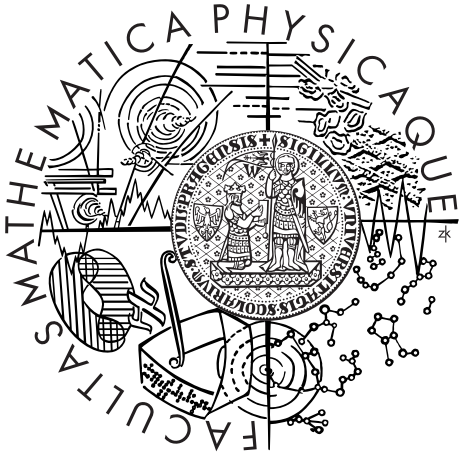
Charles University in Prague
Faculty of Mathematics and Physics

**MASTER THESIS**




Peter Hlísta

# Analysis of Real-World XML Queries

Department of Software Engineering

Supervisor of the master thesis: Doc. RNDr. Irena Holubová, Ph.D.
Study programme: Software Systems
Specialization: Software Engineering

Prague 2014

I would like to thank my supervisor Doc. RNDr. Irena Holubová, Ph.D. for helpful advices, reviews of thesis working versions, grammar corrections.
I would also like to thank my brother Igor for a lot of grammar corrections.

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague date 30.7., 2014 Peter Hlísta

Název práce: Analýza reálných XML dotazů

Autor: Peter Hlísta

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: Doc. RNDr. Irena Holubová, Ph.D.

Abstrakt: Účelem této práce bylo shromáždit a analyzovat běžně používané XQuery programy.
Ke sběru dat z internetu je nejčastěji využíván program zvaný crawler. Součástí této práce byla analýza různých crawlerů a výběr nejvhodnějšího z nich. Tento crawler byl následně upraven tak, aby nevytěžoval servery, sbíral správná data a bylo možné jeho činnost pozastavit. Před započetím sběru dat ještě zbývalo určit, kde bude vhodné začít a jak dlouho by celý proces měl trvat. Po skončení hlavní části pak ještě zbývalo data vyčistit a ověřit jejich validitu.
Předmětem následných analýz bylo použití dotazovacího jazyka XQuery a výskyt jeho gramatických symbolů. K provedení těchto analýz byla použita kombinace XPath dotazovacích výrazů a XML reprezentace XQuery programů. Pro vytvoření zmíněných reprezentací byl použit nástroj XQConveror.
Hlavní přínosem této práce jsou vlastní sběr dat a také první analýza běžně používaných XQuery programů.

Klíčová slova: analýza, reální XQuery, XQConverter, crawler

Title: Analysis of Real-World XML Queries

Author: Peter Hlísta

Department: Department of Software Engineering

Supervisor: Doc. RNDr. Irena Holubová, Ph.D.

Abstract: The aim of this master thesis was to gather and analyze the real-world XQuery programs.
The data gathering process is performed using the crawler. The thesis contains analysis of different crawlers and the most suitable crawler was chosen. The crawler was modified, so that it did not overload servers, gathered the right data and was able to pause.
Before the data gathering we analyzed where to start gathering and how long should it took. When the data was gathered, they needed to be cleaned and validated. The subjects of the analyses were use of the XQuery language and occurrences of XQuery grammar symbols. Combination of the XML representation of XQuery programs and XPath expressions for querying this representation was used to perform these analyses. XQConveror was used to create this XML representation.
The main contributions of this thesis are the gathered data and the first real-world XQuery programs analysis.

Keywords: analysis, real-world XQuery, XQConverter, crawler

# Contents

# Introduction

The Extensible Markup Language (XML) [62] is widely used language nowadays. It defines how to encode documents in a format that is both human-readable and machine-readable. Although the design of XML focuses on documents, it is widely used for the representation of data structures.
There are several ways to query an XML data. One of them is using XML Query language (XQuery) [56]. It was created by W3C workgroup exactly for this purpose.
There is no known analysis of the real-world XQuery data yet, so the aim of this thesis is to gather real-world data and to analyze them. This process is described in this thesis. The results of the analysis can be used for possible optimization of existing XQuery evaluators and can help with development of newer versions.

## Outline

In this thesis we analyze real-world XQuery data. XQuery data are queries that have complexity of programs so XQuery programs will be referred to them.
The first chapter contains definition of terms and technologies used in this thesis. The technologies mostly cover the Internet technologies, because it is the place where we took the real-world XQuery data.
The second chapter is introduction to several programs that helped us during the process of gathering and analyzing XQuery data. The XQuery analysis and difficulties of the analysis are mentioned there, together with the reasons why it is not a simple task.
The third chapter contains introduction of various data gathering programs and their classification according to the attributes. We described demands placed on the right gathering program. One of these programs is chosen at the end of the chapter.
The fourth chapter describes the preparations for the data gathering, process of gathering the data and difficulties during this process. This chapter also contains settings and editing of the program chosen for the data gathering.
The main goal is to gather as many data as possible that are valid and represent the group of common XQuery programs used for the real-world queries.
The fifth chapter starts with cleaning of the gathered data from data that are not XQuery programs. After the data is cleaned, they are prepared for the analysis. The analysis contains the frequencies of grammar symbols, uses of variables, functions, paths, etc. The construction of results from these XQuery queries is also mentioned.
The last chapter is the summary of this work and the evaluation of gained results. It presents accomplishment of the goals set in the beginning and possible areas of this thesis that might be interesting to develop.

# 1. Technologies and Terms

This thesis is focused on crawling of the Internet. This part describes basic terminology to make the text more comprehensible.

## 1.1 Internet Technologies

The Internet is a worldwide system of interconnected computer networks. This means the computer networks are connected throughout the world.
Computer network is a collection of interconnected nodes. These nodes are represented by computers, switches, routers, repeaters, etc. There are two main ways of how the nodes are connected - ethernet (for example twisted pair cables, optical wires, etc.) and wireless ethernet.
As it was said these networks are interconnected, we can say, that the internet is network of networks.
To be able to send the data using those connections, the communication protocols are defined. The standard protocol for the internet network is TCP/IP [36]. These are used in this master's thesis.

### 1.1.1 TCP/IP

The Internet protocol suite (TCP/IP) is computer networking model used by the Internet. The first two protocols were Transmission Control Protocol (TCP) and Internet Protocol (IP). TCP provides reliable, ordered and error-checked delivery of a stream of octets between programs running on computers connected through network. IP is protocol for relaying datagrams across network boundaries.
Two most important IP protocols are Internet Protocol version 4 (IPv4) [36] and Internet Protocol version 6 (IPv6) [57].
IPv4 uses a 32-bit long network address. The format of this address is X.X.X.X, where each X is a number from interval 0 to 255. 1 byte (8 bits) from a 32-bit address is thus represented by each X. The main disadvantage of IPv4 is its limited number of combinations - $2^{32}$ = 4,294,967,296, which is below the number of devices requiring IP address.
IPv6 uses a 128-bit address. Its form is Z:Z:Z:Z:Z:Z:Z:Z. Each Z has the following format - YYYY, where Y is hexadecimal number (0-F). There are some rules to shortening the strings, but they are irrelevant for the purposes of this thesis. Relevant is the fact, that one physical machine can have more addresses, e.g. one IPv4 and one IPv6 address.
A Port is associated with an IP address of the host (computer), as well as the type of protocol used for communication. Port is a number from 0 to 65,535. In TCP port 0 is reserved. In User Datagram Protocol (UDP) [36] it means 'no port'.
UDP provides unreliable, unordered, no duplicates protection delivery of datagrams (messages) to other hosts on an IP network.

### 1.1.2 DNS

The Internet maintains two principal namespaces: the domain name hierarchy and the IP address system. The Domain Name System (DNS) [51] maintains the domain namespace and provides translation services between these two namespaces.
The Domain Name, also known as host name, is a string in human readable form, so that it is easier to remember by human.

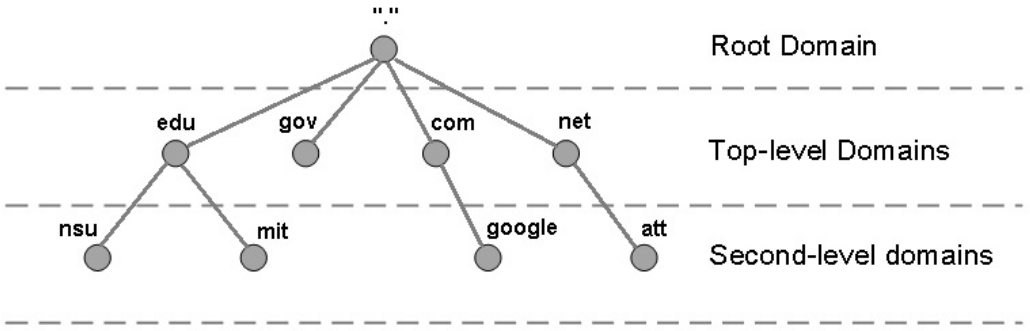



Figure 1.1: DNS hierarchy [15]

DNS is distributed throughout the world by DNS servers. The DNS servers work in a hierarchical manner. Figure 1.1 shows that on the top of the hierarchy there is root domain '.'. Root domain is an unnamed level. Beneath the root domain there are top-level domains. These indicate country, region or the type of the organisation, e.g. com, net, cz, sk. Next in the hierarchy there are second-level domains (names registered to an individual or organisation for use on the Internet (e.g. microsoft.com). Under second-level domains are only subdomains. A request for specific translation from DNS name to IP address is executed using recursion, what is time and source consuming. Besides authoritative DNS servers there exist also Caching name servers in order to make the process faster and less source consuming.
The authoritative DNS server gives authoritative responses considering the administrated domain. This response includes time to live (TTL). This information is used by cache servers. They will cache that record for the time (in seconds) specified by the TTL.
Standard port for DNS is 53.

## 1.2 The Web

The World Wide Web (WWW) [29], also known as the 'Web', is a system that operates over the Internet. It consists of web pages which user can view using the web browsers.
Web pages are hypertext documents. They are written in the HyperText Markup

Language (HTML) [41].
Every web page on the Internet has its uniform resource locator (URL) [61] also called web address or link. URL is character string used as reference to a resource. Web pages consist of two main parts: body and head. The body contains the web page structure and the header contains meta information. One of the meta information is meta tag for crawlers.
This meta tag's task is to help the crawler identify the content of the web page.

### 1.2.1 HTTP

The Hypertext Transfer Protocol (HTTP) [55] is a protocol to exchange or transfer hypertext (structured text with links). It uses Status codes for every request. Their form is XZZ, where is X is a number from 1 to 5 and ZZ is a two-digit number. For the purposes of this thesis the relevant codes are 2ZZ - Success, 3ZZ – Redirection, 4ZZ – Client Error (especially 404 Not Found) and 5ZZ Server Error.
HTTP request mentioned in the thesis (or program) are GET, POST and HEAD.

**HTTP GET**
: HTTP GET is designed for getting the data from specific server.

**HTTP HEAD**
: HTTP HEAD corresponds to HTTP GET but without the response body. This way we find out Multipurpose Internet Mail Extensions (MIME) [52, 53] type of specific file

**HTTP POST**
: HTTP POST is a request that the server accepts the entity enclosed in the request. For example this entity can be: an annotation for existing resources, a block of data as the result of submitting a web form, etc.

MIME is an Internet standard. It encapsulate files so they can be send through e-mails. Nowadays it is widely used in the HTTP communication. In header it contains information about file. Content type of file is one of the information.
HTTP has secured version: Hypertext Transfer Protocol Secure (HTTPS). Pages with this protocol mostly needs login and password to show useful content.
Standard port for HTTP is 80.

### 1.2.2 Search Engine

To be able to find some information on WWW in reasonable time we need search engines. They are built out of several components:

**crawler**
Crawler is also known as **robot**, **spider** or **bot** and we will reference to it as **crawler**.

It goes through IP addresses and downloads web pages assigned to them (this is running in multiple instances/threads).

**indexer**
Indexer categorizes and adds the links to databases with their related keywords (terms).

**presenter**
Presenter gets the search queries from uses, searches the database and provides the search results.

### 1.2.3 Characteristics of The Internet

The Internet is an international network of networks that consists of millions of private, public, academic, business, and government networks. As such it has some interesting characteristics. Some of them are:

**Dynamic Data**
The content of the Internet is dynamic. This is caused by many facts like pages can be added or removed from servers, many sites with dynamic content, etc. This is the reason why the data collected in different times mostly differ.

**Incorrect Data**
Some documents found on the Internet may contain incorrect data.

**Invisibility for Crawlers**
Web crawlers cannot naturally search the whole Internet. Here are some of the facts why it is not possible:

- For some documents there are no references from other web pages.
- Some documents can be acquired only with login information (user name and password).
- There are advices on web domains for the crawlers what files and directories should not be crawled.

**Spider Traps**
A spider trap is a set of web pages that can intentionally or unintentionally cause a crawler to make an infinite number of requests [60]. Poorly constructed crawlers can crash because of this. One of the examples is a

dynamic page with calendar. Crawler should avoid getting into these spider traps.

## 1.3 XML Family of Languages

There are XML-related languages developed to help work with XML data. There are three mentioned in this thesis besides XQuery.
XML Path Language (XPath) [45] is a query language for selecting the nodes from an XML document. XPath is going to be used for obtaining analysis from XML representation of XQuery program.
Document Type Definition (DTD) [10] is used to define the legal building blocks of an XML document.
XML Schema Definition (XSD) [58, 40]specifies how to formally describe the elements in an XML document.
Extensible Stylesheet Language Transformations (XSLT) [38] is a language for transforming XML documents into other XML documents.

## 1.4 Open Source Licenses

These are open source licenses acronyms used in this thesis:

**Apache** [2]
Apache Software License.

**Apache 2.0** [2]
Apache Software License 2.0.

**BSD** [4]
Berkeley Software Distribution License.

**Egothor 1.0** [42]
Egothor Software License version 1.00.

**Egothor 2.0** [37]
Egothor Software License version 2.00.

**GPL** [12]
GNU General Public License.

**GPLv2** [12]
GNU General Public License v2.0.

**LGPL** [12]
Lesser General Public License.

**MIT** [21]
Massachusetts Institute of Technology License.

## 1.5 Programming Languages

Programming languages used or mentioned in this thesis are C++ [7], C# [8], Java [17], PHP [25] and Python [26].

# 2. Related Work

The first chapter described terms and technologies used in this thesis. In the second chapter we will analyze existing programs for analysis of the data. The data includes XML data, XQuery data, etc.

## 2.1 Existing XML Data Analysis

In Report [50] are extensive analyses of real-world XML data. These analyses produced statistics results:

- The first set, global statistics, consisted of properties of XML data such as number of elements of various types, number of attributes, paths, depths and portion of text in documents. In XSD and DTD the depths are counted for each global element used in the sample XML documents as a root element, for recursive elements we take into account their lowest level(s) and an infinite level for expressing the recursion.
- The second set, depth statistics, consisted of depths per each category because maximum and average depth are known from global statistics. In XSD cannot be gotten similar statistics because they are too much influenced by recursion.
- The third set, level statistics, focus on distribution of elements, attributes, text nodes, and mixed contents per each level of XML documents.
- The fourth set, fan-out statistics, describe the overall distribution of XML data. In other words number of children for each node.
- The fifth set, fan-in statistics, is inverse to previous fan-out statistics.
- The sixth set, recursive statistics, deals with types and complexity of recursion.
- The seventh set, mixed-content statistics, analyze the structure and complexity of mixed contents. The number of mixed-content elements is already known from global statistics. So statistics are focused on average and maximum depth of mixed content and the percentage of simple mixed-content elements.
- The eighth set, DNA statistics, focus on pattern called DNA pattern. DNA pattern contains an arbitrary amount if trivial sub-elements and just one complex sub-element, so-called degenerated branch.
- The ninth set, relational statistics, focus on so-called relational pattern. This pattern is analyzed using a set of relational statistics. There are two types of relational patterns – relational and shallow relational – and also the statistics are computed for both separately.

- The last set, schema statistics, completes the analyses with analyzing XML schema specifics constructs and their real exploitation.

These are possible statistics of real-world data.

## 2.2 Analyzer

**Analyzer** [46] was created as an extensible program for data analysis. It was created as project in Charles University of Prague.
It is designed to be able to analyze any kind of data. The first kind of data analyzed with this program were XML documents.
The following attributes of the XML documents were subjects of the analysis:

- The size of the XML document was analyzed e.g. in bytes, by the number of elements or by the number of attributes.
- Maximum depth of the document was analyzed.
- Distribution of various types of content model over different levels was analyzed.
- Recursion of elements was analyzed.
- Maximum and average element fan-outs were analyzed.
- Usage of XML Schema versus DTD was analyzed.
- Distinct element/attribute name usage was analyzed.
- Namespace usage was analyzed.

Next the DTD data was subject of analysis. The following statistics were performed:

- The size of the DTD was analyzed, e.g. number of declarations of elements, attributes, notations, entities etc.
- Number of DTD specific declarations of elements content (`empty`, `any`, etc.) type was analyzed.
- Number of DTD specific declarations of attribute optionality (`#REQUIRED`, `#IMPLIED`, etc.) was analyzed.
- Usage of keys (`ID`, `IDREF`) was analyzed.
- Maximum, minimum and average depth was analyzed.
- Average and maximum fan-outs and fan-ins were analyzed.

Then the XSD data was analyzed. XSD data was analyzed in a similar way as XML data, because XSD is written in XML. Though the analysis was very similar, there were some special measurements added to the XSD analysis:

- Type specification (`simpleType`, `complexType`) was analyzed.
- Restriction and extension of existing types was analyzed.
- Content model of the elements (`sequence`, `choice`, etc.) was analyzed.
- Element groups and attribute groups (`group`, `attributeGroup`) were analyzed.

**Analyzer** also analyzed XQuery data but only on XQuery Use Cases [39] and XQuery Test Suite [33]. This is described in the paper [48].

### 2.2.1 Architecture

**Analyzer** framework consists of two separated levels.
The first level contains so called Shared Components – components shared with all the projects. These components are instantiated only once in a running **Analyzer** application. The most important component is **Launcher**. It is responsible for executing tasks (small units of computations). Graphical User Interface (**GUI**) is robust environment for creating projects, managing projects, running analysis and browsing reports. **Plug-ins** are meant to have the analyzing logic. Due to that for every type of analysis there must be appropriate plug-in.
The second level contains Project Components. These components are used by each project independently. In a manner of implementation it is a part, which provides isolated environment for every project.

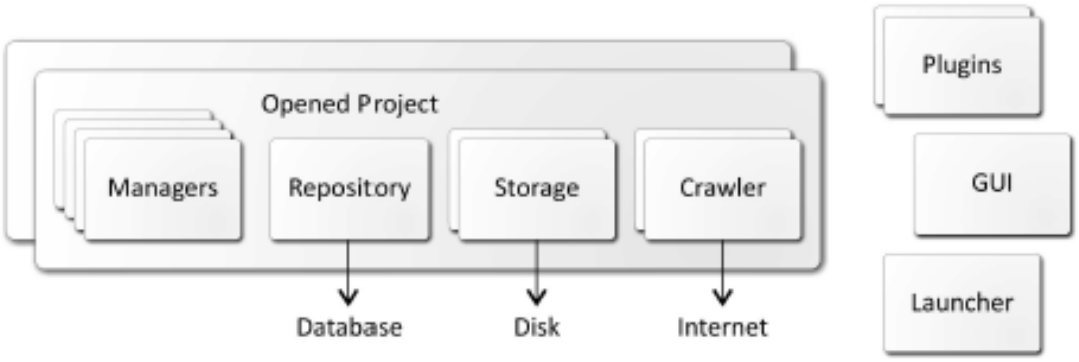



Figure 2.1: Architecture of Analyzer [48]

Figure 2.1 shows the project components:

**Crawler**
Crawler obtains data from the internet.

**Storage**
Storage obtains and saves data from local file system (hard-drive).

**Repository**
Repository is just one local database for each project. All calculated analytical data and the most of the configuration meta data are saved there.

**Managers**
Managers are responsible for creating, modifying and processing of the documents, collections and reports.

### 2.2.2 Implemented Crawlers

The first version of the program did not have implemented crawler and documents were imported only from the local file system [60]. In the next version there were two crawlers implemented into the **Analyzer**:

**Simple Embedded Crawler**
This Crawler was created from the ground using the simple Java mechanisms. It works in the synchronous mode. This means the crawler always waits for the response before sending another request. It is used for seeking out the resources during the analysis only.
The crawler is set to lower the target load server by using the simple serialization of the requests. The number of requests is limited to one request per second.

**Egothor Embedded Crawler** [42]
This Crawler was created as an adapter to the existing crawler from the Egothor Project programmed by Leo Galamboš and collective. The behaviour of the original crawler was modified in the following points:

- The support for notifying downloaded documents using listener was added.
- The reaction for external signalization was added to stop crawling without using the telnet.
- The support for URL injection without using the telnet was added.

Since then Egothor Project released the new version – Egothor2 [37].

An adjusted Apache Nutch [3] crawler is also included in the **Analyzer**. This one was implemented to automate the data collection process and download full structures of XML documents. Additional improvements of the original Apache Nutch are:

**Improved address filtration**
Crawler avoids unwanted protocols (mailto, javascript) and file formats (PDF, MP3 etc.).

**Altered document filtration**
Crawler cuts-out excessively large HTML documents, assuming that they would unlikely contain any XML-related links.

**Added whitelisting**
Whitelisting apparently XML-related documents based on their reported MIME type and a part of contents.

**Added blacklisting**
Blacklisting unwanted documents. Due to the widespread errors in the web content, authors found blacklisting more efficient than whitelisting.

**Altered scoring mechanism**
Crawler makes XML-related data more favourable in the download queue.

**Added XML-based documents parsing**
Parsing XML-based documents and locating external references in them.

### 2.2.3 Analyzer Plug-ins

**Analyzer** is universal analysis tool. Its functionality is achieved with system of plug-ins. For every kind of data there is a special plug-in to analyse them.
There are also plug-ins for other functionality in **Analyzer**. There are 8 predefined types of plug-ins:

**detector**
Detector recognizes type of the processed document.

**racer**
Racer looks for outgoing links in a given document.

**corrector**
Corrector attempts to repair a content of a given document.

**analyzer**
Analyzer produces results of the analysis for a given document.

**collector**
Collector classifies documents into the collections of a given cluster.

**provider**
Provider creates reports by aggregating results of the documents in a collection.

**viewer**
Viewer serves for browsing computed results over a document.

**performer**
Performer serves for browsing computed reports over a collection.

### 2.2.4 XQuery Analysis Plug-in

We are interested in plug-in for XQuery analysis. The plug-in for the **Analyzer** is shortly described below.

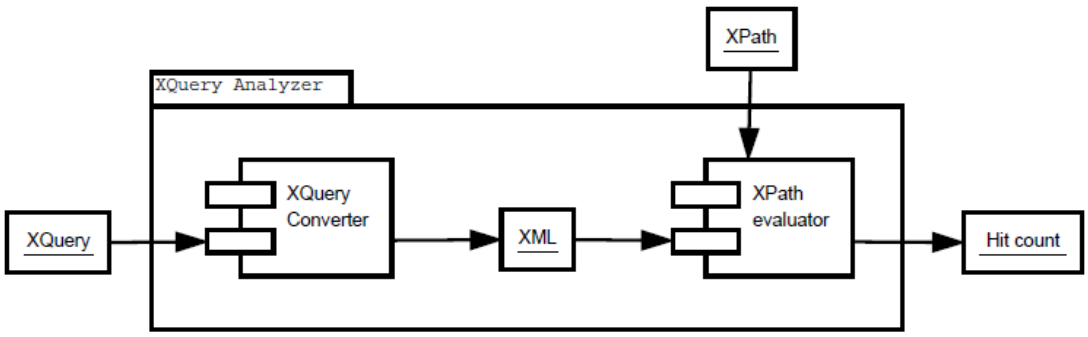



Figure 2.2: Structure of XQAnalyzer component [59]

Figure 2.2 shows the XQAnalyzer component created at thesis [59]. It consists of two parts. The XQuery Converter gets XQuery program and creates its XML representation. This representation is then given to XPath processor together with XPath expression. Final output is the number of XPath results.
This is more sophisticated then just plug-in for counting words. It is thanks to complexity of XQuery language.

## 2.3 XQuery Converter

XQuery Converter (XQConverter) is part of the tool for XQuery analysis. Not only it is included in a plug-in for **Analyzer**, it is also independent command line program. This means it can be used independently from **Analyzer**.
This tool creates XML representation of XQuery Program like in this example:

```
for $for as for in "for" return <for/>
```

From this program where every ‘for‘ has a different meaning depending on the position in the program [49] it creates this XML representation:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Module type="main">
  <QueryBody>
    <FLWOR>
      <TupleStream>
        <ForClause varname="for">
          <Type cardinality="one">
            <AtomicType name="for"/>
          </Type>
          <BindingSequence>
            <Literal quot-mark="double" type="string" value="for"/>
          </BindingSequence>
        </ForClause>
      </TupleStream>
      <ReturnClause>
        <Constructor kind="direct" type="element">
          <Name name="for"/>
        </Constructor>
      </ReturnClause>
    </FLWOR>
  </QueryBody>
</Module>
```

The first ‘for‘ is hidden in element named ‘ForClause‘. The second one is a variable and it is included in attribute named ‘varname‘. The third one is ‘AtomicType‘, the fourth is string ‘Literal‘ and the last one is ‘Name‘ of returning element.

### 2.3.1 Architecture of XQConverter

Figure 2.3 shows that the XQuery Converter component consists of tree parts.

XQConverter includes an **XQuery lexical scanner**. This would be sufficient for calculating usage frequency of the individual words with a given meaning.

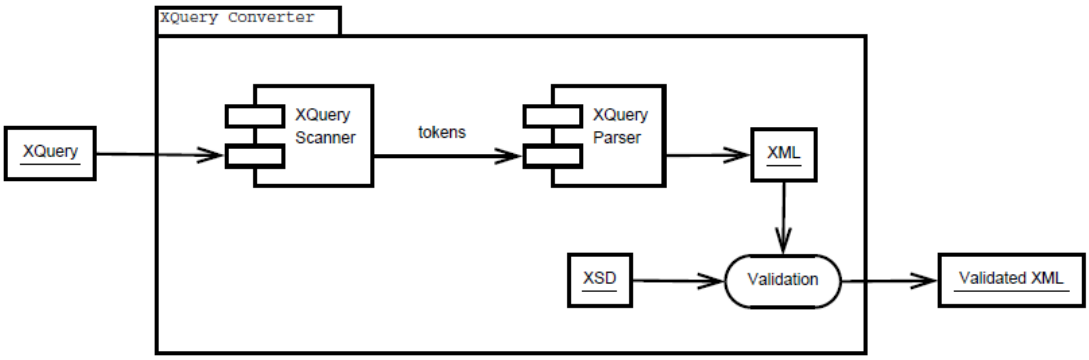



Figure 2.3: Architecture of XQConverter component [59]

The XQuery language constructs and their combinations are complex. To analyze them an **XQuery parser** is needed. By being connected to a **lexical scanner** it is able to recognize the structure of the programs according to the XQuery grammar.
The stream of the data goes into the scanner where it is transformed into the lexical tokens. The tokens are input for the parser which instead of building a syntax tree builds the XML representation of the given XQuery program.
The final XML is validated in the Validator. Validated XML can be queried by XPath processor about the statistical information of the given XQuery program.

## 2.4 Other Used Programs

DoubleKiller [11] is program used for finding and removing duplicate files. It compares files based on four comparison criteria (user can combine them):

- It compares the file names.
- It compares the file sizes.
- It compares the file dates.
- It compares the file CRC32-checksum [6].

Cyclic redundancy check (CRC) is an error-detecting code. In DoubleKiller it is used as a hash function. Blocks of data entering CRC get a short check value attached, based on the remainder of a polynomial division of their contents. This is a sum up for the file.
In this thesis was used combination of file size and CRC32-checksum.

Altova XMLSpy [1] is a complex program. It is an XML editor for modelling, editing, transforming, and debugging XML-related technologies. It offers a graphical schema designer, a code generator, file converters, debuggers, profilers, full database integration, support for XSLT, XPath, XQuery and more.

In this thesis Altova XMLSpy will be used for XPath expressions evaluation.

Other programs were written for purposes of this thesis and are on the enclosed CD.

## 2.5 Summary

Nowadays the data analysis is found to be useful and is commonly performed in order to find out interesting facts in various kinds of data. We know how to analyse XML data. We have a program for XQuery analysis but it has never been used on the real-world XQuery data. The aim of the next chapter(s) will be acquiring the real-world data.

# 3. Crawlers Analysis

In the second chapter we became familiar with most of the programs used for data analysis. Now we look at programs for data acquiring. In this chapter we will describe the term crawler, various types of crawlers, functions of the crawler and we will look on pros and cons of several specific crawlers.

## 3.1 Crawler

We have more options of how to get the right crawler for our purpose.

- It can be written from a scratch. Such decision has both positive and negative aspects. The thing is that this would be the **crawler** with the needed functionality. On the contrary it would take a lot of time designing the model, programming it, testing and debugging it.
- It can be chosen from existing crawlers and edited or extended to fit our needs. This approach is less time-consuming and so more time can be spent focussing on the heuristics about crawling and not on the basics of crawling the Internet.

We choose the second approach. In order to choose the best crawler for out purposes, some sort of classification of various crawlers is appropriate.

### 3.1.1 Crawler Types According to Searching Method

There are two searching methods:

**depth first**

**breadth first**

Depth first searching type of crawlers are generally called 'topical crawlers' or 'topic-directed crawlers'. They are used for searching in the specific domain like **.cz** (national domains), **.net** or they are used for searching for a particular topic like the baseball sites.

Breadth first searching type is more frequent. This type of crawlers often uses parallel processing.

- Parallelism on one computer:

**multi-threaded crawler**

**more processes on one PC**

- Parallelism on more computers:

  **distributed web crawler**

This may shift the bottleneck from the Central Processing Unit (CPU) speed and the Internet connection speed to local DNS server [60]. It is because breadth first search generates a lot of requests and can even create Denial of Service (DoS) on DNS server.

### 3.1.2 Crawler Types According to Policy

The web crawler behaviour can be put in more than one of these policies:

**Selection Policy**
The crawler decides which pages to download.

**Restricting followed links**
The crawler only downloads HTML pages. All the other MIME types are filtered. It can be done through HTTP HEAD request and afterwards MIME type examination or by exploring the file extension in URL address.

**Path-ascending crawling**
The crawler examines the whole URL. From that it derives other URLs which are crawled afterwards. For example, when given a seed URL of 'http://**www.w3.org**/TR/2006/REC-xml-20060816/', it will attempt to crawl '/TR/2006/REC-xml-20060816/', '/TR/2006/', '/TR/' and '/'.

**Focused crawling**
The crawler, also called topical crawler or focused crawler, attempts to download pages that have similar content. For example an academic-focused crawler, which crawls the academic resources.

**Re-visit Policy**
Internet incorporates quantum of data. A lot of data are constantly changing and new data appears. This is the reason why crawlers that are constantly searching the internet and saving new data are necessary. To be able

to effectively search over these data, an indexation is used. That means the set of terms is chosen for saving data. These terms describe contents of the data. For searching in database created this way, the terms are used. Using the terms makes searching in the database faster.
So the crawler decides by itself how often to check for changes in data it has indexed.

**Uniform policy**
All pages are re-visited with the same frequency. This version is not used often.

**Proportional policy**
Pages are re-visited due to the estimated change frequency. The pages that are changed more often are crawled more frequently. The challenge is to set the crawling frequency right considering the estimated change frequency.

**Politeness Policy**
The crawler avoids overloading websites.
The crawler can retrieve more data in shorter time than human searcher. This can cripple some websites. Because of this some etiquette must be followed during the web crawling. Part of this problem is solved by **Robots Exclusion Protocol**.
The **Robots Exclusion Protocol**, also know as the **robots.txt protocol** or **Robots Exclusion Standard**, is about giving information how to crawl the website from the site owner to the crawler. Typically it contains the information which directories not to crawl. They can contain unimportant informations of the website or they can contain spider traps, etc.
This file is always located in the root directory of the web page. The **Robots Exclusion Protocol** can specify every crawler independently.
The second part is politeness delay. It is a delay between requests to the same host. It reduces load on the servers.

**Parallelization Policy**
The crawler runs on more computers simultaneously and it coordinates its work.
This kind of crawler is called a distributed web crawler. We will not use this type because it is expensive.

### 3.1.3 Desired Crawler Functionality

XQuery is a non-XML language so the **crawler** does not need to be an XML-aware crawler. On the other hand it must have a text search engine to detect an XQuery program.

### Multi-threading

A multi-threaded crawler would be preferable, because many pages could be downloaded simultaneously. It could also change the bottlenecks of server latencies and their upload speed to the bottleneck of our download speed.
Also if we use more than one thread we can access various servers simultaneously and we do not have to wait in order not to overload some server. This way the process of getting the data becomes much faster.

### Filters

Some filters of the unwanted data are required. This way the crawler can go through web pages and data that have bigger probability to contain desirable data.

#### Whitelist

Whitelist is a list that contains types of files that should be saved. The first step is that a file is downloaded. Then it is either parsed or its MIME type is analyzed if it is a required file. If it is, then it is saved.
If the rules for filtration are simple (for faster processing) then some required files can be filtered by this list. This solution minimizes the space storage for documents. A great disadvantage is the fact that in the time of writing the rules for filtering not all types of wanted files are known [60].

#### Blacklist

Blacklist is a list that contains which files are filtered (not saved). So the file extensions are being compared if they are filtered. Also the MIME type can be used for this comparison.
This approach removes only unwanted files. For example when we search for the text data we filter all binary data. This way no binary data are downloaded. Some unwanted data would still be downloaded but only as text data. Only the unwanted data are removed.

#### Address filtration

Address filtration means that the addresses from the list are not downloaded. Usually it is the list of URL prefix that is compared whether it is present in the address for download. This can be expanded also for protocols, so that the protocols different from HTTP are being filtered.

#### Large documents

Large documents are a specific case. These documents can take a lot of memory during their parsing and in the worst-case scenario they can fill up memory of parser what can lead to program crash. So it is a good idea to download only part of them or not to download them at all.
If they are set for downloading, the size of the downloaded part should be

specified somewhere in the program.

In case filtering does not bring enough documents, some of the filters can be limited.
The **Robots Exclusion Protocol** 3.1.2 is also kind of filter. It says which directories not to crawl because they can contain unnecessary informations, spider traps, etc.

#### Politeness Delay

As mentioned before it is delay between requests to the same host. Too many downloads in a short time can lead to banning the **crawler** or even crashing the website. On the other hand too few downloads means longer crawling time.
The challenge is to set the right value, so no host servers are overloaded and crawling do no take months.

### 3.1.4 Starting Points for Crawling

We cannot afford the Internet connection and the computation performance for the deep crawling of the internet (to crawl every IP in the world) like the web search engines usually do (Google, Bing, Yahoo, etc). It would take an enormous proportion of the computation resource that we do not possess. So it is needed to clarify some points where to start crawling.
We also need seed as other small crawlers. Seeds are web pages where crawler starts it work.
The question is what keywords to use and which web search engine to use.
Probably the best choice is the web search engine which is used on the Internet most often. There is a high possibility that the results are relevant and voluminous.
The ‘web search query‘ could be:

**filetype:xq AND doc** [60]
: This query was proposed in the previous thesis where all types of XML data were collected. It should find all files with **.xq** file extension.

**filetype:xquery AND doc**
: This one is a derivation of the previous query but this time the **.xquery** file extension is asked for instead of **.xq**.

**‘xquery version‘**
: This query is based on the fact that XQuery programs should contain ‘xquery version 1.0‘ in the first line.

**xquery version**
: This one is a derivation of the previous but probably it is not going to be used. It can be used only when just a few programs were found on the Internet.

**FLWOR construct or other**
: FLWOR in XQuery language means For, Let, Where, Order by and Return. These words are used as queries if just a few programs were found, too.

This approach should give enough seeds for the **crawler** so it can find enough programs as a representative sample.

### 3.1.5 Recognizing XQuery Programs

The next step is to identify whether the documents are XQuery programs or just some text.
XQuery keywords are quite helpful during this process. We can search for them in a text and we can mark the files as XQuery programs and non-relevant files (those that do not include XQuery programs) can be thrown away. The determination whether it is XQuery program can be left for more detailed Analysis.

**file extensions**
: The convention of the file extension in W3C recommendation is **.xquery** or **.xq** but there is always a possibility that the XQuery program is in any kind of the text file (file extension different from **.xq** or **.xquery**), so the **crawler** must check whether XQuery program is in some kind of a text file [28].

**HTML page pre section**
: Some XQuery programs can be located directly in the web page code. There is a tag '`<pre>`' in the HTML to separate itself from the other languages codes, so the **crawler** should also take part of HTML page as a separated programs.

**Text Files**
: The same as `<pre>` sections in HTML page. It can contain XQuery program. So check if to save this kind of files is desired.

**Compressed Archives**
: Archives are big files that can contain anything. they are also used as backups or storage of unused data/programs. That is why the effort to download and check this data, is not useful.

This could also be data classification as a download type of XQuery program:

- XQuery programs in XQuery files.
- XQuery programs in text files.
- XQuery programs as part of web pages restricted with tag '`<pre>`'.

**XPath Versus XQuery**

XPath 2.0 is a subset of XQuery 1.0, so we also want to search for XPath 2.0. Because XPath are usually used as short queries they are not often put into files. There is a method for distinguishing one language from the another.

The XPath expression to distinguish XPath 2.0 and XQuery 1.0 is [59]:

```
/Module/@version | //ModuleDecl | //Prolog | //LetClause |
//WhereClause | //OrderByClause | //ForClause/Type |
//ForClause/@posname | //QuantifiedExpr/InClauses/InClause/Type |
//Typeswitch | //Extension | //ValidateExpr | //OrderedExpr |
//UnorderedExpr | //Constructor
```

If the resulting hit count is zero, the analyzed file is written in XPath 2.0 and also in XQuery 1.0. Otherwise it is only XQuery 1.0.

### 3.1.6 XML Documents Referenced in XQuery Program

The XQuery program can contain links on XML documents. This links are in the 'doc()' function. The parameter is absolute or relative path to the data. So the first problem is to get absolute path. Also link can be broken or damaged (some characters can be changed).
Storing them is the second problem. After download they must be connected to their XQuery programs. This is hard but it is possible. The question of having duplicities (if 2 different XQuery programs uses one XML document) must be solved.
And the last problem is how to analyze them. They must be connected to XQuery program and tool that can analyze XML documents and XQuery program is required. So writing a new tool (which connects these analyzers) is necessary.

### 3.1.7 Crawler Integration into Analyzer

There is an important question if the **crawler** is an external program and its results are being added to **Analyzer** through the import documents or the crawler is written as a component of **Analyzer**.
It would be an advantage to choose some existing crawler which is written as a

library so it could be easily inserted into **Analyzer**.
We also count with the possibility of having the crawler as a separate program.

## 3.2 Existing Crawlers

There are many crawler to choose from. We will closer focus on some of them and choose the most suitable candidate for our purposes. At the end of the chapter there is a table with all mentioned crawlers and their main functionality and informations about them.
These are mostly Java crawler:

**Heritrix** [47]

The main purpose of this crawler is to preserve the digital artefacts of our culture for the benefit of the future researchers and generations. It is an open source, extensible, web scale and archival project. It searches the Web with the **breadth first** method and uses more threads to perform the task. The authors claim that it is designed to respect the **Robots Exclusion Protocol** and META robots tags, and collect material at a measured, adaptive pace unlikely to disrupt normal website activity. Examples of how to use **Heritrix** can be found in user guide on its web page. It is written in Java and the latest release is version 3.2.0 from 2014.
This crawler already has a lot of the needed functionality. However some modifications would be necessary in this crawler so that it could be used for obtaining XQuery programs. For example indexing and archival part of **Heritrix** would not be used, but recognizing possible XQuery programs part is necessary.

**WebSPHINX** [32]

**Web**site-**S**pecific **P**rocessors for **H**TML **IN**formation e**X**traction is a Java class library and an interactive development environment for web crawlers that browse and process web pages automatically. It consists of two parts: the Crawler Workbench and The **WebSPHINX** class library.
The Crawler Workbench is a graphical user interface that lets the user configure and control the customizable web crawler.
The **WebSPHINX** class library:

- Library contains multi-threaded web page retrieval in a simple application framework.
- Library supports the **Robots Exclusion Protocol**.
- Library uses pattern matching, including regular expressions, Unix shell wildcards, and HTML tag expressions.

It has its negatives. There is no mention about politeness delays. Also the latest release is version 0.5 from 2002. It is in a process of development and it still may have some bugs.

**JSpider** [16]

**JSpider** is a highly configurable and customizable web spider engine. It is a multi-thread crawler. It is designed to crawl a small number of the sites. It respects the **Robots Exclusion Protocol**. In the documentation there are some examples.

There is also a disadvantage. There can be only one seed. To get more functionality the plug-in creation is required. The last release is 0.5.0-dev from 2003. It is in a process of development and it still may have some bugs.

**WebLech** [31]

The authors created a full featured website download and mirror tool in Java which supports many features required to download websites and emulate standard web browser behaviour as much as possible. It is a multi-threaded crawler. It can be both depth first search or breadth first search type of the crawler. It is an open source and also has some filtering of URLs.

There is no mention about the **Robots Exclusion Protocol**. Also nothing is mentioned about politeness delays. It is in the pre-alpha phase since 2004 and the version number is 0.0.4. It is in a process of development and it still may have some bugs.

**Arachnid** [54]

**Arachnid** is a Java-based web spider framework. It includes a simple HTML parser object that parses an input stream which contains HTML content. Simple web spiders can be created by sub-classing of the Arachnid and by adding up a few lines of the code called after each page of a website is parsed. It is a simple crawler with only one thread, no filters, no **Robots Exclusion Protocol** and no politeness delay. There is an example how to use it on website. The latest version is 0.4 from 2013.

Authors do not recommend using it on other people's websites. It may put a large load on a server and even crash it.

**JoBo** [19]

**JoBo** is a simple program to download complete websites to a local computer. Basically it is a web spider. The main advantage compared to the other downloaded tools is that it can automatically fill out forms and also use cookies for session handling. It also features very flexible rules to limit download by URL, size and/or MIME type.

It is an one-thread crawler. It has the **Robots Exclusion Protocol** and politeness delay implemented. The politeness delay is set to 1 second by default. There are examples of how to use it in the source code. The latest version is 1.4 from 2006.

**Web-Harvest** [30]

**Web-Harvest** is an open source web data extraction tool written in Java. It offers another way of how to collect desired web pages and extract useful data from them. It uses well established techniques and technologies for

text/XML manipulation such as XQuery, Regular Expressions, etc.
It is multi-threaded. There is no **Robots Exclusion Protocol**. It also has no politeness delay. There are examples on the web page and also in the source code. The latest release is 2.0 Beta from 2010. It is for testing purposes.

**Crawler4j** [43]

**Crawler4j** is a Java library which provides a simple interface for crawling the Web. It has nice examples on the web page. In the source code it has the file filters, **Robots Exclusion Protocol** and politeness delay. It is multi-threaded and the latest version is 3.5 from 2013.

**Ex-Crawler** [44]

**Ex-Crawler** project is divided into three sub-projects. The first of them is the Ex-Crawler server daemon, a highly configurable and flexible web crawler written in Java. It uses plug-ins which it can detect automatically. The second part is a GUI and the third part is a web search engine. The web search engine part is written in PHP.
**Ex-crawler** does not have **Robots Exclusion Protocol** and also no politeness delay. It is multi-threaded and has basic filtering file pattern. Examples are on the web page. The latest release is 0.1.6 Alpha from 2010. It is in a process of development and it still may have some bugs.

**Bixo** [5]

**Bixo** is an open source web mining toolkit. It runs as a series of Cascading pipes on the top of Hadoop. It is a topical crawler. It uses the **Robots Exclusion Protocol** and politeness delay. It is a multi-threaded crawler. There are examples in the source code. The latest version is 0.9.1 from 2014.

**Abot C# Web Crawler** [35]

**Abot** is an open source C# web crawler built for speed and flexibility. It is a multi-thread crawler. Examples are included in the source code. The latest release is 1.2.3 from 2014.

**Scrapy** [27]

**Scrapy** is a fast high-level screen scraping and web crawling framework, used to crawl websites and extract structured data from their pages.
It does not have more threads but it is possible to run more spiders at once for faster download. It is written in Phyton. The latest release is 0.22 from 2014.

**OpenWebSpider** [23]

**Open Web Spider** is an open source C# web crawler. There is poor information on what it actually does and how it works, but it has some examples

on the website of the creator. The latest release is 0.1.4 from 2013.

There are also crawlers discussed in the paper [48]:

**Larbin** [20]
: According to its author, **Larbin** is just a crawler. It does not contain any indexer. It is available for free. It was initially developed for the XYLEME project [34].
It is written in C++. The latest version is 2.6.3 from 2003.

**Egothor**
: **Egothor** was partially developed by students of the Faculty of Mathematics and Physics, Charles University in Prague. **Egothor** version 1 is part of the **Analyzer**. The latest version is 3.1.8 from 2014.

**Apache Nutch**
: **Nutch** is a highly extensible and scalable open source web crawler software project from the Apache family.
It was already used in the **Analyzer** extension as a crawler to download XML data. The latest version is 1.8 from 2014.

**Google Web API** [13]
: **Google Web API** was officially deprecated by Google on November 1, 2010.
It also has limitation on how many requests per day can be done. So it is not suitable as a crawler. In **Analyzer** it was already used as a benchmark and test suite.

Table 3.1 contains all discussed crawlers and their key features. The columns **Name of crawler** and **Version** are self-explaining. The column **Language** means programming language in which is crawler programmed. The **Multi-thread** column shows if the crawler is an one or more threaded application. It does not show whether the crawler uses more processes or runs on more than one computer. The **Filter** column shows whether the crawler has any kind of filters or **Robots Exclusion Protocol**. It can contain, for example, part which checks the **Robots Exclusion Protocol** or blacklist, etc. If none of the filters are presents it gives 'No' as an answer. The **Examples** column shows whether any example of usage is present in the web page or in the source codes. And finally column **Licence** shows which of the open source licences is used.

## 3.3 Chosen Crawler

As we have mentioned some of the crawlers do not have a good politeness policy. They do not respect the **Robots Exclusion Protocol** or politeness delay. These

| Name of crawler | Version | Language | Multi-thread | Filters | Examples | Licence |
|---|---|---|---|---|---|---|
| Heritrix | 3.2.0 (2014) | Java | Yes | Yes | Yes | Apache 2.0 |
| WebSPHINX | 0.5 (2002) | Java | Yes | Yes | Yes | Apache |
| JSpider | 0.5.0 (2003) | Java | Yes | Yes | Yes | LGPL |
| WebLech | 0.0.4 (2004) | Java | Yes | No | No | MIT |
| Arachnid | 0.4 (2013) | Java | No | No | Yes | GPL |
| JoBo | 1.4 (2006) | Java | No | Yes | Yes | LGPL |
| Web-Harvest | 2.0 (2010) | Java | Yes | Yes | Yes | BSD |
| Crawler4j | 3.5 (2013) | Java | Yes | Yes | Yes | Apache 2.0 |
| Ex-Crawler | 0.1.6 (2010) | Java / PHP | Yes | Yes | Yes | GPL |
| Bixo | 0.9.1 (2014) | Java | Yes | Yes | Yes | MIT |
| Abot C# Web Crawler | 1.2.3 (2014) | C# | Yes | Yes | Yes | Apache 2.0 |
| Scrapy | 0.22 (2014) | Python | No | Yes | Yes | BSD |
| Open Web Spider | 0.1.4 (2013) | C# | Yes | No | Yes | GPLv2 |
| Larbin | 2.6.3 (2003) | C++ | Yes | Yes | No | GPL |
| Egothor2 | 3.1.5 (2014) | Java | Yes | No | Yes | Egothor 2.0 |
| Apache Nutch | 1.8 (2014) | Java | Yes | Yes | Yes | Apache 2.0 |
| Google Web API | – (2010) | – | – | – | – | – |

Table 3.1: Comparison of suitable crawlers

are **WebSPHINX** (missing politeness delay), **WebLech** (missing **Robots Exclusion Protocol** and politeness delay), **Arachnid** (missing **Robots Exclusion Protocol** and politeness delay), **Web-Harvest** (missing **Robots Exclusion Protocol** and politeness delay) and **Ex-Crawler** (missing **Robots Exclusion Protocol** and politeness delay). This can lead to putting a large load on a single server and even crashing it or to access pages that can be possible **spider traps**. The **JSpider** can only have one seed and we need more seeds, so that we can get more results more quickly compared to using only one seed.
As we can tell **JoBo** is only one-threaded crawler. It is not bad but if we want

to crawl a lot of pages in a short time, then multi-threaded crawler is preferred. Some crawlers are poorly documented like **Larbin** and **Open Web Spider**. **Scrapy** is written in Phyton language. To use it would require to learn Phyton language.
This leaves 6 crawlers to choose from: **Heritrix**, **Crawler4j**, **Bixo**, **Abot C# Web Crawler**, **Egothor** and **Apache Nutch**. The last two of them were already used with **Analyzer** project in the earlier state. **Bixo** is a topical crawler and more of a mining tool. **Heritrix** has the archival purpose. **Abot C# Web Crawler** means no possibility to include this part of software into **Analyzer**. So based on previously stated information about various crawlers, the best choice is **Crawler4j**.

# 4. Data Gathering

In the previous chapter we chose the **crawler**. Now we can proceed to the next step – acquiring the data. Because we could no build **Analyzer** poject in NetBeans 8.0 IDE [22], we are going to use the **crawler** as a stand-alone program.
In this chapter we are going to acquire seeds as an input for the **crawler**, edit crawler's functionality for our purposes and gather the data.

## 4.1 Seeds Requiring

Google provides several ways of getting the data it collects with its crawler (Googlebot [14]). The easiest way to access the data Googlebot has crawled and indexed is via Google web page 'http://www.google.com' (you can also use particular Country top-level domains like .sk, .cz etc.).
For more skilled users there is another way using Custom Search Google API [9]. This is a programmer approach and it requires 'developer api key' and 'custom search engine id' to get started.
The last way is using Google knowledge graph. This is the only known way to get all the results of Google database of a specific query. Unfortunately, the graph for XQuery was not created yet.

We choose the first way – collecting URLs for the **crawler** using Google site search. We found program, Outwit [24], that offers relatively easy way to collect URLs from Google search, but it is not for free and it takes some time to get familiar with it.

We decided to write our own program (scraper). This program gets Google web page with the results and it extracts URLs of the results from this Google web page. In other words, Google web page with the results is input of the scraper and the output is a text file with each URL result from Google search on different line.
In order to minimize number of scrapping operations we need as much search results per page as possible. The maximum number of search results per page in Google search is limited to 100 results per page.

## 4.2 Seed File

We took suggested queries from list 3.1.4 and saved results as web pages.
Table 4.1 shows how many results (URLs) we can get from specific query. It is limited to maximum 100 * 10 because of google default behaviour.
Column **omitted results** explains whether we did search with omitted results in Google search. Column **Google results** shows the number Google writes on the first page of results as how many results it has in its database. And the last column **gained results** is the sum of all results we got from Google pages for

certain query.
These data were collected 27.06.2014.

| **Google query** | **omitted results** | **Google results** | **gained results** |
|---|---|---|---|
| filetype xq AND doc | NO | 80 | 80 |
| filetype xq AND doc | YES | 2,910 | 728 |
| filetype:xquery AND doc | NO | 30 | 30 |
| filetype:xquery AND doc | YES | 1,700 | 268 |
| ‘xquery version‘ | NO | 22,000 | 154 |
| ‘xquery version‘ | YES | 22,000 | 811 |
| xquery version | NO | 1,140,000 | 200 |
| xquery version | YES | 1,140,000 | 772 |
| filetype:xq | NO | 68 | 68 |
| filetype:xq | YES | 441,000 | 981 |
| filetype:xquery | NO | 61 | 61 |
| filetype:xquery | YES | 5,100 | 577 |
| for let where order by return | NO | 537,000,000 | 472 |
| **Together** | – | – | 5,202 |

Table 4.1: Google search results for queries

### 4.2.1 Scrapper

We created Java program ‘scrapper‘. This program takes Google web page as a text file and takes only URLs which are results of the search and puts them into separated lines of the text file. The difference between result URL and other URLs on page is that the result URL is located:

```
...<h3 class="r"><a href="result_URL">title_of_result<...
```

### 4.2.2 Script

Now that we got the results we need to put them into one file. We used Windows PowerShell script to put these files into one file.

```
Get-Content *.txt | Sort-Object -Unique | Set-Content input.txt
```

The command ‘`Get-Content`‘ gets the content of the item from the specified location – in our case all the ‘`txt`‘ files – and moves it through pipe to next command.
The command ‘`Sort-Object`‘ sorts objects by property values. In our case it sorts it alphabetically and parameter ‘`-Unique`‘ eliminates duplicates.

The last command ‘`Set-Content`‘ writes or replaces the content in an item with new content. In our case it saves the output into the file ‘`input.txt`‘.

Thanks to elimination of duplicates we get 4,198 unique URLs out of a total of 5,202 results.

## 4.3 Crawler's Adaptation And Settings

As soon as we have the seed file we can proceed to crawler's adaptation.
A lot of functions and features are already implemented into the **Crawler4j**. Configuration file parameters with default values and explanation of the settings are described in the following text::

**crawlStorageFolder**
: Relative or absolute path to folder where the **crawler** saves its data.

**resumableCrawling**
: If this is set to **true**, the **crawler** is able to resume previously stopped/crashed crawling. The default value is **false**.

**maxDepthOfCrawling**
: Sets maximum depth of crawling. The default value is **-1** what means unlimited depth.

**maxPagesToFetch**
: Sets maximum number of pages to fetch. The default value is **-1** what means unlimited number of pages.

**userAgentString**
: The string that is used for representing the **crawler** to web servers. The default value is ‘**crawler4j (http://code.google.com/p/crawler4j/)**‘.

**politenessDelay**
: Politeness delay in milliseconds. The default value is **200**.

**includeHttpsPages**
: If **true**, HTTPS pages are also crawled. The default is **false**.

**includeBinaryContentInCrawling**
: If **true**, binary data as ‘video‘, ‘image‘, etc. will also be fetched. The default value is **false**.

**maxConnectionsPerHost**
Sets maximum number of connections to one host. The default value is **100**.

**maxTotalConnections**
Sets maximum number of connections in total. The default value is **100**.

**socketTimeout**
Sets the time interval after which the socket is closed, interval is in milliseconds. The default value is **20,000**.

**connectionTimeout**
Sets the time interval after which the connection is closed, interval is in milliseconds. The default value is **30,000**.

**maxOutgoingLinksToFollow**
Sets maximum number of outgoing links from one page which are processed. The default value is **5,000**.

**maxDownloadSize**
Sets maximum allowed size of a page. Pages larger than this size will not be fetched. The default value is **1,048,576** what represents $2^{20}$**B**.

**followRedirects**
If **true**, redirections are followed. The default value is **true**.

**proxyHost**
If **crawler** should run behind a proxy, this parameter can be used for specifying the proxy host. The default value is **null**.

**proxyPort**
If **crawler** should run behind a proxy, this parameter can be used for specifying the proxy port. The default value is **80**.

**proxyUsername**
If **crawler** should run behind a proxy and username/password is needed for authentication in proxy, this parameter can be used for specifying the username. The default value is **null**.

**proxyPassword**
If **crawler** should run behind a proxy and username/password is needed for authentication in proxy, this parameter can be used for specifying the

password. The default value is **null**.

The most of the parameters kept their default values but some of them needed to be adjusted. The list of changed parameters is following:

- Variable **crawlStorageFolder** was set to '**XqueryTestCrawler**'. This means, that a folder named '**XqueryTestCrawler**' was created in directory where the **crawler** is running.
- Variable **resumableCrawling** was set to **true**. This option was tested in small gathering runs and used in big data gathering.
- Variable **maxPagesToFetch** was set to values from 10,000 to 10,000,000.
- Variable **userAgentString** was set to '**crawler4j for XQuery**' to create unique string identifier of **crawler**.
- Since we did not want to overload target servers with our numerous requests, we set the variable **politenessDelay** to **1,000** milliseconds (= **1** second).
- Value **maxDownloadSize** included our Large document filter and we set this value to **524,288** (stands for **0.5 MB**). There were two main reasons for setting this value. First, process of parsing big files means using a lot of memory so we did not want to risk running out of memory. And second, we asume that majority of the XQuery programs has small file size.

### 4.3.1 Java Settings

We use Java programs that run on Java Virtual Machine (JVM) [63]. JVM offers networking properties settings [18]. We tried various settings for sites crawling.

- We found out that the setting **sun.net.spi.nameservice.nameservers** did not affect the use of DNS server. DNS server set directly in the Operation System (OS) was used for translation, instead of one setted in program.
- We decided to use value **false** for the setting **http.keepAlive**. Value **false** means that opened HTTP connections are closed after one request and they cannot be used for the next request to the same server.
  We used this setting as a fail-save mechanism if setting **maxTotalConnections** would not be taken into account.
  Also if crawler sends many requests to different servers and run out of file descriptors, it must either wait for a long connection timeout or go through its databases whether there is any URL to crawl in opened connections. This would be too slow with a big database. On the other hand waiting for a timeout would be ineffective. Both ways, it would slow down the data collection.
  Contrary to this option if the connection is closed right after processing the request, creating a new connection to the same server takes less time than chosen politeness delay (no need to create a new connection to the same server is the only case when keep alive would be useful).

### 4.3.2 Used Filters

We also used all the filters described in Subsubsection 3.1.3.

**Large documents filtering**
It is directly in configuration settings.

**Address and protocol filtering**
It is implemented as two patterns.
Protocol filtering:

```
"(ftp|javascript|mailto|https).*"
```

This is a regular expression used for URL control – so that the beginning of the URL does not contain unwanted protocols which used **crawler** is not able to handle.
Address filtering:

```
"https?\\://(facebook.com"
+ "|(\\w*\\.?)google"
+ "|www.like-news.us"
+ "|ozmalafatra.sk).*"
```

This regular expression filters several pages, mainly those we know does not contain XQuery programs (`ozmalafatra.sk` – our own testing page). Also the pages where anti-virus program popped out a warning (`www.like-news.us`), or pages too extensive which are not likely to contain the wanted data (web searching engine `google`, social network `facebook.com`). This is an example of the filter of pages that could be excluded from crawling.

**Whitelist filter.**
Marks files supposedly being XQuery programs and saves them into special directory. It looks like this:

```
".*(\\.xqu?e?r?y?)$"
```

This means that everything with **.xq** or **.xquery** extension (other possibilities makes no harm for example **.xqy**, etc.) is saved into the special directory..

**Blacklist filter**

Filtered file extensions of files which the **crawler** will not even try to download. It is the most extensive filter and it looks like this:

```
".*(\\." // anything at beginning and then dot
  //(seperator before file extension)
+ "(css|js|ico|jar" //web files
+ "|bmp|gif|jpe?g|png|tiff?" //image files
+ "|mid|mp2|mp3|mp4|wav|wma|ogg" //audio files
+ "|avi|mov|mpe?g|ram|m4v|mkv|wmv|mod|flv|3gp" //video files
+ "|docx?|pps|pptx?|xlsx?" //office files
+ "|pdf|pdb|rtf|mobi|epub" //not wanted document files
+ "|zip|rar|z7|tar|gz" //compressed files
+ "|dll|so|bat|sh|bash" //dynamic libraries and script files
+ "|cp?p?|h|java|cs" //source code files
+ "|rm|smil|swf|exe)" //other files
+ ")$"); // (\\?.*)? - question mark and parameters
  //at the end (optional)
```

At the end of each line is a description of file extensions described in the given line.

### 4.3.3 Threads

At the beginning the **crawler** was run with 7 threads. We decided for 7 threads because examples of using the **crawler** had it set this way and we were rather focusing on testing the other parts of the **crawler**. When we got to the question, how many threads should **crawler** be run with, we were also tuning the politeness delay.
After some crawler's code editing we tried 10 threads and the result was that there were 10 accesses to one server during 1 second interval (value of politeness delay). So as a fail-safe we set it to 10 threads.

## 4.4 Crawler's Code Editing

At the beginning, the basic functionality of the **crawler** had to be specified. Examples of how to do it was the subject of tutorials. From the most of the tutorials we took some information that we used to edit our **crawler**:

- Basic crawler's example has only the most essential settings for the first run.

- A crawler that is able to download pictures and save them to hard-drive. We used the part that does not download binary data. Despite the crawler having the option not to download the binary data, we used it as a fail-safe mechanism.

- A crawler that created statistics of already searched websites. We used this and implemented further statistics. The statistics pop out after finishing current crawler's sessions. If crawler crashes, we will not be able to get to these statistics. Numbers of files downloaded, pages searched, invalid links and the runtime of the crawler are parts of the statistics.

- An example of how to start differently-set crawlers at the same tame. We needed identically-set crawlers, so we did not use this option.

- An example of how to stop crawling. We used this in the different ways. We waited in console for the user entry. It was ineffective (crawling could not be run in the background as daemon). So we changed it so that crawler controls whether the file with specific name exists all the time. When the file exists the crawler turns itself off. In our case the file name was 'quitfile'. Using this option we could test whether the crawler is able to work in runs, meaning it can run for given time period and then it stops itself. Next time it continues where it ended.

- A crawler that took care of status codes (for example 404 Page not found). We used this for the statistics – the number of invalid links was tested specifically on 404 status.

### 4.4.1 Adding Seeds

There was no code for adding seeds from file so we wrote one.
The file has to have the following structure – empty lines are ignored, lines starting with '#' are also ignored (comments) and other lines must have just one URL per line.
Before entering the URL to database the URL is compared with the filters whether it is suitable (so that useless URL would not be added).

### 4.4.2 Data Saving

From 3.1.5 we took 3 places from where we will download XQuery programs. We need 3 directories where to put downloaded programs.
The directory 'XqueryTestCrawler/Download/XQuery' is reserved for the files with XQuery file extensions.
The directory 'XqueryTestCrawler/Download/MaybeXQuery' is reserved for the text files with other file extensions.
The directory 'XqueryTestCrawler/Download/PreHtml' is reserved for the files created by extracting `<pre>` section for web page.

Naming the saved files is also a big challenge. Wrong file name may cause overwriting of the existing file. We used the file's URL as a unique character

string.
We needed to change some characters like ‘:‘, ‘?‘, ‘/‘, etc. We transformed all of them into ‘.‘. This way every file was saved.
A little mistake occurred there – too long file name were problem for Windows OS. So we started checking if the new saving name is not too long. If it was too long we created new file name.
This file name consists of default string (‘wrong filename-‘), number of file with long file name, string (‘-‘), current date and time (day.hour.minute.second.millisecond), string (‘-‘) and file extension of original file so no duplicities would occur.

### 4.4.3 Politeness Delay

Politeness delay was found out to work in a different way that it was previously thought. This was observed during the first few runs of the **crawler**.
Instead of putting to sleep threads that wanted to access host that was access in shorter time then politeness delay, it put to sleep every thread every time.
The sleep time was counted as the last sleep time (of any thread) + politeness delay.
We alter this logic to put every thread independently to sleep for politeness delay. As expected it resulted in the situation where all 10 threads were trying to access one host. Some of the hosts did not handler this. So we modify the functionality. We added dictionary as hash map, where we saved access times for added IP addresses (IPv4 and also IPv6). This was realised through DNS translation of host's domain name. We counted on the fact, that one hose can have more different IP addresses. Also one IP can have different domain names.
Here is sleeping part of code:

```
synchronized (mutex) {
    //get last time the crawler requested something
    //from server(s) of that domain
    lastFetchTime = dictionary.getValue(webUrl.getDomain());

    //get current time
    long now = (new Date()).getTime();

    //calculate delay from last access to that server(s)
    long existingDelay = (now - lastFetchTime);

    //if politeness delay is required sleep
    if (existingDelay < config.getPolitenessDelay()) {
        //calculate for how long the sleep is needed
        long newDelay = config.getPolitenessDelay();
        newDelay -= existingDelay;

        //put new access time to that server(s)
        //before sleep
        long newAccessTime = lastFetchTime;
```

```
            newAccessTime += config.getPolitenessDelay();
            dictionary.add(webUrl.getDomain(), newAccessTime);

            //finally sleep
            Thread.sleep(newDelay);
        }

        //before request put the access time to dictionary
        //(if newer time is already there it will not rewritten)
        dictionary.add(webUrl.getDomain(), (new Date()).getTime());
    }
```

When asking for the latest access time, dictionary makes DNS request to find all the IP addresses and dictionary chooses the greatest access time associated with this IP addresses. If no IP address is present for given domain name, zero is returned.
When adding access time, dictionary makes DNS request to find all the IP addresses and dictionary saves this access time for them. If record of IP address is not present than it is added. If IP address's access time is greater, it is not rewritten.

To speed the process, we clear the dictionary once it has 100,000 IP addresses present. After the clearing of the dictionary, the first X (X is based on number of threads) request for the latest access time is chosen current time + X. This ensures that politeness delay is constantly maintained.

This solution uses 3 identical DNS request to access one web page. Because we use own cache DNS server this is not an issue for us. This way the cache DNS server makes only one DNS request.
Also nearby DNS server helped us speed up process. The ping between the PC with the **crawler** program and the DNS server was lower than 1 millisecond.

### 4.4.4 Final Architecture

In figure 4.1 we can see the architecture with all the extra added functionality. The beginning is on left top corner where a file containing the seeds (URLs) is read. URLs matching the filter are discarded. The rest of them are sent to the database.
The Fetcher component asks for the URLs from the database. There is one Fetcher component for each thread. This means that each Fetcher asks independently the dictionary, when it is allowed to access the picked URL. After sleep time it sends HTTP request. If the file to fetch is too big, the large document filter denies this file.
Obtained file is passed to Parser component. Parser component distinguishes whether the file is binary, web or text. The binary file is thrown away. If it is web file all the URLs that are passing through the filter (not matching it) are extracted out of it. The rest of them are thrown away. Before uploading it into

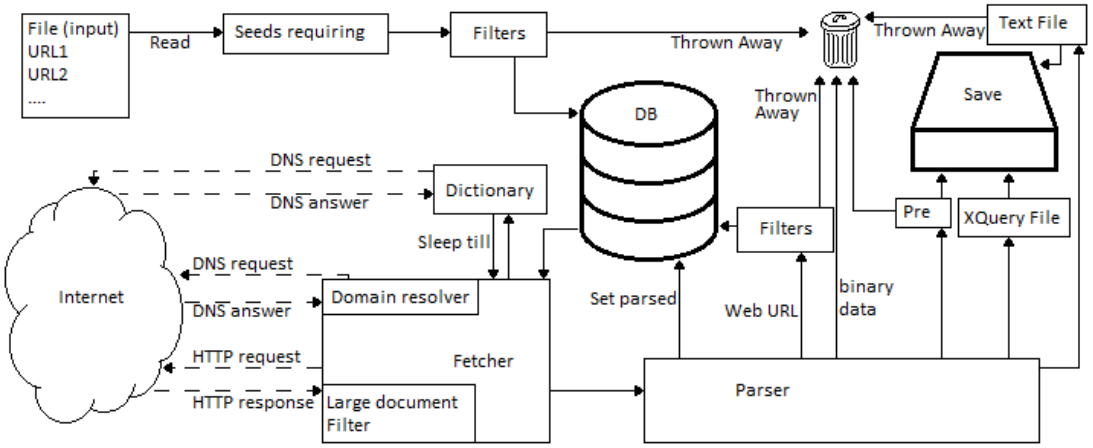



Figure 4.1: Crawler architecture

the database, it is checked whether it does not exceed the set limit of maximum websites.
Also the web page is looked through for `<pre>` sections. Those are extracted from file and controlled whether they contain some XQuery keywords. If they do contain they are saved (on hard-disk), otherwise thrown away. Text files are examined on their extensions. If it is XQuery file extension they are automatically saved. Otherwise they are examined if they contain XQuery keyword. If they do contain they are saved, otherwise they are thrown away.

## 4.5 Crawler's Runs And Results

We used this edited **crawler** to make the crawls. We did more independent crawls to get better results. Here is chosen **maxPagesToFetch** parameter for every crawl. This crawls will be referred to as runs and every run will consist of sessions because we cannot run the **crawler** continually. There are names for the runs in the round brackets:

- 10,000 pages to crawl (10k).
- 100,000 pages to crawl (100k).
- 1,000,000 pages to crawl (1M).
- 10,000,000 pages to crawl (10M).

To select right **maxPagesToFetch** we got inspiration from thesis [60].
Their crawler processed approximately 6,000,000 pages. From this were 716,000 documents downloaded, what represents approximately 12%. Only 48,262 (less that 1%) of the downloaded documents were XML-related.
We assumed, that no more than 1% of the pages would be relevant (will contain XQuery program).

We did some testing sessions during the crawling. We tested repaired functionality of some parts of the **crawler**. We do not have statistics from these testing sessions, but we kept the downloaded files.
Table 4.2 shows statistics – **starting time**, **ending time**, how long did the crawling session took, what **PC** it was run on and the name of data captured.
We used two different machines.
The first machine's (twister-ntb) parameters were: CPU: Intel dual core 2,2GHz, Random Access Memory (RAM): 2GB, OS: Windows 7.
The second machine's (mail.kolej) parameters were: CPU: AMD64 RAM: 2GB OS: FreeBSD.

| ID | starting time | ending time | running time | PC | name of run |
|---|---|---|---|---|---|
| 1 | 28.06.2014 - 22:00 | 28.06.2014 - 22:02 | 00:02:05 | twisterntb | 100k |
| 2 | 28.06.2014 - 22:04 | 29.06.2014 - 00:53 | 02:49:39 | twisterntb | 100k |
| 3 | 29.06.2014 - 03:40 | 29.06.2014 - 09:20 | 05:40:02 | mail.kolej | 10k |
| 4 | 29.06.2014 - 11:38 | 29.06.2014 - 13:20 | 01:41:40 | twisterntb | 100k |
| 5 | 30.06.2014 - 02:50 | 30.06.2014 - 02:55 | 00:05:10 | mail.kolej | 10M |
| 6 | 30.06.2014 - 03:12 | 30.06.2014 - 03:19 | 00:07:40 | mail.kolej | 10M |
| 7 | 30.06.2014 - 03:33 | 30.06.2014 - 08:03 | 04:30:00 | mail.kolej | 10M |
| 8 | 01.07.2014 - 00:09 | 01.07.2014 - 12:59 | 12:49:56 | mail.kolej | 10M |
| 9 | 02.07.2014 - 01:31 | 02.07.2014 - 12:48 | 10:16:57 | mail.kolej | 10M |
| 10 | 02.07.2014 - 22:01 | 07.07.2014 - 07:49 | 104:48:32 | mail.kolej | 10M |
| 11 | 08.07.2014 - 00:04 | 08.07.2014 - 23:46 | 23:42:00 | mail.kolej | 1M |

Table 4.2: Crawler's sessions running times

Next table 4.3 summarizes these crawler runs. It describes the time lapse of each run, how many pages it went through, how many sessions did it have and it contains importing seeds informations.

| name of run | pages | time | sessions | number of seeds | importing seeds time |
|---|---|---|---|---|---|
| 10k | 10,000 | 5:40:02 | 1 | 4,119 | 0:14:00 |
| 100k | 100,000 | 4:33:24 | 5 | 4,119 | 0:23:44 |
| 1M | 2,975,398 | 23:42:00 | 9 | 3,361 | 0:07:07 |
| 10M | 8,238,827 | 132:00:00 | 10 | 3,361 | 0:10:44 |

Table 4.3: Crawler's runs – information

As you can see 10k was slower then 100k. It is because of crawler's modifications during the testing. Also seeds-importing time is different. It is also the result of the changes.
After the first two runs we added filtering of the seeds. That is why the number of imported seeds lowered from 4,119 to 3,361.
What is interesting, the 10M run was not completed. It had database that large, that getting data out of it would last too long and the collection of documents

was hardly running.
1M run went through over 1M pages. This was caused by wrong configuration of value – maximum number of pages to fetch – it was set to 10,000,000 instead of 1,000,000 as it was originally meant to be.

The most of the sessions was run only at night so that the collection of documents would not slow down the local network. When it showed up not to be slowing it down, the sessions were run also during the day.

### 4.5.1 Sessions And Runs Statistics

| ID | importing seeds time | processed | XQuery | text files | pre section |
|---|---|---|---|---|---|
| 1 | 0:23:44 | 4,569 | 25 | 0 | 4 |
| 2 | 0:00:00 | 78,590 | 1,506 | 0 | 333 |
| 3 | 0:14:00 | 10,000 | 1,481 | 5 | 379 |
| 4 | 0:00:00 | 16,841 | 202 | 3 | 194 |
| 5 | 0:07:07 | 15,310 | 393 | 0 | 97 |
| 6 | 0:00:00 | 26,751 | 435 | 0 | 57 |
| 7 | 0:00:00 | 450,209 | 1,455 | 47 | 199 |
| 8 | 0:00:01 | 1,891,039 | 120 | 63 | 181 |
| 9 | 0:00:03 | 552,748 | 166 | 8 | 1,964 |
| 10 | 0:00:05 | 5,302,770 | 1,022 | 1,407 | 70 |
| 11 | 0:10:44 | 2,975,398 | 2,727 | 106 | 518 |

Table 4.4: Crawler's sessions statistics

Table 4.4 shows each session's statistics. As we can see, seeds where imported into database only once.
The column **XQuery** of the table represents numbers of downloaded XQuery files (XQuery extension type), the **text files** is number of text files (possible XQuery programs in files with other extension than XQuery extension ) and **pre section** represents number of `<pre>` sections from web pages (also possible XQuery programs).

During one crawler's statistic session, the **crawler** stopped responding. We terminated it and lost statistic. That is why some (downloaded files) of sums of statistics can be approximate.
Table 4.5 shows numbers of the unique (with no duplicities) files of different types in performed runs.

Graph 4.2 shows number of the downloaded files during the time lapse. The particular lines represent particular crawler's runs.

Graph 4.3 shows logarithmic dependency of processed pages on a variable 'time'. The particular lines represent particular crawler's runs.

| name of run | XQuery | maybe XQuery | pre section | together |
|---|---|---|---|---|
| 10k | 1,467 | 5 | 355 | 1,827 |
| 100k | 1,615 | 3 | 526 | 2,144 |
| 1M | 2,727 | 106 | 518 | 3,351 |
| 10M | 3,631 | 2,342 | 2,528 | 8,501 |

Table 4.5: Crawler's runs – downloaded files

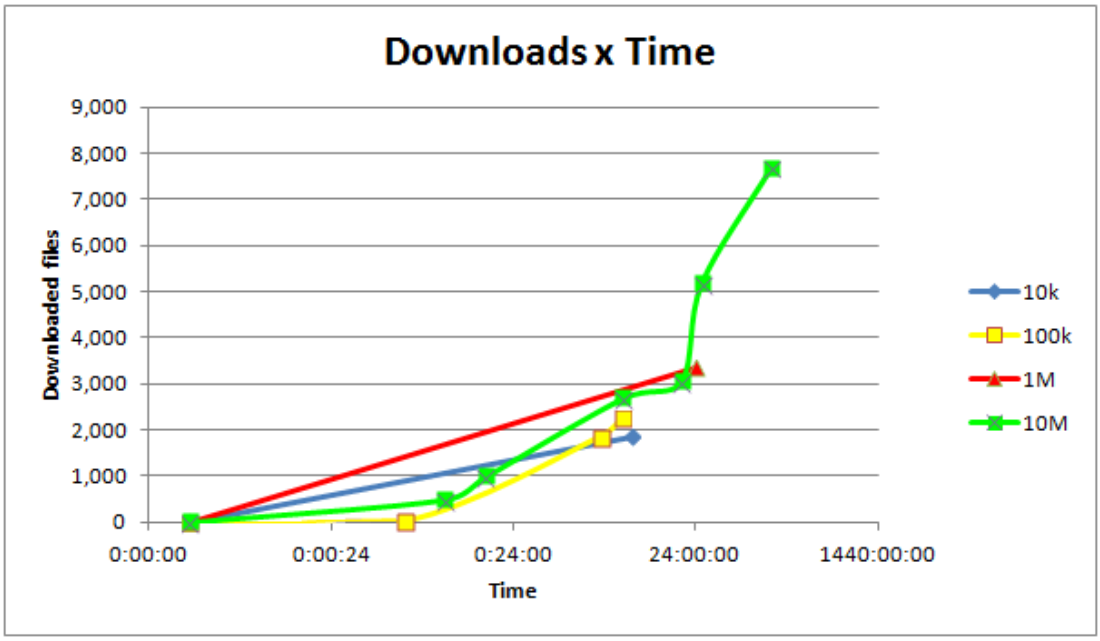



Figure 4.2: Downloads x Time Graph

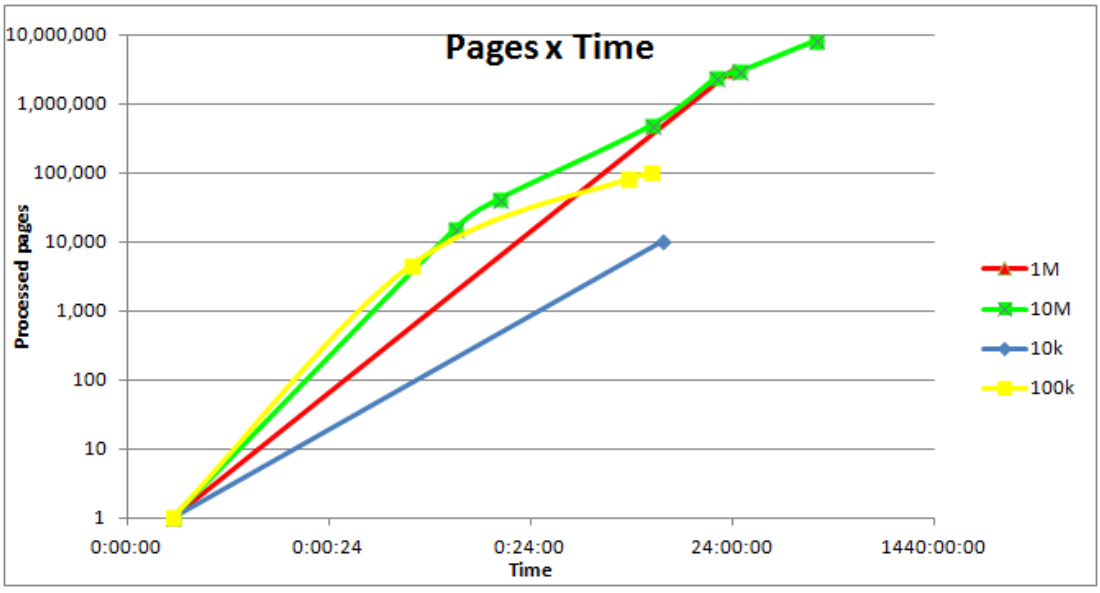



Figure 4.3: Pages x Time Graph

### 4.5.2 Crawler's Mistakes

During the runs several errors occurred. The most of them were immediately fixed and some of them (irrelevant ones) were ignored.
The first error was that politeness delay was not working the way it was supposed to be. The politeness delay was corrected.
One of other errors was that the **crawler** was requesting the DNS this way:

```
...197.63:81
...210.9:8991
...77.206:8991
...unimaas.nl:8180
...eur.nl:8080
...unizh.ch:8331
```

Fetcher part did not separate domain name and port correctly. This happened only few times so it was ignored.
There were DNS requests like:

```
...pubs.marklogic.com IN ANY + (78.128.192.7)
...pubs.marklogic.com.kolej.mff.cuni.cz IN ANY + (78.128.192.7)
...pubs.marklogic.com.mgmt.kolej.mff.cuni.cz IN ANY + (78.128.192.7)
...pubs.marklogic.com.ms.mff.cuni.cz IN ANY + (78.128.192.7)
```

This meant that fetcher added default suffixes to the domain name. This resulted only in doing more DNS requests than necessary.
Because the part of the fetcher responsible for this two mistakes was external (only executables) and the mistakes were irrelevant we did not repair them.

The **crawler** also managed to downloaded the text files and the HTML 'pre' sections without XQuery keywords or with empty contain. Some sort of filtering will be required.

## 4.6 Summary

The **crawler** processed 11,324,225 web pages (with duplicities). 10,591 files with unique file name as possible XQuery program was saved out of the total number. We removed 59 empty files. Also we run DoubleKiller program to find the duplicities (where file size and CRC32/checksum matches). It found 2,221 duplicities out of which approximately 660 where nearly the same file name (file names have different non-alphanumeric characters, that means they were from same source, but saved in different runs).
So we have 8,311 (10,591 - (59 + 2,221)) downloaded files without empty files and duplicities.
This means we got less then 1% of possible XQuery programs from processed pages (as we assumed).

# 5. Data Analysis

We obtained 8 311 files using the **crawler** in the previous chapter. In this chapter we will analyze the data. Before we can perform the analysis, we need to convert the data.
We did not manage to run compiled **Analyzer** neither with Java SDK 7 nor with Java SDK 8. So instead of using **Analyzer** we will use XQConverter (command line program) to convert the data and Altova XMLSpy to analyze them.

## 5.1 Data Conversion

Before converting obtained files, we need to remove tag `<pre>` from some of them. During the extraction of some of the files from web pages, we let tag `<pre>` in them.
For this purpose we will use this script:

```
Get-ChildItem . -Exclude *.xml,*.jar,output.txt,errors.txt,lib ‘
-Recurse | ForEach-Object { (Get-Content $_) -Replace "^<pre>","" ‘
-Replace "</pre>$","" | Set-Content -Path $_ }
```

This script takes files, which does not have `xml` or `jar` extension, or are not `output.txt` or `errors.txt` or directory `lib`. For every selected file it takes its content and removes the tag `<pre>` from begin of file content and the tag `</pre>` from end of file content.

With similar script (instead of replacing something in every file) we will run XQConvert program on every file:

```
java -jar "XQConvert.jar" "$_" "$_.xml"
```

This will produce files with `xml` extension. It will be run in script, so output file will always be created. If output file has 0B size, error occurs. Error output is redirected into the file `errors.txt`.
By analyzing this file we can find out what was cause of errors during file conversion (from XQuery programs into XML document representation).

826 files were converted out of 8,311 files. There occurred an error for 7,485 files. Using this script we can easily count number and type of the errors:

```
Get-Content "errors.txt" | Where-Object {$_ -Match ‘
"java : Illegal character"} | Measure-Object
Get-Content "errors.txt" | Where-Object {$_ -Match ‘
"java : Error : Syntax"} | Measure-Object
```

It shows that number of lexical errors was 7,215. By manually going through some of the data, they were mostly mistakes where the space was on wrong position, e.g. ‘`for $a: =`‘ instead of ‘`for $a :=`‘.
It also shows that number of syntactical errors was 264, which is much less compared to number of lexical errors.
Some of the lexical errors occurred because some of the data where not XQuery programs. So this conversion was like some kind of the last filter.

We took 826 converted files (which represent valid XQuery programs) and checked for duplicates with DoubleKiller. It showed up that 60 files where duplicates (7.3%). This means the original files were different procedures they perform were the same.

The final group consists of 766 files for the analysis.

## 5.2 XQuery Grammar Symbols

The first analysis will be occurrence of XQuery grammar symbols in programs. These symbols are explained in the end of the thesis. To get this occurrence we need well designed XPath expressions for searching in the files.
Tables 5.1, 5.2, 5.3 and 5.4 show a selection of XQuery grammar symbols for determining their frequencies in the files. We separated these into more tables (10-15 entries per table) for better arrangement.

| **XQuery Grammar** | **XPath expression** |
|---|---|
| AdditiveExpr | `//Operator[@class="additive"]` |
| AndExpr | `//Operator[@class="logical" and @name="and"]` |
| BaseURIDecl | `//BaseURIDecl` |
| BoundarySpaceDecl | `//BoundarySpaceDecl` |
| CastableExpr | `//Operator[@class="type-test" and @name="castable-as"]` |
| CastExpr | `//Operator[@class="type-cast" and @name="cast-as"]` |
| ComputedConstructor | `//Constructor[@kind="computed"]` |
| ConstructionDecl | `//ConstructionDecl` |
| ContextItemExpr | `//ContextItem` |
| CopyNamespacesDecl | `//CopyNamespacesDecl` |
| DefaultCollationDecl | `//DefaultCollationDecl` |
| DefaultNamespaceDecl | `//DefaultNamespaceDecl` |
| DirectConstructor | `//Constructor[@kind="direct"]` |
| EmptyOrderDecl | `//EmptyOrderDecl` |
| ExtensionExpr | `//Extension` |

Table 5.1: XQuery symbols and appropriate XPath expressions – part 1

| XQuery Grammar | XPath expression |
|---|---|
| FLWORExpr | `//FLWOR` |
| ForClause | `//ForClause` |
| ForwardAxis | `//Axis[@direction="forward"]` |
| FunctionCall | `//FunctionCall` |
| FunctionDecl | `//FunctionDecl` |
| GeneralComp | `//Operator[@class="comparison" and @subclass="general"]` |
| IfExpr | `//IfExpr` |
| InstanceofExpr | `//Operator[@class="type-test" and @name="instance-of"]` |
| IntersectExceptExpr | `//Operator[@class="set" and (@name="difference" or @name="intersection")]` |
| LetClause | `//LetClause` |
| Literal | `//Literal` |
| ModuleDecl | `//ModuleDecl` |
| ModuleImport | `//ModuleImport` |
| MultiplicativeExpr | `//Operator[@class="multiplicative"]` |
| NamespaceDecl | `//NamespaceDecl` |

Table 5.2: XQuery symbols and appropriate XPath expressions – part 2

| XQuery Grammar | XPath expression |
|---|---|
| NodeComp | `//Operator[@class="comparison" and @subclass="node"]` |
| OptionDecl | `//OptionDecl` |
| OrderByClause | `//OrderByClause` |
| OrderedExpr | `//OrderedExpr` |
| OrderingModeDecl | `//OrderingModeDecl` |
| OrExpr | `//Operator[@class="logical" and @name="or"]` |
| PathExpr | `//Path` |
| Prolog | `//Prolog` |
| QuantifiedExpr (every) | `//QuantifiedExpr[@quantifier="every"]` |
| QuantifiedExpr (some) | `//QuantifiedExpr[@quantifier="some"]` |
| QueryBody | `//QueryBody` |
| RangeExpr | `//Operator[@class="range"]` |
| ReverseAxis | `//Axis[@direction="reverse"]` |
| SchemaImport | `//SchemaImport` |

Table 5.3: XQuery symbols and appropriate XPath expressions – part 3

We found out that Altova XMLSpy refused to do XPath expression evaluation over 8 files. Remaining 758 files were analyzed.

Table 5.5 shows number of the files where XQuery grammar symbols were found. It is measured in percent for better visualization. The entries are sorted alphabetically.

| XQuery Grammar | XPath expression |
|---|---|
| TreatExpr | `//Operator[@class="type-cast" and @name="treat-as"]` |
| TypeswitchExpr | `//Typeswitch` |
| UnaryExpr | `//Operator[@class="unary"]` |
| UnionExpr | `//Operator[@class="set" and @name="union"]` |
| UnorderedExpr | `//UnorderedExpr` |
| ValidateExpr | `//ValidateExpr` |
| ValueComp | `//Operator[@class="comparison" and @subclass="value"]` |
| VarDecl | `//VarDecl` |
| VarRef | `//VarRef` |
| WhereClause | `//WhereClause` |

Table 5.4: XQuery symbols and appropriate XPath expressions – part 4

| XQuery Grammar | occurrence | XQuery Grammar | occurrence |
|---|---|---|---|
| AdditiveExpr | 15.30% | ModuleImport | 25.99% |
| AndExpr | 54.49% | MultiplicativeExpr | 7.78% |
| BaseURIDecl | 0.26% | NamespaceDecl | 51.06% |
| BoundarySpaceDecl | 0.26% | NodeComp | 1.45% |
| CastableExpr | 12.66% | OptionDecl | 2.77% |
| CastExpr | 0.40% | OrderByClause | 21.90% |
| ComputedConstructor | 8.05% | OrderedExpr | – |
| ConstructionDecl | 0.40% | OrderingModeDecl | 5.54% |
| ContextItemExpr | 16.23% | OrExpr | 33.77% |
| CopyNamespacesDecl | 0.13% | PathExpr | 77.97% |
| DefaultCollationDecl | – | Prolog | 68.73% |
| DefaultNamespaceDecl | 1.06% | QuantifiedExpr (every) | 2.24% |
| DirectConstructor | 75.20% | QuantifiedExpr (some) | 1.72% |
| EmptyOrderDecl | – | QueryBody | 92.61% |
| ExtensionExpr | 0.26% | RangeExpr | 2.90% |
| FLWORExpr | 84.17% | ReverseAxis | 10.29% |
| ForClause | 70.71% | SchemaImport | 1.85% |
| ForwardAxis | 77.70% | TreatExpr | 0.26% |
| FunctionCall | 88.39% | TypeswitchExpr | 1.45% |
| FunctionDecl | 48.94% | UnaryExpr | 7.78% |
| GeneralComp | 66.89% | UnionExpr | 13.19% |
| IfExpr | 50.13% | UnorderedExpr | – |
| InstanceofExpr | 2.64% | ValidateExpr | 0.92% |
| IntersectExceptExpr | 1.32% | ValueComp | 18.47% |
| LetClause | 74.27% | VarDecl | 54.35% |
| Literal | 88.65% | VarRef | 88.13% |
| ModuleDecl | 7.39% | WhereClause | 46.97% |

Table 5.5: Probability of occurrence of given XQuery symbols in files

The biggest occurrence has ‘QueryBody‘. It occurred in 92.61% of files. This is the only grammar symbol with over 90% occurrence.
There are 12 rarely used grammar symbols. Each one of them is used in less than 1% of files. Namely they are ‘BaseURIDecl‘, ‘BoundarySpaceDecl‘, ‘CastExpr‘, ‘ConstructionDecl‘, ‘CopyNamespacesDecl‘, ‘DefaultCollationDecl‘, ‘EmptyOrderDecl‘, ‘ExtensionExpr‘, ‘OrderedExpr‘, ‘TreatExpr‘, ‘UnorderedExpr‘ and ‘ValidateExpr‘.

| **XQuery Grammar** | **occurrence** | **XQuery Grammar** | **occurrence** |
|---|---|---|---|
| AdditiveExpr | 472 | ModuleImport | 448 |
| AndExpr | 5,056 | MultiplicativeExpr | 223 |
| BaseURIDecl | 2 | NamespaceDecl | 1,006 |
| BoundarySpaceDecl | 2 | NodeComp | 36 |
| CastableExpr | 1,689 | OptionDecl | 25 |
| CastExpr | 6 | OrderByClause | 586 |
| ComputedConstructor | 518 | OrderedExpr | – |
| ConstructionDecl | 3 | OrderingModeDecl | 42 |
| ContextItemExpr | 453 | OrExpr | 3,774 |
| CopyNamespacesDecl | 1 | PathExpr | 24,236 |
| DefaultCollationDecl | – | Prolog | 521 |
| DefaultNamespaceDecl | 8 | QuantifiedExpr (every) | 42 |
| DirectConstructor | 24,176 | QuantifiedExpr (some) | 33 |
| EmptyOrderDecl | – | QueryBody | 702 |
| ExtensionExpr | 2 | RangeExpr | 50 |
| FLWORExpr | 8,136 | ReverseAxis | 1,034 |
| ForClause | 5,000 | SchemaImport | 19 |
| ForwardAxis | 36,377 | TreatExpr | 5 |
| FunctionCall | 64,102 | TypeswitchExpr | 25 |
| FunctionDecl | 8,231 | UnaryExpr | 274 |
| GeneralComp | 15,085 | UnionExpr | 574 |
| IfExpr | 10,563 | UnorderedExpr | – |
| InstanceofExpr | 180 | ValidateExpr | 11 |
| IntersectExceptExpr | 49 | ValueComp | 952 |
| LetClause | 20,278 | VarDecl | 1,928 |
| Literal | 38,109 | VarRef | 86,783 |
| ModuleDecl | 56 | WhereClause | 1,717 |

Table 5.6: Number of occurrences of given XQuery symbols in files

Table 5.6 shows number of occurrences of these XQuery grammar symbol in all 758 files. The entries are sorted alphabetically.
Occurrences over 10,000 hits have these grammar symbols ‘DirectConstructor‘, ‘ForwardAxis‘, ‘FunctionCall‘, ‘GeneralComp‘, ‘IfExpr‘, ‘LetClause‘, ‘Literal‘, ‘PathExpr‘ and ‘VarRef‘. Two most used symbols are VarRef and FunctionCall – these respresent normal programs.
The lowest occurrence (lower then 10 occurrences) have these twelve grammar symbols ‘BaseURIDecl‘, ‘BoundarySpaceDecl‘, ‘CastExpr‘, ‘ConstructionDecl‘,

‘CopyNamespacesDecl‘, ‘DefaultCollationDecl‘, ‘DefaultNamespaceDecl‘, ‘EmptyOrderDecl‘, ‘ExtensionExpr‘, ‘OrderedExpr‘, ‘TreatExpr‘ and ‘UnorderedExpr‘.

As we can see in rarely used grammar symbols there is nearly no difference between number of hits in the files and number of the files. The only difference is in these two symbols ‘DefaultNamespaceDecl‘ (8 hits, used in 1.06% of files) and ‘ValidateExpr‘ (11 hits, used in 0.92% of files).
The most used grammar symbols are ‘QueryBody‘ (702 hits, used in 92.61% of files) and ‘VarRef‘ (86,783 hits, used in 88.13% of files).

Some grammar symbols did not occur in any program. It means these symbols are probably not used in XQuery programs. These are the mentioned symbols: ‘EmptyOrderDecl‘, ‘DefaultCollationDecl‘, ‘OrderedExpr‘ and ‘UnorderedExpr‘.

## 5.3 XPath queries in XQuery programs

XPath queries are used in XQuery programs as PathExpr grammar symbol. This symbol has 24,236 hits in the files.

We used these XPath expressions to get some useful analysis:

```
//Path/count(descendant::*)
//Path/count(Step)
//Path[Step/Predicates]
//Path/descendant::FunctionCall[not (@name=//FunctionDecl/@name)]
//Path/Step[last()]/Axis[@kind!="attribute"]/NameTest
//Path/Step[last()]/Axis[@kind="attribute"]
//Path/Step[last()]/Axis/KindTest[@kind="text"]
//Axis[@kind!="attribute"]/NameTest
//Axis[@kind="attribute"]
//Axis/KindTest[@kind="text"]
```

To get the average and maximum value of given number we used this script (with some modifications for every file):

```
gc inputfile.txt | where {$_ -match "xs:integer"} | foreach ‘
{$_ -replace "^xs:integer ",""} | sort | measure -average ‘
-maximum -minimum -sum > inputfile-statistics.txt
```

We counted number of elements under this node in XML representation and we found out that average number of XML elements (from XML representation) is 8.97 and the maximum number is 391. That maximum number of 391 elements is greater then expected (expected number was approximately 100).

As for number of steps of XPath we got average 2.37 and maximum of 14 steps. These values are quite similar to what was expected.
Only 5,749 out of 24,236 XPath queries uses the predicates. That represents only 23.7%. We expected the ratio around 50%.

To get XPath built-in function names we used this script:

```
gc PathFunctions.txt | where {$_ -match "FunctionCall name"} | ‘
foreach {$_ -replace ’^FunctionCall name="’,’’ -replace ’"$’,’’} | ‘
sort | sc PathFunctions-functions.txt
```

Afterwards we used Java program that took duplicate lines and output unique lines with number of duplicities. After comparing this output with list of XPath built-in functions we get (we removed function there are not XPath built-in function):

| **XPath function name** | **count** | **XPath function name** | **count** |
|---|---|---|---|
| fn:boolean | 2 | fn:name | 957 |
| fn:ceiling | 3 | fn:namespace-uri | 30 |
| fn:collection | 49 | fn:namespace-uri-for-prefix | 1 |
| fn:concat | 188 | fn:node-name | 33 |
| fn:contains | 56 | fn:normalize-space | 238 |
| fn:count | 156 | fn:not | 145 |
| fn:data | 49 | fn:number | 20 |
| fn:deep-equal | 2 | fn:position | 155 |
| fn:distinct-values | 2 | fn:QName | 1 |
| fn:doc | 2,170 | fn:replace | 29 |
| fn:doc-available | 4 | fn:resolve-QName | 1 |
| fn:empty | 34 | fn:resolve-uri | 1 |
| fn:ends-with | 19 | fn:root | 1 |
| fn:error | 1 | fn:starts-with | 72 |
| fn:exists | 74 | fn:string | 505 |
| fn:false | 67 | fn:string-join | 12 |
| fn:id | 2 | fn:string-length | 399 |
| fn:index-of | 46 | fn:substring | 15 |
| fn:last | 13 | fn:substring-after | 20 |
| fn:local-name | 356 | fn:substring-before | 43 |
| fn:lower-case | 292 | fn:tokenize | 24 |
| fn:matches | 54 | fn:trace | 1 |
| fn:max | 2 | fn:true | 29 |
| fn:min | 2 | fn:upper-case | 19 |
| fn:month-from-date | 7 | fn:year-from-date | 1 |

Table 5.7: Built-in XPath function used in XQuery programs

Table 5.7 shows number of uses of XPath built-in functions. The functions not mentioned in the table were not used in any of 758 XQuery files.

The most occurrences had function fn:doc (2,170 occurrences). This function is used to acquire data from external documents.

In the last step there were 14,032 XPath queries on elements, 5,450 on attributes and 2,035 on text nodes.
In any step there were 24,846 XPath queries on elements, 5,470 on attributes and 2,035 on text nodes.
That means XPath queries about the text nodes only in the last step. On attributes there where 20 of them that they were not in the last step queried. This is an interesting result. In one case there was function 'matches' after attribute query step used in the last step.

## 5.4 FLWOR Analysis

'FLWORExpr' in XQuery is practically a base stone. It was found in 84.17% of files (8,136 hits together).

We used these XPath expressions to get some useful analysis. The last one was used more times with little alternation (adding nodes `descendant::FLWOR` to the last two conditions in predicate):

```
//FLWOR/*/ForClause/parent::*/count(ForClause)
//FLWOR/*/LetClause/parent::*/count(LetClause)
//FLWOR/WhereClause/parent::*/count(WhereClause)
//FLWOR/OrderByClause/parent::*/count(OrderByClause)
/descendant::FLWOR[not (ancestor::FLWOR) and descendant::FLWOR and
    not (descendant::FLWOR/descendant::FLWOR)]
```

'FLWORExpr' is made from this clauses:

**ForClause**
: It is used 5,000 times together and it occurs in 4,564 FLWORs (from 8,136, this means 56.1%). Average number of the use of 'ForClause' in 'FLWORExpr' is 1.1 and maximum is 13. We expected maximum to be 2-5.

**LetClause**
: It is used 20,278 times together and it occurs in 6,064 FLWORs (from 8,136, this means 74.5%). Average number of the use of 'LetClause' in 'FLWORExpr' is 3.3 and maximum is 113. We expected maximum to be approximately 10.

**WhereClause**
: It is used 1,717 times together. It occurs in 1,717 FLWORs (from 8,136,

this means 21.1%). Average and maximum number is 1.

**OrderByClause**
It is used 586 times together. It occurs in 586 FLWORs (from 8,136, this means 7.2%). Average and maximum number is 1.

**ReturnClause**
Return is required part of FLWOR, so it occurs in every 'FLWORExpr' just once.

Mostly used clause is 'LetClause' and the least used is 'OrderByClause'.
Nested FLWORs are common in XQuery programs. We found exactly 2 times nested 'FLWORExpr' in 887 cases. We found exactly 3 times nested 'FLWORExpr' in 83 cases. We found exactly 4 times nested 'FLWORExpr' in 39 cases and we found 5 and more times nested 'FLWORExpr' in 13 cases.

## 5.5 Functions And Variables Analyses

Functions and variables have the most occurrences of all grammar symbols. For function analysis we will use these XPath expressions:

```
//FunctionDecl[@name=descendant::FunctionCall/@name]
//FunctionDecl[not (descendant::FunctionCall)]
//FunctionDecl[not (//FunctionCall/@name=@name)]
count(//FunctionCall[@name=//FunctionDecl/@name])
//FunctionCall[not (@name=//FunctionDecl/@name)]
//FunctionCall[contains(@name,'doc')]
//FunctionDecl[descendant::IfExpr or descendant::Typeswitch]
//FunctionDecl/count(descendant::*)
```

Functions have 8,231 declarations ('FunctionDecl') and 64,102 calls ('FunctionCall'). 47,917 calls are of built-in or external functions.
We found 188 recursive functions. We also found 870 functions that do not call any other functions in their declaration.
We found 1,443 functions with only 'FunctionDecl' and no 'FunctionCall' in the same program.
Function 'doc()' was found in 2,676 times, that means in 2,676 cases XQuery program is using data from external source.
We also found 3,921 function declarations that uses conditions.
As for number of elements, function has in average 72.5 elements in XML representation and maximum is 1,985 elements (in XML representation).
There is 21.4 average calls of declared functions per file (maximum is 525)

For variable analysis we will use these XPath expressions:

```
//VarDecl/count(descendant::*)
//VarDecl/VarValue/Literal[@type="string"]/@value
//VarDecl[not (@name=//VarRef/@name)]
count(//VarRef[@name=//VarDecl/@name])
//VarDecl/VarValue[@external="true"]
```

Variables have 1,928 declarations (‘VarDecl‘) and 86,783 referencies (‘VarRef‘).
External declared variables are in 358 cases. 234 times the variables are only declared and 882 variables are string.
Average length of the string variable is 36.5 characters and maximum is 422 characters.
Declared variables are used 14 times per file on average (maximum is 267 times per file).
Variable declaration has 6.2 elements in XML representation on average (maximum is 850 elements in XML representation).

## 5.6 Constructed Returns And Other Analyses

We took the query from Subsection 3.1.5 and changed it to:

```
/*[last() and not (/Module/@version or //ModuleDecl or //Prolog or
    //LetClause or //WhereClause or //OrderByClause or
    //ForClause/Type or //ForClause/@posname or
    //QuantifiedExpr/InClauses/InClause/Type or //Typeswitch or
    //Extension or //ValidateExpr or //OrderedExpr or
    //UnorderedExpr or //Constructor)]
```

This way we get programs that do not contain any of these conditions – we will get number of the files that are both XPath 2.0 and XQuery 1.0.
It is in 24 cases.

We also looked at ‘ComputedConstructor‘ (518 hits) and ‘DirectConstructor‘ (24,176 hits) and we can see that ‘DirectConstructor‘ is used more frequently than ‘ComputedConstructor‘.

In case of quantifiers the frequency of ‘some‘ and ‘every‘ is nearly the same. Quantifier ‘every‘ is used 42 times and quantifier ‘some‘ is used 33 times.

To get the analysis of constructed returns we will use the following XPath expressions:

```
//ReturnClause/count(descendant::*)
```

```
//ReturnClause[descendant::IfExpr or descendant::FunctionCall or
    descendant::Constructor[@kind="computed"] or
    descendant::Typeswitch]
//ReturnClause[descendant::Typeswitch]
//ReturnClause[descendant::IfExpr]
//ReturnClause[descendant::FunctionCall]
//ReturnClause[descendant::Constructor[@kind="computed"]]
//ReturnClause[child::Path or child::Constructor[@kind="direct"]]
//QueryBody[child::IfExpr or child::FunctionCall or
    child::Constructor[@kind="computed"] or child::Typeswitch]
//QueryBody[child::Path or child::Constructor[@kind="direct"]]
```

The most of the constructed data in XQuery programs are from FLWOR's 'ReturnClause'. The rest of them are directly in the query body ('QueryBody').

Average number of elements (in XML representation) in return clause is 39.4 and maximum is 6,788.

If we consider grammar symbols like 'IfExpr', 'FunctionCall', 'Typeswitch', etc. (symbols that can be used to create different data) we get 5,871 FLWORs with more complex 'ReturnClause' (out of 8,136 FLWORs). If we take symbols independently we get 14 FLWORs for 'Typeswitch', 2,411 FLWORs for 'IfExpr', 5,698 FLWORs for 'FunctionCall' and 246 FLWORs for 'ComputerConstructor'. If we do same analysis for 'QueryBody', we get 271 query bodies (out of 702 query bodies) with more complex return.

If we try something similar with 'PathExpr' and 'DirectConstructor' we get 2,130 FLWORs with these symbols (out of 8,136 FLWORs) and 196 query bodies with simpler return (out of 702 query bordies). This kind of return is highly unlikely to create different data.

This tells us that in about 1/3 of the cases we use more complex output of XQuery program (functionality going to XSLT - which changes XML data) and in about 1/4 we use simpler output of XQuery program (just taking some elements out of XML tree).

## 5.7 Summary

In this chapter we performed the analyses of the gathered data. Analysis description was included.

Analysis showed us number of files containing XQuery grammar symbols and frequencies of those symbols in all those files, then frequencies of the use of the XPath expressions in XQuery programs, function's and variable's behaviours in XQuery programs, the use of FLWOR and its specific clauses and the complexity of created return of XQuery programs. This showed us how XQuery programs

are used in general.
We also found out that we had 24 files that were not only XQuery 1.0 programs but also XPath 2.0 expressions at the same time
This analysis can be used for optimalization of XQuery evaluators.

# Conclusion

The aim of this thesis was to gather real-world XQuery programs that would be valid and collect as many of them as possible. The analysis of these gathered data was also part of the thesis.
The analysis of existing crawlers for gathering was described. The chosen crawler was modified in several ways. It was necessary to set the following things: what data to save, what data not to crawl, pause option and delay between request (so that the crawler behaves politely). In other aspects it was well chosen program.
It crawled approximately 11,000,000 pages and gathered approximately 10,000 programs out of which approximately 700 were valid XQuery programs. That is approximately 7% success rate.
XQConverter is program used to convert XQuery programs to XML representation of XQuery program and thus also validate that it is an XQuery program. Most of errors during conversion were lexical errors in programs. Only a few errors were syntactical. This conversion verified that XQConverter is a well implemented program.
This XML representation was afterwards queried by XPath expressions to get awaited analyses.
The results showed statistical information about XQuery programs.

The further adjustments to this thesis would be more detailed analysis, downloading also XML documents referred to in valid XQuery programs, connecting these XQuery programs and XML documents.
Analysis of these XML documents and connected XQuery programs would require special program.

# Bibliography


[1] Altova XMLSpy 2014. URL http://www.altova.com/simpledownload2c.html?gclid=COyGr6bM278CFW-WtAod2HoATg. [cit. 2014-07-18].

[2] Apache License 1.0, 1.1, 2.0, . URL http://www.apache.org/licenses/. [cit. 2014-07-26].

[3] Apache Nutch v. 1.8, . URL https://nutch.apache.org/. [cit. 2014-03-26].

[4] BSD License Definition. URL http://www.linfo.org/bsdlicense.html. [cit. 2014-07-26].

[5] Bixo: A Web Mining Toolkit v. 0.9.1. URL http://openbixo.org. [cit. 2014-03-26].

[6] Cyclic redundancy check. URL http://en.wikipedia.org/wiki/Cyclic_redundancy_check. [cit. 2014-07-18].

[7] C++ programming language. URL http://www.cplusplus.com/. [cit. 2014-07-26].

[8] C# Language. URL http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx. [cit. 2014-07-26].

[9] Custom Search Google API. URL https://developers.google.com/custom-search/. [cit. 2014-07-16].

[10] Document Type Definition. URL http://www.w3.org/TR/2004/REC-xml11-20040204/#NT-doctypedecl. [cit. 2014-07-26].

[11] DoubleKiller v. 1.6.2.82. URL http://www.bigbangenterprises.de/en/doublekiller/. [cit. 2014-07-18].

[12] GPL Licenses. URL http://www.gnu.org/licenses/. [cit. 2014-07-26].

[13] Google Web Search API, . URL https://developers.google.com/web-search/. [cit. 2014-03-26].

[14] Googlebot, . URL https://support.google.com/webmasters/answer/70897?hl=en. [cit. 2014-07-16].

[15] How DNS Works. URL http://www.itgeared.com/articles/1354-domain-name-system-dns-tutorial-overview/. [cit. 2014-07-24].

[16] JSpider v. 0.5.0. URL http://j-spider.sourceforge.net/. [cit. 2014-03-26].

[17] Java 8, . URL http://www.oracle.com/technetwork/java/index.html. [cit. 2014-07-26].

[18] Java 7 Networking Properties, . URL http://docs.oracle.com/javase/7/docs/technotes/guides/net/properties.html. [cit. 2014-07-16].

[19] JoBo v. 1.4. URL http://www.matuschek.net/software/jobo/index.html. [cit. 2014-03-26].

[20] Larbin: Multi-purpose Web Crawler v. 2.6.3. URL http://larbin.sourceforge.net/index-eng.html. [cit. 2014-03-26].

[21] The MIT License (MIT). URL http://opensource.org/licenses/MIT. [cit. 2014-07-26].

[22] NetBeans IDE 8.0. URL https://netbeans.org/. [cit. 2014-07-26].

[23] OpenWebSpider v. 0.1.4. URL http://www.openwebspider.org/. [cit. 2014-04-02].

[24] Outwit v. 4.0. URL http://www.outwit.com/. [cit. 2014-06-13].

[25] PHP scripting language 5.5.15. URL https://php.net/. [cit. 2014-07-26].

[26] Python. URL https://www.python.org/. [cit. 2014-07-26].

[27] Scrapy v. 0.22. URL http://scrapy.org/. [cit. 2014-04-02].

[28] W3C Technical Reports and Publications. URL http://www.w3.org/TR/. [cit. 2013-11-08].

[29] The World Wide Web Consortium (W3C). URL http://www.w3.org/. [cit. 2014-07-26].

[30] Web-Harvest v. 2.0, . URL http://web-harvest.sourceforge.net/. [cit. 2014-03-26].

[31] WebLech URL Spider v. 0.0.4, . URL http://weblech.sourceforge.net/. [cit. 2014-03-26].

[32] WebSPHINX: A Personal, Customizable Web Crawler v. 0.5, . URL http://www-2.cs.cmu.edu/~rcm/websphinx/. [cit. 2014-03-26].

[33] XML Query Test Suite 1.0. URL http://dev.w3.org/2006/xquery-test-suite/PublicPagesStagingArea/. [cit. 2014-03-10].

[34] XYLEME PROJECT. URL http://xml.coverpages.org/xyleme.html. [cit. 2014-03-26].

[35] Abot C# Web Crawler v. 1.2.3. URL http://code.google.com/p/abot/. [cit. 2014-04-02].

[36] RFC 791 INTERNET PROTOCOL, September 1981. URL http://tools.ietf.org/html/rfc791. [cit. 2014-07-26].

[37] W. Sherlock E. Nolan V. Sadlon M. Tamas K. Dufkova J. Podhorny Ch. Christacopoulos, L. Galambos and M. Pirchala. Egothor2 v. 3.1.5. URL http://www.egothor.org/cms/egothor2. [cit. 2014-03-26].

[38] J. Clark. XSL Transformations (XSLT) Version 1.0, November 1999. URL http://www.w3.org/TR/xslt. [cit. 2014-07-26].

[39] D. Florescu M. Marchiori D. Chamberlin, P. Fankhauser and J. Robie. XML Query Use Cases, March 2007. URL http://www.w3.org/TR/xquery-use-cases/. [cit. 2014-04-09].

[40] A. Malhotra C. M. Sperberg-McQueen D. Peterson, S. Gao and H. S. Thompson. XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes, April 2012. URL http://www.w3.org/TR/xmlschema11-2/. [cit. 2014-07-26].

[41] A. Le Hors D. Raggett and I. Jacobs. HTML 4.01 Specification, December 1999. URL http://www.w3.org/TR/REC-html40/. [cit. 2014-07-26].

[42] L. Galamboš and Others. Egothor v. 1.3. URL http://www.egothor.sf.net/. [cit. 2014-03-26].

[43] Y. Ganjisaffar. Crawler4j v. 3.5. URL http://code.google.com/p/crawler4j/. [cit. 2014-03-26].

[44] Y. Hoppe and G. Karpouzas. Ex-Crawler v 0.1.6. URL http://ex-crawler.sourceforge.net. [cit. 2014-03-26].

[45] S. DeRose J. Clark. XML Path Language (XPath)Version 1.0, November 1999. URL http://www.w3.org/TR/xpath/. [cit. 2014-07-26].

[46] J. Stárka M. Svoboda J. Schejbal, J. Sochna and I. Mlýnková. Analyzer - A Tool for Batch File Analysis 1.0. URL http://analyzer.kenai.com/. [cit. 2013-11-08].

[47] P. Jack and N. Levitt. Heritrix v. 3.2.0. URL http://crawler.archive.org/. [cit. 2014-03-26].

[48] Jan Sochna Jiří Schejbal Irena Mlýnková Jakub Stárka, Martin Svoboda and David Bednárek. Analyzer - A Complex System for Data Analysis. *The Computer Journal*, 55(5):590–615, October 2011. URL http://dx.doi.org/10.1093/comjnl/bxr103. [cit. 2013-11-08].

[49] Irena Mlýnková Jiří Schejbal, Jakub Stárka. XQConverter: A System for XML Query Analysis. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6059805&tag=1. [cit. 2013-11-08].

[50] K. Pokorny J. Mlynkova, I. Toman. Statistical Analysis of Real XML Data Collections. Technical report, Charles University, Prague, Czech Republic, June 2006. URL http://www.ksi.mff.cuni.cz/~holubova/doc/tr2006-5.pdf. [cit. 2013-11-08].

[51] P. Mockapetris. RFC 1035 DOMAIN NAMES – IMPLEMENTATION AND SPECIFICATION, November 1987. URL http://www.ietf.org/rfc/rfc1035.txt. [cit. 2014-07-26].

[52] N. Borenstein N. Freed. RFC 2045 Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, November 1996. URL http://www.ietf.org/rfc/rfc2045.txt. [cit. 2014-07-26].

[53] N. Borenstein N. Freed. RFC 2046 Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, November 1996. URL http://www.ietf.org/rfc/rfc2046.txt. [cit. 2014-07-26].

[54] R. Platt. Arachnid Web Spider Framework v. 0.4. URL http://arachnid.sourceforge.net/. [cit. 2014-03-26].

[55] J. Mogul H. Frystyk L. Masinter P. Leach R. Fielding, J. Gettys and T. Berners-Lee. RFC 2616 Hypertext Transfer Protocol – HTTP/1.1, June 1999. URL http://tools.ietf.org/html/rfc2616. [cit. 2014-07-26].

[56] M. F. Fernández D. Florescu J. Robie S. Boag, D. Chamberlin and J. Siméon. XQuery 1.0: An XML Query Language, December 2010. URL http://www.w3.org/TR/xquery/. [cit. 2014-03-10].

[57] R. Hinden S. Deering. RFC 2460 Internet Protocol, Version 6 (IPv6) Specification, December 1998. URL http://tools.ietf.org/html/rfc2460. [cit. 2014-07-26].

[58] C. M. Sperberg-McQueen S. Gao and H. S. Thompson. XML Schema Definition Language (XSD) 1.1 Part 1: Structures, April 2012. URL http://www.w3.org/TR/xmlschema11-1/. [cit. 2014-07-26].

[59] J. Schejbal. *A System for Analysis of Collections of XML Queries*. Master thesis, Charles University in Prague, Czech Republic, May 2010. URL http://www.ksi.mff.cuni.cz/~bednarek/dp/Schejbal.pdf. [cit. 2013-11-08].

[60] J. Sochna. *Collecting XML Data and Meta-Data from the Internet*. Master thesis, Charles University in Prague, Czech Republic, May 2010. URL http://www.ksi.mff.cuni.cz/~bednarek/dp/Sochna.pdf. [cit. 2013-11-08] in Czech.

[61] L. Masinter T. Berners-Lee and M. McCahill. RFC 1738 Uniform Resource Locators (URL), December 1994. URL http://www.ietf.org/rfc/rfc1738.txt. [cit. 2014-07-26].

[62] C. M. Sperberg-McQueen E. Maler T. Bray, J. Paoli and F. Yergeau. Extensible Markup Language (XML) 1.0, August 2006. URL http://www.w3.org/TR/2006/REC-xml-20060816/. [cit. 2013-11-08].

[63] G. Bracha T. Lindholm, F. Yellin and A. Buckley. The Java® Virtual Machine Specification, Ferbruary 2013. URL http://docs.oracle.com/javase/specs/jvms/se7/html/index.html. [cit. 2014-07-26].

# List of Tables

# List of Figures

# Attachments

## CD Content

The enclosed CD contains:

**crawlers-sources**
: Here are some sources of the analyzed crawlers from this thesis.

**data**
: Contains gathered, converted, created data for thesis.

- Directory ‘1-seeds‘ contains data for seed file.
- Directory ‘2-crawl_data‘ contains data gained during crawling, cleaned data (withoud duplicities) and DNS logs.
- Directory ‘3-xml_tree‘ contains data before and after conversion to XML tree representation. Also cleaned data are here.
- Directory ‘4-xpath_hits‘ contains results – outputs of XPath evaluation. Table ‘Thesis XPath Evaluation.xlsx‘ contains XPath expressions with descriptions.
- File ‘Thesis data tables.xlsx‘ contains tables for seed requiring, crawl sessions and conversion of data.

**sources**
: Contains sources of used programs, their binaries, scripts, etc.

- Directory ‘1-scrapper‘ contains NetBean’s project for scrapper, executable jar and scripts for running scrapper, and for putting results into one sorted file.
- Directory ‘2-crawler‘ contains NetBean’s project for crawler, executable jar, DoubleKiller program and scripts for running crawler on Windows or Unix.
- Directory ‘3-xqconverter‘ contains NetBean’s projects of XQAnalyzer and XQConverter, XQConverter jar executable with libraries and scripts for running it, editing some of input files, getting only valid downloaded XQuery Programs and counting types of errors from conversion.
- Directory ‘4-xpath‘ contains NetBean’s projects of count duplicities, executable jar, script for creating empty text files for results from XPath evaluation and scripts to count statistics.
- Directory ‘analyzer‘ contains Analyzer sources and documentation files.

**text**
: Contains sources, images and pdf of this thesis.

# XQuery grammar symbols

**AdditiveExpr** – This symbol represents subtraction or addition operation.
**AndExpr** – This symbol represents logical expression and. It is true only if both values are true.
**BaseURIDecl** – This symbol specifies the base URI property of the static context. The base URI property is used when resolving relative URIs within a module.
**BoundarySpaceDecl** – This symbol sets the boundary-space policy in the static context, overriding any implementation-defined default.
**CastableExpr** – This symbol tests whether a given value is castable into a given target type.
**CastExpr** – This symbol converts a value to a specific datatype.
**ComputedConstructor** – This symbol allows to create nodes alternatively.
**ConstructionDecl** – This symbol sets the construction mode in the static context, overriding any implementation-defined default.
**ContextItemExpr** – This symbol evaluates to the context item, which may be either a node or an atomic value.
**CopyNamespacesDecl** – This symbol sets the value of copy-namespaces mode in the static context, overriding any implementation-defined default. Copy-namespaces mode controls the namespace bindings that are assigned when an existing element node is copied by an element constructor or document constructor.
**DefaultCollationDecl** – This symbol sets the value of the default collation in the static context, overriding any implementation-defined default.
**DefaultNamespaceDecl** – This symbol can be used in a Prolog to facilitate the use of unprefixed QNames.
**DirectConstructor** – This symbol is a form of element constructor in which the name of the constructed element is a constant.
**EmptyOrderDecl** – This symbol sets the default order for empty sequences in the static context, overriding any implementation-defined default. This declaration controls the processing of empty sequences and NaN values as ordering keys in an order by clause in a FLWOR expression.
**ExtensionExpr** – This symbol consists of one or more pragmas, followed by an expression enclosed in curly braces.
**FLWORExpr** – This symbol consists of 4 optional clauses (‘for clause‘, ‘let clause‘, ‘where clause‘ and ‘order by clause‘) and 1 required clause (‘return clause‘). The result of the FLWOR expression is an ordered sequence containing the results of these evaluations, concatenated as if by the comma operator.
**ForClause** – This symbol generates an ordered sequence of tuples of bound variables, called the tuple stream.
**ForwardAxis** – This symbol represents one of these axis ‘child‘, ‘descendant‘, ‘attribute‘, ‘self‘, ‘descendant-or-self‘, ‘following-sibling‘, ‘following‘.
**FunctionCall** – This symbol consists of a QName followed by a parenthesized list of zero or more expressions, called arguments.
**FunctionDecl** – This symbol specifies whether a function is user-defined or external. User-defined function, the function declaration includes an expression called the function body that defines how the result of the function is computed

from its parameters. External functions are functions that are implemented outside the query environment.
**GeneralComp** – This symbol represents comparison using one of these symbols ‘=‘, ‘! =‘, ‘<‘, ‘<=‘, ‘>‘, ‘>=‘.
**IfExpr** – This symbol represents condition. If the effective boolean value of the test expression is true, the value of the then-expression is returned. If the effective boolean value of the test expression is false, the value of the else-expression is returned.
**InstanceofExpr** – This symbol returns true if the value of its first operand matches the SequenceType in its second operand.
**IntersectExceptExpr** – This symbol eliminates duplicate nodes from their result sequences based on node identity. The intersect operator takes two node sequences as operands and returns a sequence containing all the nodes that occur in both operands. The except operator takes two node sequences as operands and returns a sequence containing all the nodes that occur in the first operand but not in the second operand.
**LetClause** – This symbol generates an ordered sequence of tuples of bound variables, called the tuple stream.
**Literal** – This symbol represents different values as ‘xs:integer‘, ‘xs:decimal‘, ‘xs:double‘, ‘xs:untypedAtomic‘, ‘xs:string‘.
**ModuleDecl** – This symbol serves to identify a module as a library module. A module declaration begins with the keyword module and contains a namespace prefix and a URILiteral.
**ModuleImport** – This symbol imports the function declarations and variable declarations from one or more library modules into the function signatures and in-scope variables of the importing module.
**MultiplicativeExpr** – This symbol represents multiplication or division or modulo operation.
**NamespaceDecl** – This symbol declares a namespace prefix and associates it with a namespace URI, adding the (prefix, URI) pair to the set of statically known namespaces.
**NodeComp** – This symbol represents comparison using one of these symbols ‘is‘, ‘<<‘, ‘>>‘.
**OptionDecl** – This symbol serves as a particular option that will be recognized by some implementations and not by others. The syntax is designed so that option declarations can be successfully parsed by all implementations.
**OrderByClause** – This symbol is used to reorder the tuple stream.
**OrderedExpr** – This symbol sets the ordering mode in the static context to ordered for a certain region in a query.
**OrderingModeDecl** – This symbol sets the ordering mode in the static context, overriding any implementation-defined default.
**OrExpr** – This symbol represents logical expression or. It is true if one of values is true.
**PathExpr** – This symbol can be used to locate nodes within trees. A path expression consists of a series of one or more steps, separated by ‘/‘ or ‘//‘, and optionally beginning with ‘/‘ or ‘//‘.
**Prolog** – This symbol is a series of declarations and imports that define the processing environment for the module that contains the Prolog.

**QuantifiedExpr** – This symbol begins with a quantifier, which is the keyword ‘some‘ or ‘every‘, followed by one or more in-clauses that are used to bind variables, followed by the keyword ‘satisfies‘ and a test expression.
**QueryBody** – This symbol consists of an expression that defines the result of the query.
**RangeExpr** – This symbol can be used to construct a sequence of consecutive integers.
**ReverseAxis** – This symbol represents one of these axis ‘parent‘, ‘ancestor‘, ‘preceding-sibling‘, ‘preceding‘, ‘ancestor-or-self‘.
**SchemaImport** – This symbol imports the element declarations, attribute declarations, and type definitions from a schema into the in-scope schema definitions. For each user-defined atomic type in the schema, schema import also adds a corresponding constructor function.
**TreatExpr** – This symbol can be used to modify the static type of its operand. It does not change the dynamic type or value of its operand. Its purpose is to ensure that an expression has an expected dynamic type at evaluation time.
**TypeswitchExpr** – This symbol represent switch condition. It has an expression enclosed in parentheses, called the operand expression. This is the expression whose type is being tested. The remainder of the typeswitch expression consists of one or more case clauses and a default clause.
**UnaryExpr** – This symbol represents unary plus or unary minus.
**UnionExpr** – This symbol eliminates duplicate nodes from their result sequences based on node identity. It takes two node sequences as operands and return a sequence containing all the nodes that occur in either of the operands.
**UnorderedExpr** – This symbol sets the ordering mode in the static context to unordered for a certain region in a query.
**ValidateExpr** – This symbol can be used to validate a document node or an element node with respect to the in-scope schema definitions.
**ValueComp** – This symbol represents comparison using one of these symbols ‘eq‘, ‘ne‘, ‘lt‘, ‘le‘, ‘gt‘, ‘ge‘.
**VarDecl** – This symbol adds the static type of a variable to the in-scope variables, and may also add a value for the variable to the variable values.
**VarRef** – This symbol is a QName preceded by a $-sign.
**WhereClause** – This symbol filters the tuple stream, retaining some tuples and discarding others.