Class `Database` provides methods for cooperation with the database. And `LoginInformation` is an auxiliary construct for effective storage of user's login data.

Resource is a very interesting and probably the most important package in our project. It describes two general concepts of our program: `Link` and `Page` classes, which represent the data extracted from the Web. Both of them describe a Web resource. `Link` is a generalised form of a Web resource. Whilst `Page` is a detailed version of a Web resource. It inherits all `Link` attributes and methods.

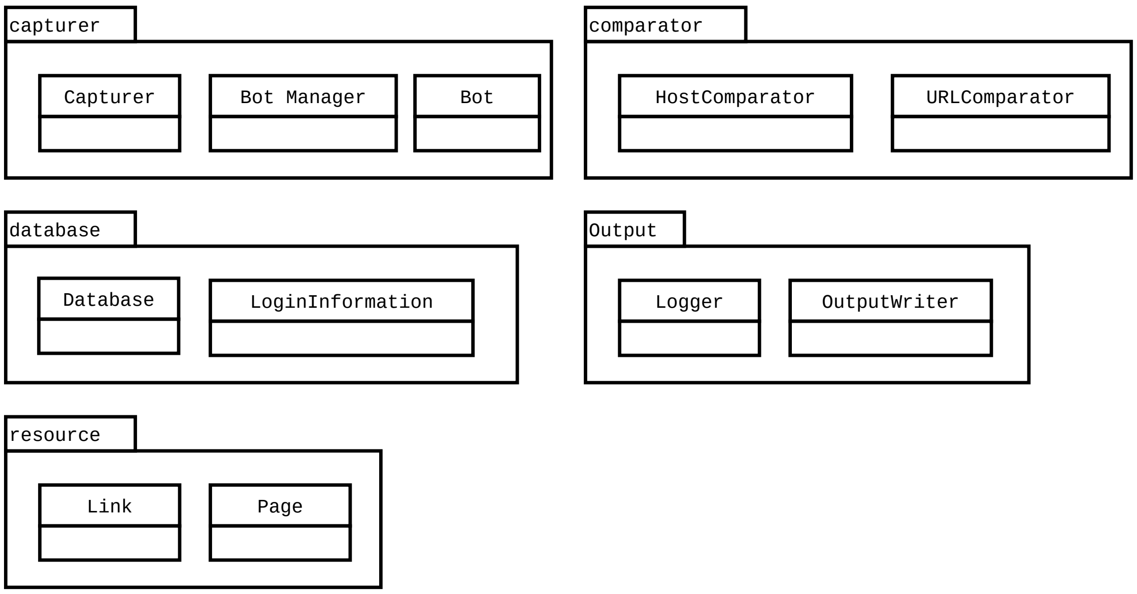

Figure 2: Package Diagram

At this point we should clarify the main difference between `Link` and `Page`. `Link` realises a hyperlink found in the source code of a Web document. Every `Link` contains raw intermediate data, which has to pass a preliminary analysis procedure and then will be given to the database for storage as a possible candidate for more detailed future examination. Whereas the `Page` object is built of data requested from the database for final processing. Every `Page` instance has a special container for collecting links found in its source code.

Comparator is a subsidiary package responsible for comparison of different link attributes. Its `HostComparator` and `URLComparator` classes implement Java `Comparator` interface.

Output is the last major package. It includes `OutputWriter` and `Logger` classes and is intended to support program's logging.

6.2 Key Classes

Now when we know the structure of primary packages we can turn to discussion of program's key classes. Pay attention to the Figure 3. This is a UML class diagram. For the sake of lucidity classes depicted here are arranged in the same order as they were mentioned in the above paragraph. Some classes as `Capturer` and `Test` are not listed here since they have neither complicated structure nor contain crucial methods.

To follow the main idea of object oriented programming we have reduced the visibility of program's methods and attributes to the required minimum. We will debate on visibility only in special cases and assume that in general each class is permitted to access the attribute fields and class instances it really needs. Some classes offer public accessor and mutator methods for their attributes where appropriate.

Bot is based on the functionality of built-in Java class `Thread`. However some specific attributes as `runs`, `db`, `file`, and `page` were added to it. Boolean variable `runs` indicates bot's working status. It shows if the current bot is active at the moment. It should be set to `false` to interrupt the bot's operation. Database object `db` is a connection to the database granted to every bot. `file` is an instance of class `OutputWriter`. Using this attribute bot logs his control messages into a file. Field `page` is intended to save the intermediate Web resource, which is processed by the bot. A new bot is created from supplied name and login information in a constructor `Bot()`. And the overall page processing is performed in the method `run()`.

BotManager is the next class in our diagram. Its attributes are vector `bots`, which tracks the running bots, string array `botNames` containing bots' nicknames ready for designation, and a constant `MAX_BOTS_NUM` implemented to limit the maximum allowed quantity of running bots. Methods `startBot()` and `stopBot()` are used to control the bots. Accessor methods `getBotByName()` and `getBotByIndex()` return a `Bot` instance according to supplied name or index parameter. And method `getNumBots()` delivers the current size of `bots` vector.

Database class owns the following attributes: `con`, which is a connection to the database, `rs`, that is a `ResultSet` object containing the table of results returned on a select query, `page` representing the `Page` instance a database works with, and a number of strings for storage of SQL queries or statements message bodies. On the diagram they are shown as `selN`, `insN`, and `updN` fields. Every database object has also one statement `stm` and many prepared statement attributes denoted as `psSelN`, `psInsN`, and `psUpdN` on the diagram. The last attribute is the `MAX_ROWS_NUM` constant, where the maximal number of delivered rows for the `rs` `ResultSet` is kept. Login information in the constructor `Database()` is used to establish connection and prepare all statements required for the interaction.

The most substantial methods of the class are `requestPage()`, which requests a new unprocessed Web resource from the database and `submitPage()`, that sends the completely processed page back to the database. Both `requestPage()` and `submitPage()` were made `synchronized` in case number of `Bot` objects will share a database connection and support of concurrent access to these methods would be required in future revisions of the program.

In order to retain clearness of the program the comprehensive method `submitPage()` was divided into four private methods: `operateHost()`, `operateResource()`, `operateBond()`, and `operatePage()`. Each of these hidden methods is intended to accomplish its own part of the whole submit procedure. Method's name points to the function it completes. This means that `operateHost()` for example communicates with the table "Hosts" and `operateResource()` with the table "Resources", and so on in compliance with the method's name. The method `operateBond()` makes an entry about an existing bond between the current page and a link into the table "Links". And

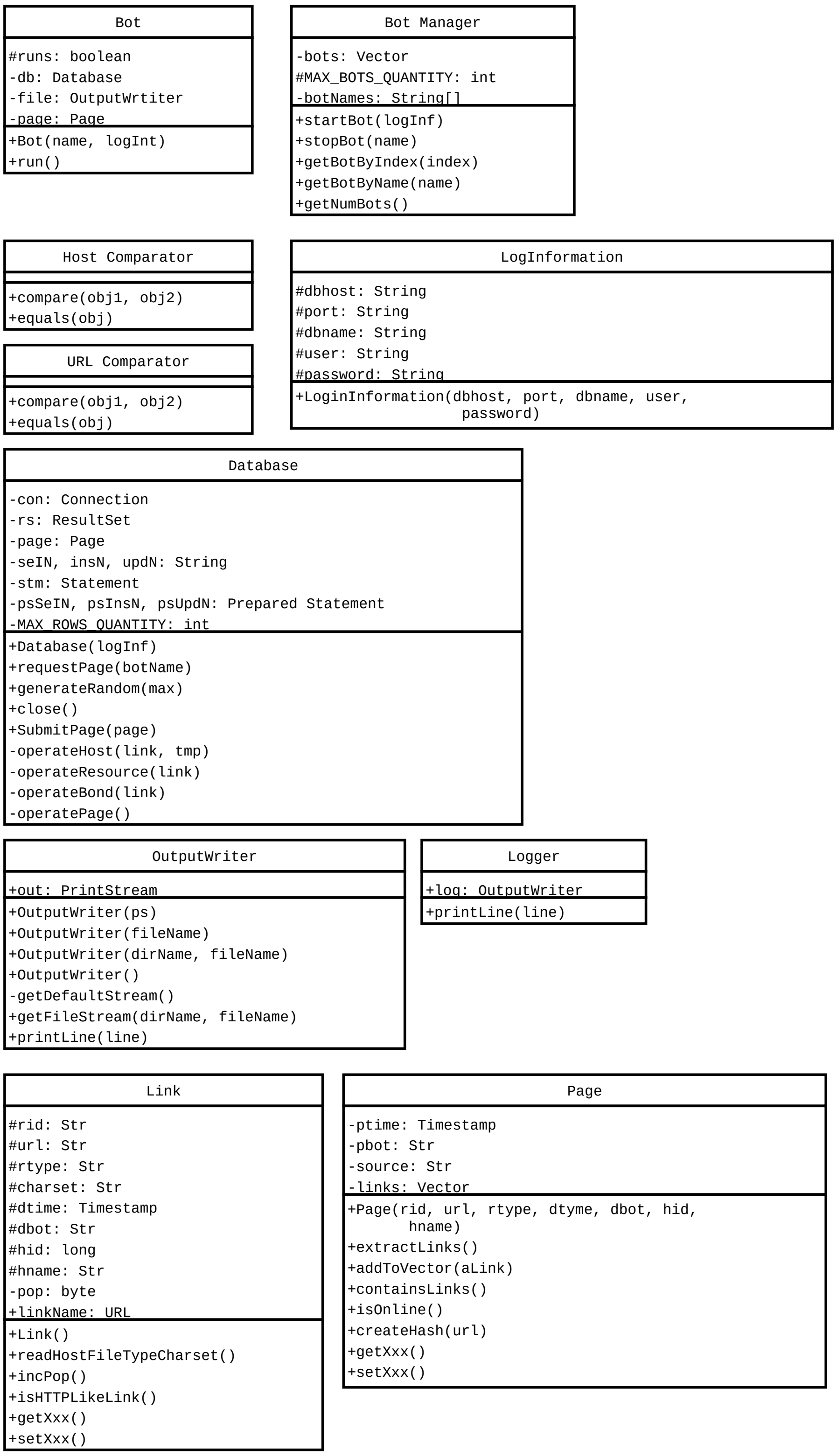

Figure 3: Class Diagram

operatePage() refreshes the Page instance with the latest data explored, imprints it with the processing bot's name and a timestamp, and finally updates its record in the "Resources" table in the database.

To provide an arbitrary choice of the row in a ResultSet method generateRandom() was introduced. The last method in this class is close(). It closes all prepared statements created and terminates connection to the database.

LoginInformation is a simple container class. It represents different text strings required for successful connection to a database as a single object. The class attributes are dbhost, port, dbname, user, and password. And they keep information about the host name of the database, port number to connect to, database's name, user account, and password respectively. This class has only one constructor method, which creates a LoginInformation object from the data supplied.

Link class has a wide range of attributes intended to reflect the same structure of a Web document as presented in the table "Resources" in the database. Visibility of link's attributes is set to protected, while most of them will be inherited by a successor class Page. DB2-SQL source data types [IBM05] in our Link class are introduced as follows: all fixed- and varying-length string objects as well as CLOB are mapped to Java strings, TIMESTAMP variables are left of the same type in Java and type long, which is a Java equivalent for BIGINT, is used to keep the HID value.

Variable pop is a counter, indicating link's popularity in the source code of the page. We have set its type to byte, as we assume that none of the analysed Web documents will have more than 255 equal hyperlinks in its source code. This counter is the only private attribute in the class, since it does not have to be succeeded by the Page object. The linkName is an additional attribute, which is handled in many places of the program, especially by Link and Page classes. It is used to create a Java URL object and be able to apply specific methods predefined in java.net package of JRE System Library. For example a good use of it is made in the readHostFileTypeCharset() method, where link's host, file part of the URL, content type, and character set are gained. Constructor Link() creates a single Link instance with NULL values of all fields, except the pop, which is one. Method incPop() increments the popularity's counter value, in case the same hyperlink was found in the source code of the document again. Other valuable method in this class is isHTTPLikeLink(). It is a compound method, which consists of a collection of logical instruments responsible for filtering out mail, JavaScript, FTP, IRC, news, Telnet, WAIS, and Gopher hyperlinks. The class also contains accessor and mutator methods for each of its attribute. They are marked getXxx() and setXxx() on the diagram.

Page class adopts almost all Link attributes. At the same time it has an extra set of fields completing full structure of a Web document. Attributes ptime, pbot, and source are containers for holding last processing time, processing bot's name, and HTML/XML source code values, respectively. Vector links is an additional construct. It will keep the links, that belong to the current Page object. Page object is built of six primary attributes: RID, URL, RType, DTime, DBot, HID, and HName. They are delivered by the database on a requestPage() call. These fields are needed to start the page analysis and contained links extraction, which is performed by the `extractLinks()` method. Logical methods `isOnline()` and `containsLinks()` are charged with the preprocessing of the page instance. As soon as a new link is found in the source code, `createHash()` generates the RID value for it. Implementation of `createHash()` uses the functionality of the Apache's SOAP [Ap0] method `Hex.encode()` to produce the hexadecimal form of a digest created. Further processing of the new link object takes place in the `addToVector()` method, which finally puts this link into a vector.

HostComparator and URLComparator classes serve for defining the principles of `Link` fields equality. First compares the host names of `Link` objects and second the URLs. We will use these classes to sort vector links in the alphabetical order according to the parameter given. As long as these classes implement Java `Comparator` interface they only override methods `compare()` and `equals()`. Since we wanted to indicate the sorting criterion ourselves we have not used pre-defined Java `sort()` or `contains()` methods.

OutputWriter class has only one variable `out`. It holds the value of the `PrintStream` instance used by the class. Four different constructor methods are defined to create an `OutputStream` object. However only one of them `OutputWriter(dirName, fileName)` is used extensively in our code. It allows us to instruct a program in what directory should a log file be created. The method `getFileStream()` assigns the output stream to a file. And `printLine()` duplicates the functionality of standard Java `printLine()` method using the file stream indicated earlier in this class. `Logger` class for example is a special use case of an `OutputWriter`. It provides `synchronized` access to a `printLine()` method, allowing every other object of the program write into the same log file.

6.3 Main Algorithm

Take a look at the Figure 4. This is a UML activity diagram, which shows the approximate steps undertaken by the crawler while parsing a single Web structure.

On launch of the program user has to deliver an input string. This may be either Web address of the page, from which the information extraction will start or any sentence for a Google query. In the latter case Google's response page will be taken to initialise the data discovery. Then the quantity of running bots will be checked. And if it has not reached its upper bound a new bot will be started. At start each bot receives a unique name and login information for a database. After that a database connection will be designated to this bot.

Every bot in our program has its own database connection. Thereby we avoid generation of possible bottlenecks, during the database access by bots. The bots will compete with each other only at the DBMS level, not at the Java programming level contending for a chance to put their information into the database.

After all of these preparation steps have been done, the program enters into its main loop. A bot will proof if it was asked to stop operating. Since it operates continuously until it

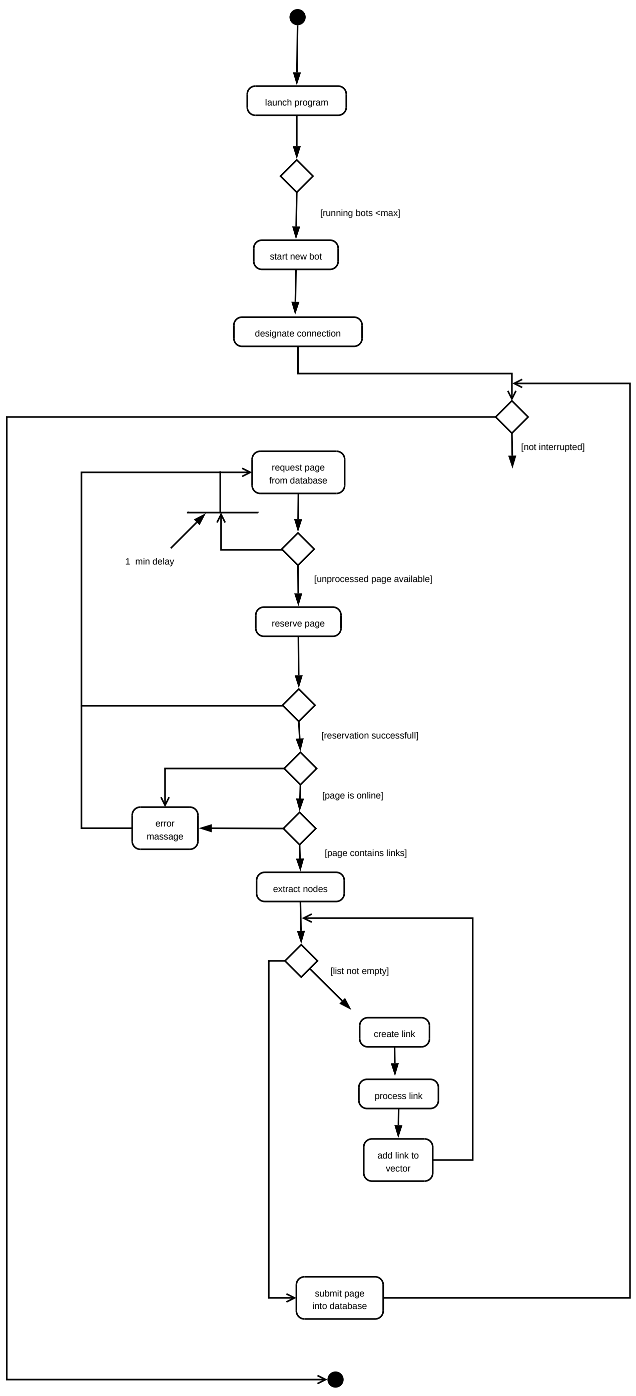

Figure 4: Activity Diagram

receives a stop signal from the bot manager. After successfully passing the check it sends a request to the database, that it is ready to analyse a new unprocessed page.

Since we have many bots running simultaneously a special method was implemented to support immediate page reservation by a bot. It prevents the program from assigning the same page for parsing by more than one bot. If the page was successfully reserved we proceed to the next two checks, otherwise a bot will wait for one minute and try to request a page again.

The page has to be accessible online and has to contain hyperlinks in its source code to be parsed. If the page meets both conditions it comes to the next phase where all HTML tags passed through our filter will be extracted and saved as abstract nodes into a list. Elsewise an error message will be printed out and a bot will be sent back to its requesting activity.

After we obtain a list with nodes a program enters into its second important loop. As long as the list would not be empty a new `Link` object will be constructed from each node, analysed, and saved into a vector. This is the other data structure, intended for storage of handled links. On the last stage of program's cycle the page will be updated with some additional information and along with its links vector submitted to a database.

6.4 Pseudo Code

In this section we will explain the functionality of four vital and valuable methods of our program. These are `Bot.run()`, `Database.requestPage()`, `Page.extractLinks()`, and `Database.submitPage()`. We will describe them in order of their occurrence in the program. First operation of each method will be presented by means of a pseudo code and then method's implementation features will be discussed.

6.4.1 Bot Execution

Default Java class `Thread` was extended to implement a `Bot`. The standard method `run()` was overridden to provide the functionality of a single bot. Let us look at the pseudo code, which briefly states the main steps a bot takes to process a Web document.

```
while (bot B is not interrupted) {
request new page P from the database D

if P was successfully requested {
if (P is online) AND (P contains links) {
extract links from P
} else { // page is not online or does not contain links
set current time as last processing time for P
}
submit P into D
destroy P
} else { // no page was requested
B wait for 1 minute
}
}
close D
```

It is easy to confront these operations with the first outer loop at the activity diagram. Some particular properties of this algorithm should be noted. Current presence of a document in the Web is controlled over boolean method `isOnline()`. It returns `true` only if a HTTP status code of the page in under 400, this means we will try to work with the page in case error neither from client nor server side is registered.

Next method `containsLinks()` checks if a Web resource may contain links to other resources. It bases on the MIME content type value found in the header of the Web resource. The bot will proceed with links extraction if the document is of HTML or XML subtype.

Other interesting aspect of the algorithm is setting current time as last processing time for this resource if it was not online or did not contain links. The reason for this is that a page in the database is considered unprocessed until it has `NULL` values in its `ptime` and `pbot` fields. Therefore a Web resource misses its chance to be parsed, in case it was not accessible over an error, since we go through the table "Resources" only once using the top-down approach.

6.4.2 Page Request

Method `requestPage()` returns a new unprocessed page from the database. Simplified version of its algorithm is presented below in pseudo code form.

```
request set S of unprocessed pages from the database D
define rows number N in S

if N < MAX_ROWS_NUM
set row index I to 1
else
choose random I from MAX_ROWS_NUM range

take row R with index I from S
if (R was successfully taken from S) {
create new page P from the data of R
reserve page P for bot B

if P was successfully reserved
return P
else return null

} else return null
```

The method may be divided into two logical parts. These are request and reservation of a page. Watch closely at the beginning of the code. To improve performance of parallel bots' access to the new data a special mechanism was invented. We request not a single resource from the database, but a set of them limited with a `MAX_ROWS_NUM` constant. Every bot will choose a random row from the result set returning a number of unprocessed pages. However at start while the delivered set's size has not reached the `MAX_ROWS_NUM` constant's value the bot will always take the first row.

In the second part of the method each page will be reserved for the current bot. The reservation procedure occurs right after the actual page request, when a page is stamped with the bot's name and current time given. This helps to avoid classical transaction problems. The probability that two or more bots will request and process the same resource is extremely low. Even at the moment of reservation the DBMS checks that no other bot has already booked the page in the time interval between actual resource request, values computation and setting by a current bot. This happens since the `WHERE` clause of the SQL statement permits page record's update only in case this entry did not have `ptime` and `pbot` parameters set yet.

6.4.3 Links Extraction

Page parsing and links extraction takes place in the `extractLinks()` method. Let us consult the pseudo code below for detailed information on this procedure.

```
define new parser R for url of the page P
define new filter F to pass attributes "HREF" and "SRC"
using R extract all nodes that match F into list T

for each node N in T {

create new link L
save attribute "HREF" OR "SRC" into url field U of L

if (U is not null) AND (L is http like link) {
obtain host, file, type, and charset fields of L
create rid value from U
set current time as discovery time of L
set current bot as discovery bot of L

if (L exists in vector V)
increment pop counter
else {
add L to vector V
sort V alphabetically according to U
}
}
destroy L
}
```

```
sort V alphabetically according to host name
obtain source code of P from R
```

First of all a new `Parser` object will be defined from the page's URL value. Then the "HREF" and "SRC" parameters are given to the `Filter`. When the nodes are extracted from the page's source code we enter into the second inner loop of the program. These steps may be also found on the UML activity diagram. At this point we extract node's "HREF" or "SRC" attribute. The obtained value will be put into the `URL` field of a new link object. This is the major parameter of a `Link` instance. Storage of found URL is the general purpose of a link.

At the next step will be proven that found URL is not empty and is of HTTP or HTTPS protocol type. Then the wide spectrum of different operations will be applied on this link. Document's header will be parsed for resource type and character set values. After that host name of the URL will be gained and possible URL's reference part cut off. It is removed while anchored URL value point to the same Web document.

Later resource identification RID is generated using the `createHash()` method. It composes a unique MD5 digest from the URL of the link. Next current time and bot's name are saved into `dtime` and `dbot` fields of the link. Now we have to check, if link with the following URL already exists in page's vector. If this is the case a `pop` counter is incremented, otherwise the link will be added into vector.

We suppose, that keeping vector in alphabetical order speeds up the check of link existence in a vector. That's why `URLComparator` class methods are used to sort the vector after every link's addition. After all operations are done the `Link` object must be cleaned up, to be successfully destroyed by the Java garbage collector.

When the loop is left all nodes are processed and vector is full of effective links. The page treatment ends up with one more sorting of the vector. But this time according to host name. The purpose of the latter ordering will be made clear in the next section. At last page's source code is obtained from the parser. At this point the source may contain slight modifications made by the parser. End tags will be inserted where appropriate.

6.4.4 Page Submission

Method `submitPage()` is the second valuable procedure executed by a program at the end of page's processing. Major steps of the method can be found in the following pseudo code.

Pay attention to the first line of the method. Automatic commit is the default setting for the database. It means every SQL conversion to any table will be treated as a single transaction. And changes caused by any individual statement will be automatically committed into the database. Page submission is a comprehensive procedure, that includes a lot of insert and update operations affecting all tables of the database. We have turned the automatic commit off for this method, while the whole set of SQL activities submitting a Web resource should be considered as a single transaction.

```
try {
turn automatic commit off for the database D
create temporary link T

for each link L in vector V of given page P {
insert L values into table ''Hosts'' of D
insert L values into table ''Resources'' of D
insert L values into table ''Links'' of D
} // end of for loop, vector is inserted

update P values in table ''Resources'' of D
commit changes to D

} catch and print out SQL exception
roll back changes in D
```

Each piece of link's data in a vector passes through an SQL insert statement. New information should be inserted into "Hosts", "Resources" and "Links" tables of the database. To simplify data insertion and error tracking processes the method was divided into four submethods: `operateHost()`, `operateResource()`, `operateBond()`, and `operatePage()`. Each submethod interacts with only one of the tables. Methods `operateHost()` and `operateBond()` insert data into "Hosts" and "Links" tables, respectively. To prevent confusion a different name was chosen for the method operating with "Links" table. Since it inserts information about a bond subsisting between father and child Web resources, we have labelled it `operateBond()`. Both methods `operateResource()` and `operatePage()` are used to communicate with the table "Resources", whereas `operateResource()` is called on every iteration in the loop and `operatePage()` applies last changes to the page object itself and updates its record in the database.

This is the only pseudo code snippet, where an exception handling block is shown. We depict it since it is an important feature of the method. Because we want to avoid input and presence of inconsistent data in the database, all changes made to the tables will be rolled back in case any SQL error occurs during the insert or update operation. This means all information collected in `Page` object will not reach the database and be lost. None of the tables will contain data about links or bounds found. And since the bot reserves a Web resource at the moment of request, no more retries will be made to process this document.

Currently `operateResource()` is the only component method of `submitPage()` capable of an independent exception handling. This functionality was added to identify method's attempts to insert an already existent document into the table "Resources". It may happen if the document really exists in the table, or in case of an MD5 collision, when an incorrect RID is generated. If this situation occurs an error message is saved in a log file and the *local* roll back operation is executed. This means record in table "Hosts" will be deleted, but the main `submitPage()` procedure will continue for other links remained in a vector.

6.4.5 Host Processing

For better understanding of `operateHost()` main idea take a look at another pseudo code fragment. A very elegant mechanism was developed to effectively operate upon the "Hosts" table.

```
if (host name of L equals to host name of T)
assign to L host id of T
else {
if (host name of L exists in hosts table in D) {
assign to L host id obtained from D

else { // host name of L was not found in hosts table
insert host name of L into hosts table
acquire host id from D
assign acquired host id to L
}
change host name and id of T
}
```

The main aim of the method is to assign a host identification number to every host name saved in the link object's field. The host identification is required to make further data inserts into other tables of the database. First of all note a temporary link object, which is intended to hold host identification and name values of the last inserted link.

Before a new data will be inserted into "Hosts" table it should pass two complementary checks. We remember that our links vector was alphabetically sorted according to the host name at the end of `extractLinks()` method. After alphabetical ordering all links with equal host names will be disposed one after another. At the beginning of the method every link is compared with the temporary link. And in case we find out that the current link we are processing has the same host name as the temporary, we can get the host identification number right away. Excess calls to the database are saved when the program has vector entries with equal host name values. The database is loaded only once at the moment of first link's registration.

However if temporary link's host name is different to the one being analysed, we are supposed to put this link's host into the database. But at first we should check, if the database already has an entry with the following host name. If this is the case and this host already exists in the "Hosts" table, the program merely has to obtain the identification of the host and assign it to the link object.

Only when two preparatory procedures are done and the program has recognized, that the host was not equal to the temporary link's value and was not found in the "Hosts" table a new record is made. Pay attention to change of temporary link's values that occur if the program had to search for the host name or insert an absolutely new entry into the table. After reassigning the temporary link's attributes we are ready to process the next link from the vector.

7 Implementation Features

7.1 Threads

Usage of threads is one of the fundamental features of our program. For the sake of efficiency we allow simultaneous data extraction and storage by a number of bots. However only basic threads functionality was selected for our implementation. We have not specially adjusted thread priorities and made use of groups, ids and threads of type daemon. Every thread of execution in our implementation uses the simple variant of allocation. The `Bot` class is declared to be a subclass of `Thread` and overrides the standard `run()` method. It was possible to use a constructor where the stack size for the thread could be set. But since stack's space reservation by the Java Virtual Machine is highly platform dependent and there exist some platforms where this value is merely ignored, we have decided not to rely on this setting to reserve a buffer for a thread based on the estimated capacity requirements calculated in Section 5.3.

7.2 Message-Digest Algorithm 5

Another point we should mention is selection of MD5 hash function for generation of resource identifications in `createHash()` method. It is a widely used algorithm, which generates 128-bit digest values. We have applied MD5 to our project, although many weaknesses of the function were found recently and it is not recommended to rely on its stability in serious security implementations, where SHA-1 or other algorithms should be chosen. As long as we use cryptographic properties of the hash function not for security purposes, but for creation of identification values, we assume, that it is not drastically if some collisions occur.

Duplicate identification value would prohibit resource insertion into the database. It was also possible to implement SHA-1 algorithm, but we have not decided for it, because of its increased 160-bit digest size. Usage of SHA-1 would brought additional 6.023 KB space load for every average Web resource containing 256 hyperlinks, while we would require extra 8 bytes for storage of each SHA-1 identification value.

7.3 Result Set

Cooperation with the database in our program occurs by means of built-in Java interface `ResultSet`. It presents a table returned by a database on an SQL query. One important concept is that `ResultSet` object manages a cursor which is used to move between the rows in the result table. After initialisation the cursor is usually set before the first row. The logical method `next()` shifts the cursor forward to point to the next row. Since it returns `false` when the result table contains no more visited rows it is often used in case simple iteration through the whole table is required.

But potentialities latent in the `ResultSet` interface are very wide, and we suppose `ResultSet` may be realised as an artificial buffer a Java program utilises to communicate with the database. `ResultSet` objects may be constructed to support updatability and scrolability [IBM05], this means it is possible to set an absolute cursor position, move backward through the table, and use iteration patterns different from the standard top-down movement approach. Updatability allows to alter values in the ResultSet object and after calling an `updateRow()` method submit changes to the underlying table in the database from where the information was derived. It is enough to insert rows only into the `ResultSet` object and they will be automatically reflected to the source table in the database. The deletion of rows is also available for an updatable `ResultSet`. It is essential to mention that `ResultSet` is cleaned up and ready to operate upon new data every time its corresponding Statement object is re-executed.

For the purposes of our program a `ResultSet` object was made using `TYPE_SCROLL_INSENSITIVE` and `CONCUR_UPDATABLE` constants. The first parameter permits a cursor to scroll to a distinct row in the result table, but changes made to the underlying table will not be visible to the cursor itself. The second constant indicates that created `ResultSet` object will be updatable.

7.4 Prepared Statements

Usage of `PreparedStatements` is another interesting aspect of our project, which allows to increase performance of the database operation. The general idea of a `PreparedStatement` is that a SQL statement could be made ready before its execution. This is especially effective if the statements have identical structure and are intended for frequent calls to a database.

At the moment of statement's allocation positions of future parameters are marked with question marks. And at the time of actual execution they must be replaced with real values. Special setter methods indicating type of an argument should be used to specify an input value. Most of `Database` class methods are founded on `PreparedStatement` concept. Another helpful property of `PreparedStatement` interface is its ability to retrieve any auto-generated keys created by the last statement's operation. The keys are made accessible to the program by means of `getGeneratedKeys()` method, which is also widely employed in our project.

8 Conclusion and Future Work

A large amount of work has been done to produce a program capable to explore, analyse, and store effective information about Web structures. A DB2 database was designed and deployed for filing purposes. Special attention has been paid to implementation of multi-threading environment and optimised cooperation with the database. During the research time five packages consisting of eleven classes have been developed and set up.

Based on "HREF" and "SRC" attribute filters our data extracting technique fails to discover other possible containers of hyperlinks in a Web page, such as the Java Script tags, for example. But we reckon that the vast majority of hyperlinks are indicated under these two tag attributes.

Major extraction's tasks in our program are accomplished by means of HTML Parser. But nowadays high emphasis is placed on information retrieval using machine learning methods. Since crawler's implementation has modular structure it might be possible to include these methods into future revisions of the program as necessary.

Our program does not verify if the HTML/XML source code contains malformed tags. However, the heuristics built into the tag parsing process of the HTML Parser will insert terminating nodes where end tags are required. But no other validation is done on the page [OR05].

Web document's source code verification combined with subsequent correction might be done before the actual data extraction. This would remove one of possible error constituents and extend crawler's productivity. For example the JTidy, which is an open source Java package, might be integrated into our program for Web document's syntax clean up and repair purposes [RQPL01].

Source code of a Web page is saved without international characters. It is known that Java compiler uses UTF-16 as its default encoding. And according to feedback provided by HTML Parser's developers, the Parser also delivers Unicode version of the source code. The database created in the DB2 environment from the very beginning was set up to support UTF-8, an alternative representation form of Unicode. We suppose the international symbols are lost over usage of `setObject()` method in Java, during conversion of Java string with a source code into `CLOB` type of the DB2.

One possible solution to this problem might be a switch to National Character Large Object `NCLOB`.[1]Whereas common `CLOB` is limited to storage of one-byte symbols, `NCLOB` was specially designed to keep international text data [KE01].

A lot of other realisation questions were left open for further examination. For example it is noticeable that most memory used for storage of an average Web structure is allocated by its source code. Perhaps it would be more efficient to apply compression techniques to this attribute and make it logging free. In DB2 the default settings might be changed by means of `compact` and `not logged` commands.

Thorough and stress tests have to be done upon our program. They will help to reveal and repair code errors. The testing procedure will also provide basis for tuning capabilities and enhancements in program's main algorithm. The database might be also ad-

[1]In DB2 this data type is called `DBCLOB`, which stands for Double-Byte Character Large Object.

justed after the simulation procedures. It could be a good idea to apply indices on some columns in the tables to increase the database responsiveness upon crawler's operations.

Numerous improvements can be made to the manner of data processing. The classical producer-consumer model might be applied to bots cooperation procedure, where bots would work on the data in couples. One could extract the valuable information and the other send it to the database for storage. The model could be adjusted according to test results for the best performance. For instance a number of exploring bots could be designated to one storing. Another operation's mode might be suggested where groups of two or more bots would use the same connection to a database.

The full potential of `ResultSet` interface was not used. The extracted data might be more effectively operated by means of row insert, update and delete statements in the retrieved set of information and not by appealing to the database directly.

The data structures selection for the internal storage of extracted information in our program could be thought over. Another constructions like hash tables or binary trees might be chosen for holding the bots and the set of generated links. Binary tree would be probably a better alternative for keeping links as a vector, while it performs elements' sorting faster.

The program also lacks of Graphical User Interface (GUI). Visualisation would improve crawler's administration, clearness, and usability.

As long as the Web contains over 12 milliard sites [GS05], it might be a better intention to focus on parsing of a definite type of documents, or to select a smaller scope for information extraction. Operating on a finite number of Web resources, e.g. within the limits of a single organisation or country, would also reflect on more precise final results.

In conclusion the most interesting and important research field, that might be based on our work, is certainly the gathered data analysis. A large number of different methodologies could be applied to study dependencies between Web structures. Special programs like the TouchGraph Google Browser [To0], representing the strength and depth of intersite relationships in a graphical way may be written.

Acknowledgements

Most of all we would like to thank our scientific adviser Mr Christopher Popfinger for contributing his share to the work. We gratefully acknowledge his patience, critical observations, and suggested ideas. We also thank the HTML Parser developers for creating this powerful Java package extensively used in our program, and particularly Derrick Oswald for emphasising Parser's default behaviour. We kindly thank Iļja Nevdahs for explanation of Java theoretical basis and supplied examples of logging classes, Vadims Adamovičs for his remarks on program's style and countenance, and finally Dmitrijs Abaļenkovs for drawing sketch versions most of the UML diagrams.

Appendix: SQL Statements

Tables Creation

```
CREATE TABLE hosts (
  hid BIGINT NOT NULL PRIMARY KEY
   GENERATED ALWAYS AS IDENTITY (START WITH 1, INCREMENT BY 1),
  hname VARCHAR(64)
)

----- ----- ----- ----- -----

CREATE TABLE resources (
  rid CHAR(32) NOT NULL PRIMARY KEY,
  url VARCHAR(1024),
  rtype VARCHAR(128),
  charset VARCHAR(40),
  dtime TIMESTAMP,
  dbot CHAR(16),
  ptime TIMESTAMP,
  pbot CHAR(16),
  source CLOB,
  hid BIGINT NOT NULL,

  FOREIGN KEY (hid)
    REFERENCES hosts (hid)
)

----- ----- ----- ----- -----

CREATE TABLE links (
  frid CHAR(32) NOT NULL,
  rid CHAR(32) NOT NULL,

  FOREIGN KEY (frid)
   REFERENCES resources (rid)

  FOREIGN KEY (rid)
   REFERENCES resources (rid)
)
```

Tables Initial Infill

```
INSERT INTO hosts VALUES (
  DEFAULT,
  'www.google.com'
)

----- ----- ----- ----- -----

INSERT INTO resources VALUES (
  'EE18BAD53F741BA1C24D3F992A10E411',
  'http://www.google.com/index.html',
  'text/html',
  'UTF-8',
  CURRENT TIMESTAMP,
  'Naiad',
  DEFAULT,
  DEFAULT,
  DEFAULT,
  1
)

----- ----- ----- ----- -----

INSERT INTO links VALUES (
  'D41D8CD98F00B204E9800998ECF8427E',
  'EE18BAD53F741BA1C24D3F992A10E411'
)
```

References

[Ap0] Simple Object Access Protocol – SOAP, Version 2.3.1. http://ws.apache.org/soap/, Accessed 2006-03-12.

[CGJP03] Peter Charles, Nathan Good, Laheem Lamar Jordan, and Joyojeet Pal. How Much Information 2003? Technical report, University of California at Berkeley, School of Information Management and Systems, 2003. http://www.sims.berkeley.edu:8000/research/projects/how-much-info-2003/index.htm, Accessed 2006-03-12.

[Day99] Rebecca Day. Web Hounds. Technical Report 3, Think Research, 1999. http://domino.research.ibm.com/comm/wwwr_thinkresearch.nsf/pages/webhounds399.html, Accessed 2006-03-12.

[EN02] Ramez A. Elmasri and Shamkant B. Navathe. *Grundlagen von Datenbanksystemen*. Pearson Education Deutschland GmbH, Martin-Kollar-Strasse 10-12, D–81829 München/Germany, third edition, 2002.

[Etz96] Oren Etzioni. The World-Wide Web: Quagmire or Gold Mine? In *Communications of the ACM*, volume 39, November 1996.

[Eu0] Reference Web Site. http://www.euronews.net, Accessed 2006-03-15.

[FGM$^+$99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. Network Working Group, June 1999. Request for Comments: 2616.

[Fin05] Finkernet Marketing. *Basics of Search Engine Marketing*, May 2005. E-Book, http://www.finkernet.com/sem/, Accessed 2006-03-12.

[Fla02] David Flanagan. *Java in a Nutshell, Fourth Edition*. O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, March 2002.

[GS05] Antonio Gulli and Alessio Signorini. The Indexable Web is More than 11.5 billion pages. Technical report, Università di Pisa, Dipartimento di Informatica and University of Iowa, Computer Science, January 2005.

[IAN05] IANA. *Character Sets*, January 2005.
http://www.iana.org/assignments/character-sets,
Accessed 2006-03-15.

[IBM05] IBM Corporation. *DB2 Information Center*, January 2005.
http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp,
Accessed 2006-03-12.

[Ju0] Information on Security Vulnerability:
POP3 Content-Type Name Too Long.
http://www.juniper.net/security/auto/vulnerabilities/vuln2446.html,
Accessed 2006-03-12.

[KB00] Raymond Kosala and Hendrik Blockeel. Web Mining Research: A Survey. Technical report, Katholieke Universiteit Leuven, July 2000.

[KE01] Alfons Kemper and Andre Eickler. *Datenbanksysteme: eine Einführung*. Oldenbourg Wissenschaftsverlag GmbH, Rosenheimer Strasse 145, D–81671 München, fourth edition, 2001.

[Moc87] P. Mockapetris. *Domain Names – Implementation and Specification*. Network Working Group, November 1987. Request for Comments: 1035.

[Moo04] Alfred Moos. *Datenbank-Engineering: Analyse, Entwurf und Implementierung objektrelationaler Datenbanken – Mit UML, DB2-SQL und Java*. Friedr. Vieweg & Sohn Verlag/GWV Fachverlage GmbH, Wiesbaden, third edition, February 2004.

[OR05] Derrick Oswald and Somik Raha. *HTML Parser, Version 1.6 (Integration Build 2005-11-12)*, November 2005. http://htmlparser.sourceforge.net/, Accessed 2006-03-12.

[Ri0] The Bat! IssueTracker, ID 2956:
The Bat sometimes generates too long MIME headers.
https://www.ritlabs.com/bt/print_bug_page.php?bug_id=2956,
Accessed 2006-03-12.

[RQPL01] Dave Raggett, Andy Quick, Gary L. Peskin, and Sami Lempinen. *JTidy, Version 1.6 (2001-08-01)*, August 2001. http://jtidy.sourceforge.net/, Accessed 2006-03-12.

[To0] Google browser.
http://www.touchgraph.com/TGGoogleBrowser.html,
Accessed 2006-03-15.

[Wes01] Douglas Brent West. *Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ, second edition, 2001.