Charles University in Prague
Faculty of Mathematics and Physics

**MASTER'S THESIS**

Bc. Viktor Mašíček

# XSLT Benchmarking

Department of Software Engineering

Supervisor: RNDr. Irena Mlýnková, Ph.D.
Study Program: Informatics, ISS

2012

I would like to thank to my supervisor RNDr. Irena Mlýnková, Ph.D. for her helpfulness and time, which she willingly gave me. As well, I would like to thank for her suggestions and tips, which were valuable clue for me for realization of this thesis.

**Název práce:** XSLT Benchmarking
**Autor:** Bc. Viktor Mašíček
**Katedra:** Katedra softwarového inženýrství
**Vedoucí diplomové práce:** RNDr. Irena Mlýnková, Ph.D.
**E-mail vedoucího:** mlynkova@ksi.mff.cuni.cz

**Abstrakt:** Hlavním cílem této práce bylo vytvoření XSLT Benchmarku, tedy srovnání dostupných XSLT procesorů. Nejprve jsme si stanovili hlavní kritéria XSLT procesorů (*cena*, *korektnost*, *rychlost*, *využití paměti*, *podpora*, *OS* a *UX*), která budeme diskutovat. Poté jsme shrnuli existující XSLT procesory, popsali jednotlivé typy procesorů (*program*, *knihovna* a *prohlížeč*) a ohodnotili kritéria *cena*, *podpora*, *OS* a *UX*. Abychom mohli ohodnotit kritéria *korektnost*, *rychlost* a *využití paměti*, museli jsme si vytvořit vhodné testovací prostředí. Vytvořený program *XSLT Benchmarking* je jedním z hlavních přínosů této práce. Součástí programu jsou také připravené testy. Při jejich vytváření, jsme vycházeli z 5 787 stažených XSLT souborů. Testy byly vytvořeny na základě analýzy těchto souborů. Zaprvé jsme zkoumali obecné vlastnosti XSLT souborů (použité elementy, použité XSLT verze, formát výstupu atd.). Zadruhé jsme zkoumali zaměření jejich použití. Dalším velkým přínosem této práce je shrnutí výsledků testů z různých pohledů. V tomto shrnutí jsme okomentovali další kritéria a vlastnosti testovaných procesorů. Na závěr jsme shrnuli možná rozšíření našeho programu i naší analýzy.

**Klíčová slova:** XSLT, benchmarkování, analyzování, generování

**Title:** XSLT Benchmarking
**Author:** Bc. Viktor Mašíček
**Department:** Department of Software Engineering
**Supervisor:** RNDr. Irena Mlýnková, Ph.D.
**Supervisor's e-mail address:** mlynkova@ksi.mff.cuni.cz

**Abstract:** The main goal of this work was to create an XSLT Benchmark, to compare available XSLT processors. At first, we determined main criteria of XSLT processors (*price*, *correctness*, *speed*, *memory usage*, *support*, *OS* and *UX*), which we discussed. Next, we summarized existing XSLT processors, described individual types of processors (*program*, *library* and *browser*) and measured criteria *price*, *support*, *OS* and *UX*. We had to create appropriate test environment to measure criteria *correctness*, *speed* and *memory usage*. Created program *XSLT Benchmarking* is one of the main benefits of this work. The program also includes tests. We created tests based on 5 787 downloaded XSLT files. Tests were created based on the analysis of these files. Firstly, we researched common features of XSLT files (used elements, used XSLT versions, output format etc.). Secondly, we researched focuses of their usages. Next big benefit of this work is a summary of results of tests from different views. We discussed other criteria and features of tested processors. Finally, we summarized possible extensions of our program and also of our analysis.

**Keywords:** XSLT, benchmarking, analyzing, generating

# Contents

# Chapter 1

# Introduction

In thich chapter, we describe the term *benchmark*. Next, we summarize our motivation for creating XSLT Benchmarking and we determine the goal of this thesis. Finally, we summarize contents of individual chapters.

## 1.1 Benchmarking

The benchmarking is a process used in many areas. It is widely used in business world, hardware world or software world. However, all of them are based on the same principles. Their principles lie in comparison of competing products and assessing which is the optimal solution for set criteria.

The criteria depend on consumer's needs and requirements and usage of the product. For example, we have to test balloons for parents or children. All of them use a balloon for playing (typically together). However, some criteria are different for them. *Price* and *endurance* are the main criteria for parents, whereas *color* and *endurance* are the main criteria for children. As we can see, some criteria depend on consumer, so they are different, others depend on usage, so they are identical.

Firstly we have to define our consumers and their criteria so that we can compare them afterwards.

## 1.2 Motivation

XML technologies are rising in importance and thus it is needed to be interested in them. Many products exist in this field and many are being developed, so it is beneficial to benchmark them.

As stated in [1], XML technologies are very comprehensive now and they grow very fast, so it is impossible to create a universal and functional benchmark. It is necessary to focus on particular areas. Many areas are not discussed in any benchmarking or their benchmarking is obsolescence.

XSLT [2] is one such area. Some benchmarking exists for XSLT, but it is outdated. The absence of parameterization of these benchmarks is a big reason of their outdated. Hence, we try to make actual benchmark with enough parameters.

## 1.3 The Goal of the Work

The main goal of this work will be to create an XSLT Benchmark, to compare available XSLT processors. Other goals will be related to this main goal. We will have to create flexible test environment including enough tests. It will be required that tests will be based on real XSLT files. Thus, next goals will be to collect enough real-world XSLT files and analyze them. Of course, we will have to sufficiently discuss results of tests at the end of the work. We will have to create comparison of tested XSLT processors. At the beginning of the work, summary of existing XSLT benchmarks will be inspiring for us.

## 1.4 Contents Overview

Chapter 1 determines goals of this work and the term benchmark. In addition, it describes our motivation for creating XSLT Benchmarking.

Chapter 2 describes a brief overview of XSLT technology used in this thesis.

Chapter 3 defines main criteria for comparing XSLT processors in next analysis.

Chapter 4 provides an overview of existing XSLT processors and discusses of some of their characteristics.

Chapter 5 describes existing XSLT benchmarking projects.

Chapter 6 describes the approaches of our testing of XSLT processors.

Chapter 7 describes approaches of collecting real XSLT templates.

Chapter 8 analyzes collected XSLT templates. The first part describes analysing of common features of XSLT templates. The second part describes analysing of categories of XSLT templates, thus focuses on their usages.

Chapter 9 descibes our created program XSLT Benchmarking for comparing of some criteria of XSLT processors.

Chapter 10 makes an overview of our tests used in the program.

Chapter 11 discusses the results of our tests. There are discussions from various points of view.

Chapter 12 summarizes the whole thesis.

Chapter 13 summarizes all possible extensions of our analysis, our program and other parts of our thesis.

# Chapter 2

# Summary of XSLT Technology

XSLT is in the center of interest of this thesis. XSLT technology comes from the family of XML technologies [3]. A detailed description of XML technologies can be found in [4].

An XSLT document is an XML-based document intended for transformation of XML documents. The output is in a plain text format. A typical output is another XML document or an HTML [5] document. An XSLT document has a typical extension *xslt* or *xsl*.

The transformations are made by an XSLT processor (e.g. Saxon [6] or xsltproc [7]). An XSLT document contains “`xsl:template`” elements for transformation of input XML documents and the processor generates the output according to the content of these templates.

Templates have two variants of usage. The first one is with attribute “`match`” that contains an XPath [8] expression. The expression defines a part of data from the input XML document to be transformed by the content of the template. Thus, the using of attribute “`match`” can be understood as non-procedural approache. The second variant is with attribute “`name`”. This variant is explicitly called from another template. Data for transformation are passed by parameters into the template. Thus, templates with attribute “`name`” can be understood as functions, thus it can be understood as procedural approache. Note that all templates are explicitly defined. In addition, XSLT language contains implicit templates to be applied in case there is no explicit template, so even an empty XSLT document typically returns a non-empty output.

XSLT also supports constructs like *if*, *for*, *variable* or *parameter* and it is possible to process multiple input documents.

There are currently two versions of XSLT, XSLT 1.0 [2] and XSLT 2.0 [9]. XPath 1.0 [8] is used in XSLT 1.0 and XPath 2.0 [10] in XSLT 2.0. There are many extensions in the second version: e.g. the output can be generated into multiple documents, it is possible to define own functions that can be used in XPath expressions, there is support for grouping nodes in 'for' and many others.

Here are some examples for better notion. The first one is an input XML document containing lists of films. There are two types of lists of films (dvd and avi), in Example 2.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<films>
   <dvd>
      <item category="fantasy">Duna</item>
      <item category="action">Blade</item>
      <item category="fantasy">Harry Potter 1</item>
      <item category="drama">Fountain</item>
      <item category="action">RED</item>
      <item category="fantasy">Narnia 2</item>
      <item category="sci-fi">Twelve Monkeys</item>
      <item category="mute">Rosen på Tistelön</item>
   </dvd>
   <avi>
      <item category="action">Blade</item>
      <item category="comedy">Ace Ventura</item>
      <item category="comedy">Mask</item>
      <item category="drama">Trainspotting</item>
      <item category="action">Die Hard</item>
   </avi>
</films>
```

Example 2.1: **films.xml**: An input XML document containing two types of lists of films (dvd and avi).

The XSLT script (listed in Example 2.2) transforms an input XML document into an HTML document. It makes a list of film categories with list of their films (see Example 2.3). Categories are grouped from all types of lists. There are used constructs from XSLT 1.0 such as “`template`” with “`match`” and “`name`” attributes, “`value-of`”. Also, there are used constructs from XSLT 2.0 “`for-each-group`” and “`character-map`”.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version='2.0'
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>

   <xsl:output method="html"
      use-character-maps="specChar"/>

   <xsl:character-map name="specChar">
      <xsl:output-character character="å"
         string="&amp;aring;" />
      <xsl:output-character character="ö"
         string="&amp;ouml;" />
   </xsl:character-map>
   <xsl:template match="/">
      <html>
         <xsl:call-template name="head" />
         <body>
            <xsl:for-each-group select="//item"
               group-by="@category">
               <h1>
                  <xsl:value-of select="@category" />
               </h1>
               <p>
                  <xsl:value-of
                     select="current-group()"
                     separator=", " />
               </p>
            </xsl:for-each-group>
         </body>
      </html>
   </xsl:template>
   <xsl:template name="head">
      <head>
         <title>Films - category</title>
      </head>
   </xsl:template>
</xsl:stylesheet>
```

Example 2.2: **films.xslt**: The XSLT script transforms the input XML document into an HTML document.

This is the output HTML document, in Example 2.3.

```html
<html>
   <head>
      <meta http-equiv="Content-Type"
         content="text/html; charset=UTF-8">
      <title>Films - category</title>
   </head>
   <body>
      <h1>fantasy</h1>
      <p>Duna, Harry Potter 1, Narnia 2</p>
      <h1>action</h1>
      <p>Blade, RED, Blade, Die Hard</p>
      <h1>drama</h1>
      <p>Fountain, Trainspotting</p>
      <h1>sci-fi</h1>
      <p>Twelve Monkeys</p>
      <h1>mute</h1>
      <p>Rosen p&aring; Tistel&ouml;n</p>
      <h1>comedy</h1>
      <p>Ace Ventura, Mask</p>
   </body>
</html>
```

Example 2.3: **films.html**: The output HTML document of the XSLT transformation.

A new norm XSLT 3.0 [11], is being created too. However, an analysis in this thesis (see Chapter 8) showed, thad XSLT 1.0 is used mainly and XSLT 2.0 a little. One of the reasons could be, that XSLT 3.0 is only a draft of expected final recomendation. Thus, we will focus mainly on XSLT 1.0 and partly on XSLT 2.0 in this thesis.

# Chapter 3

# XSLT Benchmarking

The puprpose of benchmarking was described in general in Chapter 1.1. In this chapter, we define our *consumers* and their *criteria* for comparing XSLT processors.

XSLT documents are XML-based, thus XML benchmarking can have some common features. So, we describe their specifics at first. Secondly, we define criteria of XSLT processors important for our consumers.

## 3.1 Overview of XML Benchmarking

As we have already mentioned, XSLT is XML-based, therefore many XML benchmarking projects and frameworks exist. So, we can learn from XML benchmarking, theses or reports about XML benchmarking. Report [1] describes different XML benchmarks and divides them into categories by the type of testing. There are two main classifications. The first one distinguishes XML benchmarks by data origin. We have *real-world* data and *synthetic* data. The second classification distinguishes XML benchmarks by data parameterization. We have *fixed* data and data that are *user-configurable*.

Next, there are some described problems relevant for XSLT. Some benchmarks use simple data and do not cover all constructs. Typically, it is associated with real-world data. On the other hand, synthetic data are typically user-configurable. Although they cover all constructs, they are very complicated. Excessive specialization is the next problem. Very complex benchmark may be confused, but, on the other hand, some benchmarks are too specialized and easy.

We made some conclusions from report [1]. We will use both origins of data, real-world and synthetic (or combination of them). It allows us to cover a big part of constructs and makes data user-configurable and extensible. Next, we try to cover only aspects of tested XSLT processors that we deem important.

## 3.2 Criteria of XSLT Processors

XSLT processing belongs to computing and software areas and XSLT processors are typically used by programmers, administrators and IT professionals. They are our consumers. However, using of XSLT processors can be very diverse. IT professionals have typically the same criteria, but they have different priorities of criteria. For example, the speed of a processor is not the main criterion for infrequent using. Conversely, consumer can be a beginner and thus a good documentation is important for him. On the other hand, possibility of running a processor from a command line as a script can be the main criterion for other consumers (e.g. administrators).

These are the main criteria for all consumers together: *price*, *correctness*, *speed*, *memory usage*, *support* (timeliness, liveliness, availability, documentation), *OS*[1] and *UX*[2] (simplicity of installation, user friendliness, running in scripts). As we discussed in Chapter 3.1, we want to cover the main aspects of processors. So, we include all listed criteria into our final benchmark of XSLT processors.

We compare each criterion separately. Then, consumers can select relevant criteria for them. Price, support and layout can be described and compared easily. A big part of this thesis discuses correctness, speed and memory usage of processors. We have to create a special tool for their comparison.

[1]Operating System (OS)
[2]User Experience (UX)

# Chapter 4

# XSLT Processors

In this chapter, we describe types of XSLT processors. Next, we make an overview of existing XSLT processors and discuss some of their criteria (price, support, OS and UX), features and important aspects.

XSLT processors exist as separate programs (either programs allowing only XSLT transformations or programs allowing XSLT transformations and other functions), downloadable libraries for programming and scripting languages (C++, Java, PHP etc.) or components of web browsers. These types of processors will be called *program*, *library* and *browser*, respectively.

Note that some processors can be classified into several types. For example, XSLT processor "Some XSLT Processor" can be downloaded as library and this library can be used in a web browser. Then, "Some XSLT Processor" is classified into types *library* and *browser*.

The list of XSLT processors with their types and criteria price, support and layout is provided in Table 4.1. The processors we used for testing are listed at the beginning of the table. These processors are described in next chapters. In addition, there are not tested processors at the end of the table. These processors are not tested because they would be too complicated or they are small or outdated processors. These processors are listed for fuller view.

| **XSLT Processor** | **Type** | **Price** | **Support** | **UX** | **OS** |
|---|---|---|---|---|---|
| **Tested processors** | | | | | |
| Saxon [6] | program library | free money | full | good | all |
| Xalan [12] | program library | free | almost full | partial | all |
| XT [13] | program library | free | little | partial | all |
| libxslt [14] | program library | free | almost full | almost good | all |
| MSXML [15] | browser library | money | full | good | windows |
| Sablotron [16] | program | free | no | almost good | windows linux |
| **Not tested processors** | | | | | |
| 4Suite [17] | library | free | half | unknown | all |
| TransforMiiX [18] | browser | free | almost full | good | all |
| xslt.js [19] | library | free | half | partial | all |
| ajaxslt [20] | library | free | little | unknown | all |
| Unicorn XSLT [21] | program | free | half | good | windows |
| AltovaXML [22] | program library | money | full | almost good | windows |
| XmlPrime [23] | program library | money | almost full | good | windows |

Table 4.1: **List of XSLT processors**: The list of XSLT processors with some of their properties.

## 4.1 Saxon

Saxon [6] is one of the most famous XSLT processors. It is implemented in Java, which enhances its portability. It is available as a library in Java and Java jar file for easy running directly from the command line. It is available free of charge up to version 6.5.5. Currently, it is available in free version HE and commercial versions PE and EE. Saxon comes with high-quality user and API documentation. It is still continuously updated. Basic usage of the XSLT transformation is simple with a few command line switches, so it can be used in batch scripts. The only disadvantage is less clear command line help for HE version.

## 4.2 Xalan

Xalan [12] is a fairly widespread XSLT processor. It is available as a Java and C++ library and Java jar file to be run from the command line. So, it is available practically on all operating systems. However, installation is sometimes awkward. When using Java jar file version, we need to download multiple files and choose the right one to use. We need to download a package containing a large number of exe files and select the correct file for the command line variant on Windows. The installation is without problems via command line variant on Linux if we use a packaging system. Basic use is simple with a few command line switches. Xalan is easily downloadable and the available documentation is good. However, its latest version is from the year 2007, so it has not been updated recently. Nevertheless, the project seems to be living and perhaps more recent version will be created.

## 4.3 XT

XT [13] is an XSLT processor also implemented in Java. Therefore, it can be used on almost all operating systems. It is freely downloadable in the form of a Java library and Java jar files (so it can be run from the command line). However, the last version is from 2005 and it is a dead project, which is a big disadvantage. Another disadvantage is the necessity to download external XP [24] (XML Parser) and SAX [25] for its full functioning. API documentation is good, but the help for the first run is insufficient. The advantage is the ability to run the XT processor as a script.

## 4.4 Libxslt

Libxslt [14] is one of the most famous libraries for XSLT transformation. It is implemented in C and freely downloadable in the form of a C library, a PHP library (and others) or a command line program called xsltproc [7]. The project is currently living and developers are working on it. However, the last version is from 2009. The documentation is very good for both the library and the xsltproc program. The library is available for most operating systems. The main disadvantage is however the uncomfortable installation of xsltproc for Windows. It is necessary to download additional libraries (libxml2 [26], iconv [27], zlib [28]). Subsequent usage of the command line running is easy.

## 4.5 MSXML

MSXML [15] is a library for working with XML, XSLT etc. It is a part of Windows [29]. It is commercial and only available for Windows, which is its major disadvantage. Internet Explorer [30] uses it and it can be used in standalone JavaScipt. No installation is required, since it is natively included in Windows OS. It has a good documentation. If we want to use it as a command line program, we need to write own script and use the program cscript [31] to run it. This is not a disadvantage because it is not primarily intended for use from the command line.

## 4.6 Sablotron

Sablotron [16] is an XSLT processor written in C++. It is available as a command line program for Linux (installable as a package) and Windows (downloadable as a zip file). It should be available as a library for C++, Python etc. However, we found only the binary version. Unfortunately, the project is already dead. The latest version is from 2006. We found the documentation only for the Python version. Other versions probably are not available. If we want to use Sablotron on Windows, we must download and install the Expat XML parser [32], which is a little disadvantage.

# Chapter 5

# Existing XSLT Benchmarking Projects

In this chapter we describe the existing XSLT Benchmarking projects. We compare their advantages and disadvantages for better determination of parameters for our benchmark. Criteria defined in Chapter 3 will be discussed too.

## 5.1 XSLTMark

XSLTMark [33] is one of the best known XSLT benchmarks. Many other benchmarks are based on it (e.g. Caucho [34] or David Parshley [35]). Unfortunately, it is outdated. The published results are from 2001, it is no longer maintained and it is not downloadable now. Nevertheless, XSLTMark is a famous XSLT benchmark.

It contains 40 synthetic tests and it has good metrics for measurement of transformation speed. It uses kilobytes-per-second value for each test, where kilobytes are the average of input and output document size. The result for one XSLT processor is the sum of kilobytes-per-second value for all tests. Note, that authors discussed the metrics based on nodes-per-second, but they did not use it. The real time (instead of CPU[1] time) is measured during tests. It is good idea, because real time is important for the major part of consuments. Each test has to be run multiple times for measurability of real time.

[1]Central Processing Unit (CPU)

Result verification is achieved by normalizing the output. Normalizing means sorting attributes alphabetically, removing insignificant white spaces etc. It allows us to compare the outputs with the expected normalized result.

XSLTMark rates only speed and correctness. Unfortunately, it does not cover other criteria, such as documentation and manageability, that can be important for some consumers. It is possible to upgrade XSLTMark by new tests and to add new tested processors. However, upgrades are no longer possible, because it is not downloadable at this moment. On the other hand, we can take inspiration by some used processes like measurement of transformation speed, using real time for measuring and normalizing the output.

## 5.2 Hello World

Hello World benchmarks [36] contain two XSLT benchmarks. Both benchmarks contain one test. They do not aim to be full benchmarks. They aim to test different XSLT processors run under the Apache web server [37]. They want to detect the major faults of the tested processors.

The published results are from April 2003. Benchmarks are downloadable now, but they have not been updated since publication of results. Only speed of transformation is tested and other aspects are not taken into account. So, these benchmarks probably do not contain anything for inspiration. However, they are one of the few benchmarks focused on XSLT.

## 5.3 XSLT is Way Faster Using Java 5

This benchmark [38] is focused on a different area than ours. However, it is one of the few benchmarks focused on XSLT, thus we mention it.

It was designed to compare speed of Java Virtual Machines[2] (JVMs). The test runs 1 000 times parsing a small XML document through a small XSLT template. The used XSLT processor is irrelevant, but the same XSLT processor has to be used in all JVMs. The published results are from February 2005 and the project is no longer maintained.

[2]Java Virtual Machine (JVM) is a set of computer programs and data structures that uses the virtual machine to run other computer programs and scripts created in programming language Java.

The question is whether this simple test is relevant. We do not know what would be the results of more complex tests. However, this benchmark could be an inspiration for us. We should compare Java XSLT processors under different JVMs. On these bases, we can assess JVMs and combinations of JVMs with XSLT processors.

## 5.4 Summary

Currently, there are very few XSLT benchmarks, the existing ones are outdated and have many faults. Nevertheless, we can take inspiration from some of them – e.g. the measurement of speed and normalizing of output documents like in XSLTMark or the possibility of testing different Java Virtual Machines like in XSLT is Way Faster Using Java 5. Infact we will not test XSLT processors in different Java Virtual Machines in our tests. However, we will prepare our test environment enough flexible for it. For more information about test environment see Chapter 9.

# Chapter 6

# Structure of Testing

XSLT processors with some of their criteria (defined in Chapter 3.2) were described in Chapter 4. Still, we have to assess criteria *speed*, *correctness* and *memory usage*. However, testing of these criteria is complex.

We have to make a set of tests for testing of speed, correctness and memory storage and an environment allowing to test different types of XSLT processors. Repeated set up tests, extensibility of test sets and addition of new XSLT processors are required.

We should base the tests on XSLT documents used in real applications for real applicable of the results. The first step is to collect enough XSLT documents (see Chapter 7).

Firstly, we will determine frequency of all elements from the downloaded data. The created test will be similar to real-world documents. Likewise, we will determine additional features of the downloaded documents (depth of XML tree, version of XSLT, fan-out of elements etc.) for creating tests. Moreover, we will create a test for typical file which combines all most frequently occuring values of features together.

Secondly, we will try to detect the usage of downloaded documents. So, we will make list of categories with their specifics. It can be expected that different categories will have different results of analysis. Thus, we will make a diferent set of tests for each category. We will describe both parts of the analysis in Chapter 8.

After analyzing the data we need test environment. Thus, we will create a program that allows to add new configurable tests, testing of different XSLT processors and results reporting. The program will be able to run on different operating systems and to influence its by settings. This environment will be described in Chapter 9.

After creating test environment, we will create tests based on our analysis. Most tests will be crated configurable for future extension. The tests are described in Chapter 10.

Finally, we will run the tests to get results. Results will be many numbers in many tables. It will be necessary to analyze results and presented interesting findings. The findings are presented separately from different views in Chapter 11.

# Chapter 7

# Collecting of XSLT Documents

An important part of XSLT benchmarking is analysis of real-world XSLT documents for real applicable of the results of tests. Thus, we have to collect a representative set of them. All scripts used for collecting data are saved on the attached CD (see Appendix A.1).

The crawler[1] Wget [39] was used for downloading the analyzed data from the Internet. We used various methods for searching addresses for downloading the data and we downloaded 19 650 XSLT files. The downloading methods are described in Chapter 7.1. Note, that we will denote these data as *dirty*. It means, there are some duplicities of documents in data or there are some non-XSLT documents.

We had to merge all the data downloaded by all methods, correct their contents, remove non-XSLT files, duplicities and similarities. Finally, we collected 5 787 documents for analysis after such *cleaning*. Merging and cleaning of data are described in detail in Chapter 7.2.

The success[2] of downloading was about 30 %. It is a good success due to the nature of the Internet which includes many duplicates and incorrect data in all areas.

[1]A *crawler* is program for automatic downloading of web sites from the Internet. An important feature is automatic following of links on the sites.

[2]A *success* is comparing the amount of downloaded dirty data and data after cleaning.

## 7.1 Downloading Methods

At first, we used the Google Search [40] with requests “filetype:xslt” and “filetype:xsl” (via [41]). The contents of the results were acquired by console web browser W3M [42]. Next, from the downloaded contents all found addresses were selected. The list of selected addresses is saved on the attached CD (see Appendix A.1). The addresses were used for two cases. Firstly, we downloaded the documents themselves. Secondly, we made seeds[3] from the addresses and the data were downloaded recursively. The seeds were created, because an analysis showed that Google Search did not find all XSLT documents and other XSLT documents were found manually on some sites. So, we tried to go through whole sites.

Note, that some addresses were deleted from the list of addresses searched on Google. In particular, these were addresses involving string “/fmi/”. A research showed, that these sites are engines for searching which include many links. Most of the links on these sites have an extension “xsl”, but the content of the files is in HTML. To be precise, about 100 addresses included “/fmi/”, hence their usage as seeds would take too much time without any benefit.

A similar method was used for collecting addresses from the Google Code [43] with requests “label:xslt” and “label:xsl”. The content was acquired by W3M and addresses were selected again. The list of selected addresses is saved on the attached CD too. The addresses link to project sites, that have a format “`http://code.google.com/p/[project name]/`” which include SVN[4] repositories of projects that include many XSLT documents. Unfortunately, the files are shown in HTML layout. On the other hand, for all collected addresses with described format exist addresses with a format “`http://[project name].googlecode.com/svn/`”. These addresses link to SVN of projects without HTML layout. So we created new addresses from original ones (collected from the Google Code) and used them as seeds for recursive downloading.

Next, we searched some addresses manually and used them as seeds for the recursive download, too. We collected these addresses during reserching

[3]A *seed* is a web site for start searching by crawler. We have to use restrictions for downloading, otherwise too many uninteresting sites are visited by following this links.

[4]The Subversion (SVN) is a software versioning and a revision control system. It allows us to merge differences, to access old verions and to share of source code, documentation, pictures etc.

for this thesis. Only addresses not found by Google were included into the list. The list of these addresses is saved on the attached CD too.

Finally, we selected addresses from a log file of downloading that include the string “xslt?” or “xsl?”. These files were not downloaded. We used restriction on file names by wildcards “*?.xsl” and “*?.xslt” in the crawler, thus links on XSLT files with parameters after question mark were skipped. So, we made a list of these files without parameters after question mark and downloaded them non-recursively. The list of these addresses is saved on the attached CD too. Note, that we could make better restriction, that covers files with parameters after question mark. Unfortunately, we noticed the mistake after time-consuming downloading. Thus, it was faster to download not downloaded files non-recursively instead of repeating all downloads.

There were also lines containing links to non-XSLT files in a log file, but links containing a link to an XSLT file as such parameter. Typically a parameter named “xslfile”. Links from parameters were downloaded non-recursively, too.

The numbers of addresses or seeds, downloaded documents and efficiency of downloading are listed in Table 7.1 for each method.

| **downloading method** | **# addr.** | **# files** | **effic.** |
|---|---:|---:|---:|
| manual | 10 | 227 | 22.70 |
| search | 1 437 | 1 339 | 0.93 |
| seeds (Google Search) | 1 793 | 9 881 | 5.51 |
| seeds (Google Code) | 476 | 7 872 | 16.53 |
| nonrecursive | 391 | 331 | 0.85 |
| **sum** | 4 107 | **19 650** | 4.78 |

Table 7.1: **Downloaded dirty data**: Numbers of downloaded dirty files for each method of downloading. There are also numbers of addresses or seeds used for collecting dirty data for each method and efficiency of each method (files/# addr.).

## 7.2 Merging and Cleaning Data

Firstly, we had to merge all the data downloaded by all methods.

Next, we had to detect incorrect files and repair or delete these files. We repaired about 100 files manually and deleted about 700 files.

Last but not least, we had to detect duplicates and similarities in the data. We deleted 4 733 files detected as duplicates and any files detected as similar.

Generally, we deleted about 5 500 files together. Finally, we have 5 787 XSLT files for the analysis after all the cleaning. The comparison of data before and after merging and cleaning is in Table 7.2.

| **data type** | **# files** |
|---|---:|
| dirty | 19 650 |
| clean | 5 787 |
| **efficiency** | **0.29** |

| **data type** | **# files** |
|---|---:|
| merged | 10 512 |
| clean | 5 787 |
| **efficiency** | **0.55** |

Table 7.2: **Clean data**: Numbers of downloaded files and numbers of files after merging and cleaning. There is also the efficiency of cleaning ("clean data/dirty data" and "clean data/merged data").

### 7.2.1 Merging

The dirty data was saved separately for each method. Then, we copied them together. However, some files can be downloaded several times by different methods. Each method of downloading can download file a separately. So, we had 10 512 XSLT files after merging all files.

### 7.2.2 Incorrect files

Unfortunately, some downloaded files had a correct extension (xslt or xsl), but the content of files was not an XSLT stylesheet (typically it was in HTML). So, we had to detect correct and incorrect XSLT files. The program xmllint [44] was used for testing validity of data.

We had three types of incorrect files in dirty data. First, there were files which included HTML sites that shown content of XSLT document plus another features of software for versioning (e.g. SVN or CSV). Then we had to select only the source of XSLT documents. Secondly, there were files which included pure HTML data. And, finally, there were XSLT files with small mistakes.

The files which contained XSLT with mistakes or XSLT included in HTML were detected by an easy test for inclusion of string "xsl:". Mistakes were repaired manually. For XSLT files included in HTML, clean XSLT documents were downloaded manually. Overall, there was about 100 files of such type.

Naturally, files which contained pure HTML were deleted. This was the case of about 700 documents.

### 7.2.3 Duplicates or Similarities

There were numerous duplicates or similar files downloaded into different directories in all the data. We can distinguish two cases.

The first one is when duplicate or similar files were downloaded from different domains. This case is desirable, because if the document is used in different domains, then it is typically used in a different way. It is of no importance, if duplicate or similar documents are created by two different authors independently or they use the same file (e.g. copied from the Internet). Anyway, the importance of a file is bigger in global, than the importance of its features in the analysis.

The second one is duplicates or similarities downloaded from the same domain. This case is undesirable, because they would distort the analysis. Typically, they are from different versions of the same project, different projects that used the same framework etc. On the other hand, duplicates or similarities with different names are desirable as well as duplicates or similarities within different domains. Typically, these files are used for different problems, so we want to enhance the importance of their features in the analysis.

Therefore, we had to detect duplicates or similarities within each domain. However, we compared only files with the same names. To detect duplicates and similarities we used the diff program [45]. For each domain we compare each two files with same names. The divergence value $DV$ was counted for each pairs of compared files.

$$DV = \frac{\text{number of different lines}}{\text{number of all lines in both files together}} \qquad DV \in [0;1] \qquad (7.1)$$

As we can see in Equation 7.1, the $DV$ corresponds to the percentage divergence of compared files. So, a pair of files with identical contents has $DV = 0$ and it was marked as duplicate.

For example, if we compare Example 7.1 and Example 7.2, then $DV = 0.2$ which means that their divergence is 20 %. Files have 20 lines together and the diff program [45] detected 4 different lines. There are two different lines in each example contrary to the second, thus $DV = 4/20 = 0.2$. Note, that the different lines are lines 6 and 7 in both examples.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <xsl:stylesheet version='1.0'
      xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
 3    <xsl:output method="html" indent="yes"
         encoding="UTF-8" />
 4    <xsl:template match="/">
 5       <html><body>
 6          <h1>Number of users</h1>
 7          <xsl:value-of select="count(//user)" />
 8       </body></html>
 9    </xsl:template>
10 </xsl:stylesheet>
```

Example 7.1: **dv_a.xslt**: The first file for demonstration of calculation of the divergence value.

```
 1 <?xml version="1.0" encoding="UTF-8"?>
 2 <xsl:stylesheet version='1.0'
      xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
 3    <xsl:output method="html" indent="yes"
         encoding="UTF-8" />
 4    <xsl:template match="/">
 5       <html><body>
 6          <h1>Number of orders</h1>
 7          <xsl:value-of select="count(//order)" />
 8       </body></html>
 9    </xsl:template>
10 </xsl:stylesheet>
```

Example 7.2: **dv_b.xslt**: The second file for demonstration of calculation of the divergence value.

A log file was generated for pairs of files with their divergence values and their differences for further checking and detection of similar files. Only pairs with divergence less than 10 % ($DV \in [0; 0.1)$) were logged, because pairs of files with divergence greater than 10 % cannot be considered as similar files. We checked the log file manually and only about 50 % of differences were significant, regardless the value of similarity. And the number of highly significant differences was too small. Therefore, we did not delete any files marked as similar.

Duplicate files ($DV = 0$) were deleted automatically. Of course, we deleted only one file from the pairs of files. In particular, 4 733 identical files were detected and deleted. Typically, only several duplicates files were detected and deleted in each domain, but in some domains a significant number of files was deleted. About 50 % of deleted files were from "`docbook.sourceforge.net`". The domains with significant number of deleted files are listed in Table 7.3.

| **SIGNIFICANT DOMAINS** | **# deleted files** |
|---|---:|
| docbook.sourceforge.net | 2 208 |
| scm.dspace.org | 737 |
| svn.openlaszlo.org | 688 |
| tecfa.unige.ch | 262 |
| www.w3.org | 161 |
| svn.repoze.org | 151 |
| quexml.svn.sourceforge.net | 96 |

| | |
|---|---:|
| number files from significant domains | 4 303 |
| number files from other domains | 430 |
| **sum** | **4 733** |

Table 7.3: **Deleted duplicates in domains**: There are numbers of deleted duplicate files per domain that have significant numbers of deleted files and the sum of all deleted files.

## 7.3 Summary

In total we downloaded 19 650 potencial XSLT files using different methods for getting address of files for downloading. The main tool for getting address was Google Search. Downloaded files had to be checked for duplicities and detecting non-XSLT files. Finally, after cleaning we colected 5 787 XSLT files for subsequent analysis.

# Chapter 8

# Analysis of XSLT Documents

After collection of documents, we analyze their content and identify categories of documents. The particular parts of the analysis are described in this chapter. All scripts used for analysis of data are saved on the attached CD (see Appendix A.2).

## 8.1 Scan

After collecting enough data, we scan all of them. Firstly, we describe the scanning. Secondly, we describe features watched during the scanning and discuss the results. Percentage results are counted against the total number of scanned files, which is 5 787. Finally, typical XSLT document is described.

Note that, in our analysis, we will not focus on XPath expressions. It is important to any part of XSLT transformations. Hovewer, it involves very complex issues, which should be discussed in a separate thesis. Nevertheless it can be built on our work and on our knowledge. Thus, we focus on the analysis of other parts.

It would be possible to analyze many aspects of downloaded XSLT files. We focused only on some of them for creating basic set of tests (see Chapter 10). It is possible to extend these tests. Our test environment is very flexible (see Chapter 9).

### 8.1.1 Process of Scanning

All downloaded files were scanned with a PHP [46] script called "`scan.php`" (see Appendix A.2). The PHP built-in class XMLReader, that reads documents by a SAX[1] method, was used for scanning. The SAX method was selected, because its memory requirements are not dependent on the size of the document it processes. The measured results were stored into files (separately for each feature) for further analysis. Some results will be represented by graphs for better idea. Exact values of all results are listed in tables in Appendix B.

### 8.1.2 Feature: Maximum Depth

Maximum depth of an XSLT document is one of its basic features. So, we measure it for each downloaded file and consider results in tests.

Graph 8.1 shows measured results. It is limited to files with maximum depth between 0 and 20 for better visualization, which represented 99 % of scanned files. In Table B.1 are all exact results.

### 8.1.3 Feature: Depth of Nesting

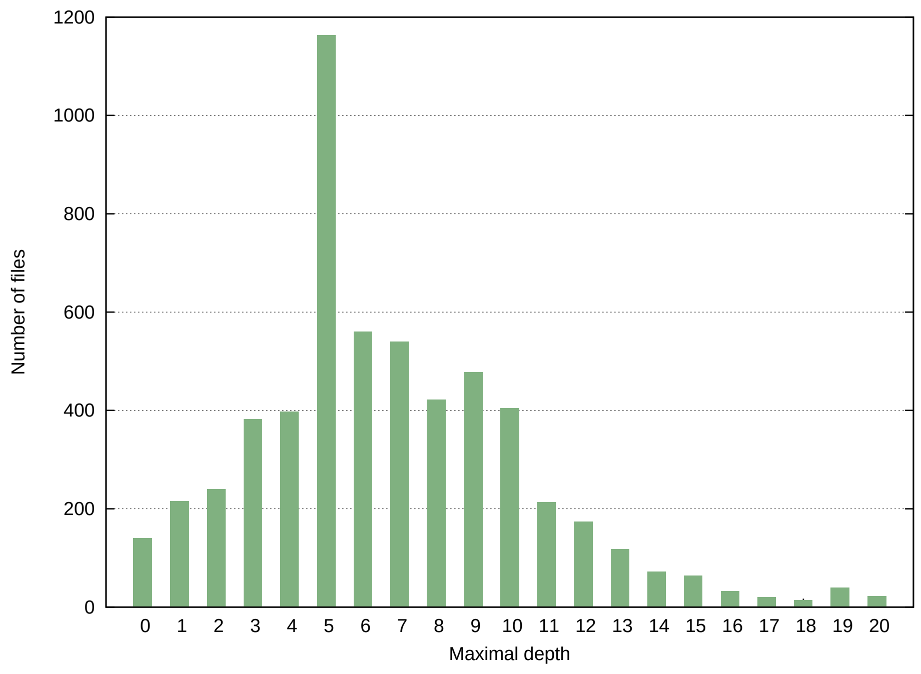
Constructs of XSLT language "`for-each`", "`choose`" and "`if`" (see Chapter 2) increase complexity of XSLT stylesheets (as similar constructs in other programming languages). Complexity increases more rapidly when these constructs are nested. Thus, we will measure the depth of nesting of selected elements. The depth of nesting will be measured separately for each construct (see Example 8.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version='2.0'
   xmlns:xsl='http://www.w3.org/1999/XSL/Transform'>
   <xsl:template match="/">
      <xsl:if test="...">
         <xsl:for-each select="...">
            <xsl:if test="...">
               <xsl:if test="...">
                  <xsl:for-each select="...">
                     ...
```

[1]SAX (Simple API for XML) [47] is an event-based sequential access parser API.

```
                  </xsl:for-each>
               </xsl:if>
            </xsl:if>
         </xsl:for-each>
      </xsl:if>
   </xsl:template>
</xsl:stylesheet>
```

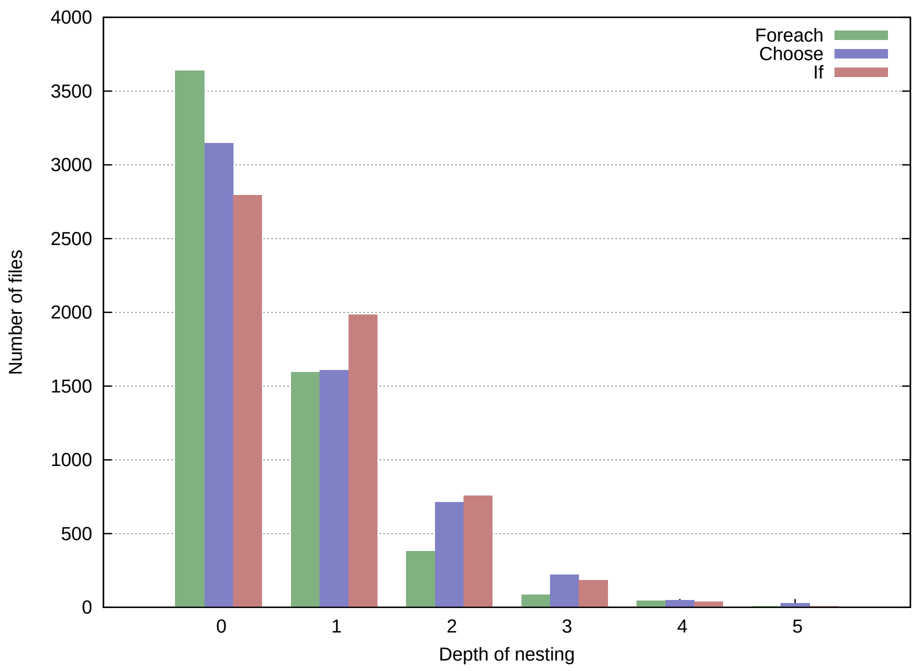
Example 8.1: **nesting.xslt**: Depth of nesting of construct “`if`” is 3 and depth of nesting of construct “`for-each`” is 2.



Graph 8.1: **Maximum depth of files**: The graph shows numbers of files (axis Y) with the respective maximum depth (axis X). The graph shows only files with maximum depth less than or equal to 20, that in sum represent 99 % scanned files.

From the measured results, we found out that more than 99 % of files have maximum depth of nesting of all selected elements less than or equal to 5. Results for depth of nesting between 0 and 5 are shown in Graph 8.2. All exact measured results are in Table B.2. Note, that 0 nesting means non occurence of a element in a file.

Note that, constructs "`choose`" and "`if`" could be analyzed together, because they are very similar constructs. Hovewer, it is not necessary, because of their very small nesting. Next, analyses would be focused on nesting of included XSLT files (by element "`include`" or "`import`"). Nevertheless, basic analyses is sufficient for our basic set of tests.



Graph 8.2: **Depth of nesting**: The graph shows numbers of files (axis Y) with the respective depths of nesting (axis X) separately for elements "`for-each`", "`choose`" and "`if`". The graph shows only files with depth of nesting less than or equal to 5, that in sum represent more than 99 % scanned files.

In Table 8.1, we can see percentage representation of selected groups of depth of nesting. As we can see a large percentage of files have no nesting (depth of nesting is 0 or 1).

### 8.1.4 Feature: XSLT Version

XSLT version used in an XSLT document is its important feature. It determines which constructs can be used in a stylesheet in accordance with W3C XSLT specification [2, 9] (see Chapter 2). Thus, we detect distribution of

| **Depth of nesting** | **Foreach** | **Choose** | **If** |
|---:|---:|---:|---:|
| 0 | 63 % | 54 % | 48 % |
| 1 | 27 % | 28 % | 34 % |
| **0-1** | **90 %** | **82 %** | **82 %** |
| 2-5 | 9 % | 17 % | 17 % |
| **0-5** | **99 %** | **99 %** | **99 %** |

Table 8.1: **Depth of nesting - percentage of significant groups**: In the table there are percentages representations of significant groups of depth of nesting for elements "`for-each`", "`choose`" and "`if`".

XSLT versions in scanned files. XSLT version can be simply found out from attribute "`version`" in root element ("`stylesheet`" or "`transform`").

About 85 % of scanned files have version "1.0". Given such a large representative only version "1.0" will be used in tests. However, about 7 % of scanned files have version "2.0", so we will create a few tests explicitly testing elements of version "2.0" used in scanned files (see Chapter 8.1.6). The results are shown in Table 8.2.

| **XSLT Version** | **Number of files** | **Percentage** |
|---:|---:|---:|
| 1.0 | 4 884 | 84.4 % |
| 1.1 | 79 | 1.4 % |
| 2.0 | 430 | 7.4 % |
| 3.0 | 1 | 0.0 % |
| unknown | 393 | 6.8 % |

Table 8.2: **XSLT version**: In the table there are detected XSLT versions of scanned files and their numbers and percentage.

An extended analysis of this feture would analyse of combination of XSLT version and occurence of particular XSLT elements. For example, we can detect not using of elements from XSLT 2.0 in XSLT file, that declare using of XSLT 2.0. Or even, we might detect using of elements from XSLT 2.0 in XSLT file, that do not declare using of XSLT 2.0.

### 8.1.5 Feature: Fan-out of Elements

Fan-out of an element in an XML document is a number of its sub-elements (child elements). Average fan-out of an document is an important feature, because it gives us an idea about structure and complexity of the document. Also, it is a potential aspect that could influence memory usage of XSLT transformation.

From the measured results, we found out that 98 % of files have the average fan-out 1, whereas other files have average fan-out 0 (see Table 8.3).

| Average fan-out | Number of files | Percentage |
|---:|---:|---:|
| 0 | 111 | 2 % |
| 1 | 5 658 | 98 % |

Table 8.3: **Average of fan-out**: In the table there are average of fan-out of scanned files and their numbers and percentage.

However, 86 % scanned files have maximum fan-out between 0 and 40 (see Graph 8.3). In Table B.3 there are all exact measured results.

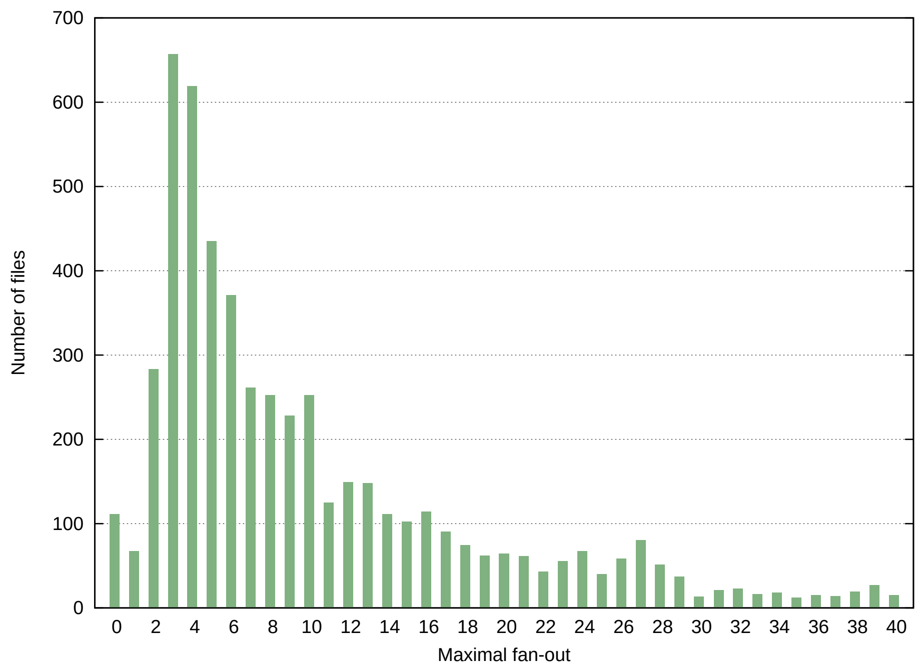
### 8.1.6 Feature: Element Numbers

W3C XSLT specification [2, 9] defines many elements in namespace "`xsl`". However, not all of them are used in practice. Thus next, we determine which elements are used in practice. Also, the usage of selected attributes of some elements were monitored too (e.g. attributes "`name`" and "`match`" for element "`template`").

An important finding was that 99 % elements (from namespace "`xsl`") were from XSLT version "1.0" (see Table 8.4).

Moreover, about 95 % of all elements are the same 16 elements. These elements with their rates are shown in Graph 8.4. We also noticed that 80 % elements "`template`" are used with attribute "`match`", which suggests the non-procedural approach.

List of all elements from namespace "`xsl`" found in scanned files is in Table B.4.



Graph 8.3: **Maximum fan-out**: The graph shows numbers of files (axis Y) with the respective maximum fan-out (axis X). The graph shows only files with maximum fan-out less than or equal to 40, that in sum represent 86 % of scanned files.

### 8.1.7 Feature: Size of files

The size of a file is a very important feature. Some XSLT processors may have a problem with big XSLT templates due to lack of the memory.

The sizes of all scanned files were determined by console program du [48].

About 74 % of scanned files are smaller than 10 kB and 96 % files are smaller than 50 kB (see Graph 8.5). They are very small files, thus it is not necessary to create tests with regard to the size of files. All measured files sizes are listed in Table B.5.

### 8.1.8 Feature: Recursion

The recursion is one of the basic practices in programming. Named templates in XSLT (element “`template`” with attribute “`name`”) can be understood as functions. Thus, we wanted to find their recursive calls in scanned files. The

| **XSLT Version** | **Number of rate** | **Percentage** |
|---:|---:|---:|
| 1.0 | 708 765 | 99 % |
| 2.0 | 7 692 | 1 % |
| unknown | 224 | 0 % |
| **sum** | 716 681 | 100 % |

Table 8.4: **Elements versions**: In the table there are numbers of rates and percentage of elements in sum for individual XSLT versions.

number of recursions (a cycle of calling named templates) and the longest cycle (the longest cycle of calling named templates) was found.

We found out, that recursion is not very frequent. 87 % scanned files do not involve recursion and 7 % have only 1 recursion cycle (see Table 8.5). Moreover, the longest recursion cycle in files has length 1 (the template calls itself) in 95 % (see Table 8.6). All measured numbers of recursions are listed in Table B.6 and all measured longest recursion cycles in files are in Table B.7.

| **Number of recursions** | **Number of files** | **Percent** |
|---:|---:|---:|
| 0 | 5 058 | 87 % |
| 1 | 414 | 7 % |
| 2-5 | 251 | 4 % |
| **sum** | 5 723 | 99 % |

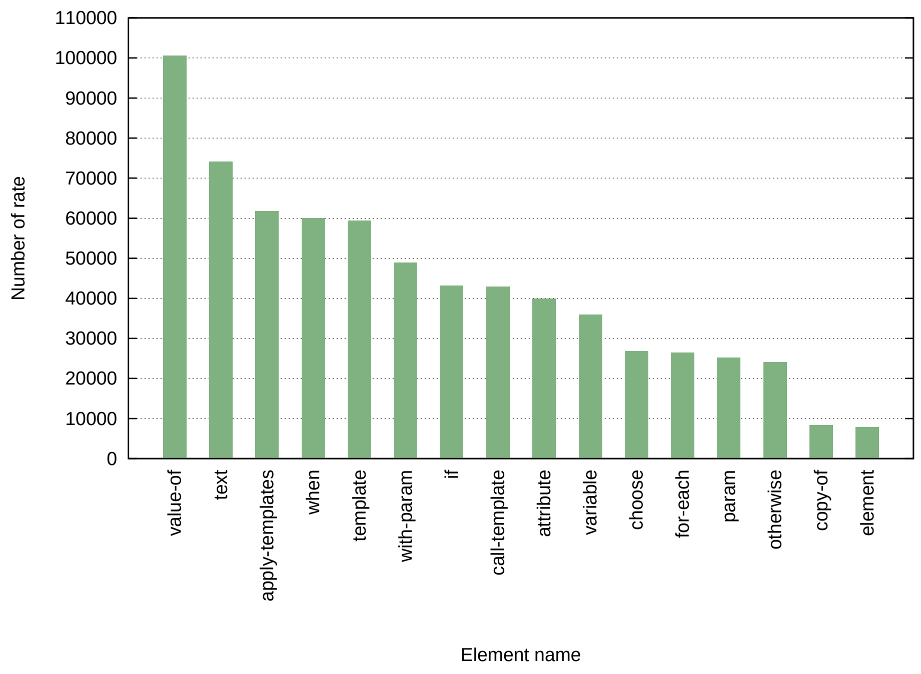
Table 8.5: **Numbers of recursions**: In the table there are numbers of scanned files and percentage of their relevant measured numbers of recursions for significant groups.

### 8.1.9 Feature: Output Format

Output format of an XSLT document is one of its bases features.

The output format can be simply found in attribute "`method`" in top-level element "`output`". When output format is not set, we assume it to be "XML". However, if the first output element of transformation is element "`html`", we assume the output format to be "HTML".

The explicitly set output format (by element "`output`") was saved during scanning of every file. Next, potential output of element (as element "`<html>`" or by element "`<xsl:element ...>`") was searched for.



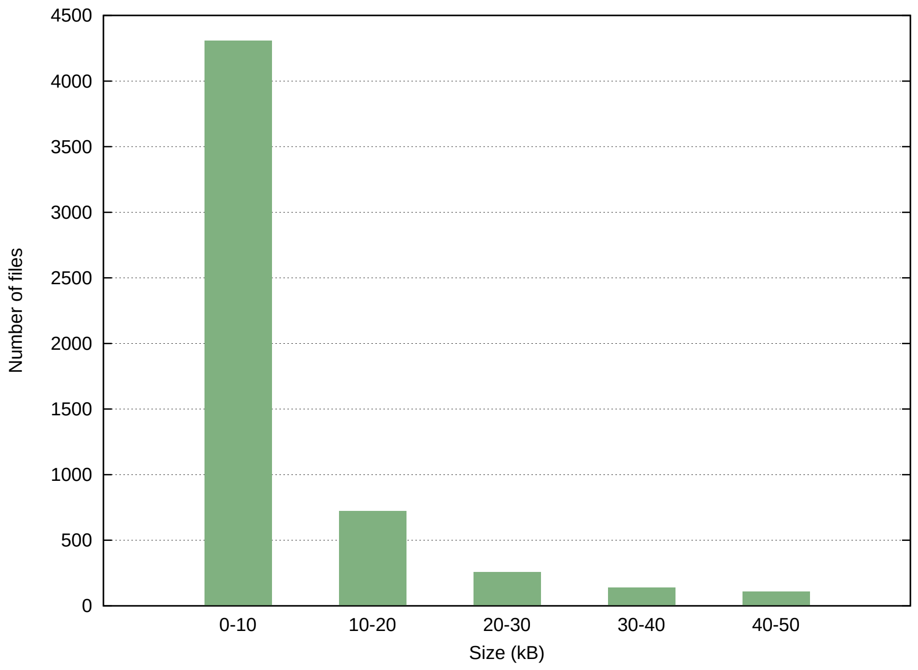
Graph 8.4: **Number of elements rates**: The graph shows rates of elements in scanned files. The graph is restricted to 16 most frequent elements, which in sum represent 95 % of all found out elements from namespace "`xsl`".

We found out that in more than 50 % of all scanned files there is no preset output format. 87 % of these files have default output format "XML". Within files with explicitly set output format (using "`output`" element), 50 % are of type "XML", 35 % of type "HTML" and 11 % of type "TEXT". In total 71 % of files generate output in "XML", 23 % in "HTML" and 5 % in "TEXT".

The results of output types occurences are summarized in Table 8.7.

### 8.1.10 Typical XSLT document

Based on the results of scanning we can describe a typical XSLT document. A typical XSLT document is quite small (it has maximum size of 10kB). It has maximum depth 5, it does not contain any nested elements "`for-each`", "`choose`" and "`if`", it does not contain any recursive calling of named templates, it has default output format "XML", it has average fan-out of elements 1 and maximum fan-out between 0 and 40 and it uses only 16 typical



Graph 8.5: **Size of files**: The graph shows numbers of files (axis Y) with respective size of file (axis X). The graph shows only files with maximum size less then or equal to 50 kB, that in sum represent 96 % of scanned files.

elements from XSLT version 1.0. And elements “`template`” are used with attribute “`match`”.

## 8.2 Categories

After scanning all files and describing all their main features, we detect whether they fit into one of the predetermined categories. Firstly, we describe detecting of categories. Secondly, we describe each category separately and discuss results.

The categories were selected based on another research of the Internet. Scripts saved on CD (see Appendix A.2) were used for analyze the categories. Thus, it is possible to analyze other categories by the scripts.

| **Longest recursion** | **Number of files** | **Percent** |
|---:|---:|---:|
| 1 | 677 | 95 % |
| 2 | 25 | 4 % |
| 3 and more | 9 | 1 % |
| **sum** | 711 | 100 % |

Table 8.6: **Lengths of the longest recursions**: In the table there are numbers of scanned files and percentage of their relevant measured lengths of longest recursion cycle for significant groups.

| **Mode** | **Default** / **Set** | **All** | **Percentage** |
|---:|---:|---:|---:|
| XML | 2626 | 4091 | 71 % |
| | 1465 | | |
| HTML | 381 | 1334 | 23 % |
| | 953 | | |
| TEXT | 0 | 316 | 5 % |
| | 316 | | |
| XHTML | 0 | 46 | 1 % |
| | 46 | | |

Table 8.7: **Output**: In the table there are numbers of scanned files and percentage of their relevant measured output type. There are numbers for default output and set output explicitly too.

### 8.2.1 Process of Categorization

All files were scanned and for each file we determined categories the file belongs to.

In particular, for each file we found the number of occurences of properties from the list. The sum of these numbers of occurences, multiplied by a weight assigned to specific property, is called "Property Value" ($PV$).

$$PV = \sum_{properties} \text{(number of occurrences of property)} * \text{(weight of property)} \tag{8.1}$$

All *PVs* were compared to threshold values that were set for each category. On this basis, it was decided which files belong to which categories and for which files decisions have to be made manually.

The list of properties was created separately for each category, as well as threshold values. Lists of properties and threshold values were set by research and repeatedly running of scan and correcting based on files marked for handmade decisions. Finally, lists of files belonging to categories were returned.

The main results of this chapter are numbers of files belonging to categories that are shown in Table 8.8.

| | **Category** | **Number of files** | **Percentage** |
|---|---|---|---|
| 1 | RSS reader | 81 | 1.40 % |
| | RSS generator | 30 | 0.52 % |
| 2 | Google Search Appliance | 13 | 0.22 % |
| 3 | GraphML | 4 | 0.07 % |
| 4 | XGMML | 8 | 0.14 % |
| | LOGML | 0 | 0.00 % |
| 5 | DocBook reader | 719 | 12.42 % |
| | DocBook generator | 1 | 0.02 % |
| 6 | RDF reader | 18 | 0.31 % |
| | RDF generator | 141 | 2.44 % |
| | RDFS reader | 6 | 0.10 % |
| | RDFS generator | 1 | 0.02 % |
| | RGML reader | 0 | 0.00 % |
| | RGML generator | 0 | 0.00 % |

Table 8.8: **Number of files belonging to categories**: In the table there is the list of all categories with numbers of files belonging into them.

Because some of the properties occur multiple times in one file, we created a script that returns the number of occurences or existence of a distinct property in each file. These scripts will be used in tests detecting categories. There is the list of all properties that can be detected in files. Note, that for all string comparisons the case insensitive comparison was used.

1. **File substring** – the number of occurences of the input string

2. **File name** – the file name is identical with entered name
3. **File name substring** – the file name includes entered string
4. **Variable name** – the file includes entered XSLT variable
5. **Template name substring** – the number of occurences of templates with attribute “`name`” with input string
6. **Template match substring** – the number of occurences of templates with attribute “`match`” with input string
7. **Template name count** – the number of templates with attribute “`name`”
8. **Template match count** – the number of templates with attribute “`match`”
9. **Element name** – the number of generated elements by “`element`”, with name equal to input string

### 8.2.2 Category: RSS

RSS [49] is an XML format for publication of news. RSS Feeds are intended for reading news. Channels (links to XML files in the RSS format) are registered into RSS Feed for showing news from them filtered by required criteria.

RSS is very widespread format so we decided to analyze it. We watched two types of XSLT templates. First, there are templates for transformation of XML data into the RSS format, typically into the HTML format. This category is marked as “RSS reader”. Second, there are templates for generating XML data into the RSS format from any of XML formats. This category is marked as “RSS generator”.

In Tables C.1 and C.2 there are listed criteria used for selecting templates belonging to categories “RSS reader” and “RSS generator”, with their input values and weights. The threshold values are 8 and 13 for both categories “RSS reader” and “RSS generator” (see Chapter 8.2.1).

Of all downloaded files, 243 templates (4.2 %) were found to belong to the category “RSS reader”. When we checked these templates manually, we found out, that only 33 % were categorized successfully. Thus, 81 templates

(1.4 %) belong to the category. Most of them are similar, so we create typical template based on them for our tests. The template "`newsfeeds.xslt`" is very frequent so this template will be used as a test too.

Of all downloaded files, 34 templates (0.59 %) were found to belong to the category "RSS generator". When we checked these templates manually, we found out, that only 4 templates do not belong to the category, which is a good result. Thus, 30 templates (0.52 %) belong to the category. Most of them are similar, so we create typical template based on them for our tests.

We manually checked all templates that were identified as potential candidates for categories "RSS reader" and "RSS generator" and found out that none of these templates belongs to these categories, so we can state that the criteria were created correctly.

### 8.2.3 Category: Google Search Appliance

Google Search Appliance [50] is a hardware solution for local searching of documents within a local network of a company. The layout of the result of searching can be modified by special XSTL template [51]. The template includes many parameters (set by XSLT elements "`xsl:variable`") which is not typical. Thus, we decided to use them in tests. The template with default values suffices for our test.

Although Google Search Appliance is a solution intended mainly for intranets, 13 templates (0.22 %) were detected in the downloaded files.

In Table C.3 there are criteria used for selecting templates belonging to the category, with their input values and weights. The threshold values are 5 and 10 for the category.

### 8.2.4 Category: GraphML

GraphML [52] is an XML format for saving a graph structure including additional information.

There are interesting XSLT templates on the web page [53] so we decided to analyze them. The following templates will be used in tests:

1. Template that generates random graphs.

2. Template that adds information for drawing using the algorithm Spring [54].

3. Template that transforms a graph from the format GraphML to the format SVG [55].

In Table C.4 there are listed criteria used for selecting templates belonging to the category, with their input values and weights. The threshold values are 5 and 10 for the category.

From the downloaded files, only 4 templates (0.07 %) were found. These are templates downloadable from [53]. Although a very few templates exist, they are very interesting. These templates will be used in our tests too. We manually checked all templates that were identified as potential candidates for the category and found out that none of these templates belongs to it, so we can state that the criteria were created correctly too.

### 8.2.5 Category: XGMML, LOGML

XGMML [56] is an XML application based on GML [57] which is used for graph description. GraphML (see Chapter 8.2.4) refers to this format, as the related format. Thus, we decided to analyze them too. Also, we decided to analyze the format LOGML [58]. It is the XML format to describe log reports of web servers. It uses XGMML for saving a structure of a web.

In Tables C.5 and C.6 there are listed criteria used for selecting templates belonging to categories "XGMML" and "LOGML", with their input values and weights. The threshold values are 6 and 10 for the category "XGMML" and 6 and 12 for the category "LOGML".

Of all downloaded files, 124 templates (2.14 %) were identified to belong to the category "XGMML". However, it was found out that only 8 templates (0.14 %) belong to it, during the handmade control. These are templates that can be downloaded from project‘s website [59]. All these templates are very similar, so we will use only one of them in our tests. We manually checked all templates that were identified as potential candidates for the category and found out that none of these templates belong to it, so we can state that the criteria were created correctly.

In the downloaded files, no templates were detected for the category "LOGML", even though we used different settings of criteria. Thus, no test will be created for this category.

Given the small number of detected templates, we can assume that these formats are not intensively used. Alternatively it may be caused by the fact that the authors usually choose not to publish such templates on the Internet.

### 8.2.6 Category: DocBook

DocBook [60] is an XML format for structured writing of documents. Because of XML format, it can be transformed by XSLT templates into different formats of documents (e.g. HTML, PDF, HTML Help).

DocBook is very widespread format and lots of XSLT templates for transformation of XML files to DocBook format and vice versa can be found on the Internet, so we decided to analyze those templates as well. We found two types of DocBook-related XSLT templates. Templates for transformation of XML to DocBook format belong to the first type. We will call this category "DocBook reader". The second type of templates generates DocBook format from other XML-based formats. This second category will be called "DocBook generator".

In Tables C.7 and C.8 there are listed the criteria used for selecting templates belonging to categories "DocBook reader" and "DocBook generator", with their input values and weights. The threshold values are 6 and 10 for the category "DocBook reader" and 8 and 12 for the category "DocBook generator".

Of all downloaded files, 719 templates (12.42 %) were found to belong to the category "DocBook reader". When we manually checked these templates, we found out, that the major part of them belongs to the category. Moreover, we found out, that these templates are mainly prearranged templates downloadable from [61]. Thus, these templates will be use for our tests. The transformation consists of a large number of templates and XSLT element "`include`" is used very often. Thus, we will create a special test for testing this element. We manually checked all templates that were identified as potential candidates for the category and found out that none of these templates belongs to it, so we can state that the criteria were created correctly.

Of all downloaded files, 4 templates (0.07 %) were found to belong to the category "DocBook generator". When we manually checked these templates, we found out, that only 1 template (0.017 %) belongs to the category. We manually checked all templates that were identified as potential candidates for the category and found out that none of these templates belongs to it, so we can state that the criteria were created correctly. Thus, generating XML in DocBook format is not as frequent as reading XML in DocBook format. Thus, no test will be created for the category "DocBook generator".

### 8.2.7 Category: RDF, RDFS, RGML

RDF [62] is a family of specifications, used for modeling information. Its basic idea is a triplet *subject-predicate-object*, where the subject denotes the resource, and the predicate denotes traits or aspects of the resource and expresses a relationship between the subject and the object. RDF is very widespread format. RDFS (RDF Schema) [63] purpose is to extend RDF with descriptions of classes and their attributes. Next, we found an XML format RGML [64] during analyzing XGMML (see Chapter 8.2.5). It is a vocabulary for description of graphs by RDF. For all 3 categories, we detected separately templates for reading ("RDF reader", "RDFS reader" and "RGML reader") and templates for reading any XML and generating the given format ("RDF generator", "RDFS generator" and "RGML generator").

In Tables C.9, C.10, C.11, C.12, C.13 and C.14 there are listed the criteria used for selecting templates belonging to all the mentioned categories, with their input values and weights. The threshold values are:

1. 7 and 10 for the category "RDF reader"
2. 10 and 13 for the category "RDF generator"
3. 10 and 13 for the category "RDFS reader"
4. 10 and 13 for the category "RDSF generator"
5. 6 and 10 for the category "RGML reader"
6. 4 and 13 for the category "RGML generator"

Few templates were detected as belonging to categories "RGML reader" and "RGML generator". However, no templates truly belong to these categories. Thus, we will not create any test for these categories.

We focused on description of classes in categories "RDFS reader" and "RDFS generator". Some RDFS elements and attributes are only meant to extend RDF (e.g. "`rdfs:seeAlso`" or "`rdfs:range`"). Thus, we focused mainly on elements "`rdfs:class`" and "`rdfs:subClassOf`", that describe information by classes and sub-classes like object oriented programming languages.

Of all downloaded files, 88 templates (1.52 %) were found to belong to the category "RDFS reader". However, only 6 templates (0.1 %) truly belong to it. We manually checked all templates that were identified as potential candidates for the category and found out that none of these templates belong to it, so we can state that the criteria were created correctly. Thus, no test will be created for this category.

Of all downloaded files, 14 templates (0.24 %) were found to belong to the category "RDFS generator". However, only 1 template (0.02 %) belongs to it. We manually checked all templates that were identified as potential candidates for the category and found out that none of these templates belong to it, so we can state that the criteria were created correctly. Thus, no test will be created for this category.

Analyzing category "RDF generator" was more successful. In the downloaded files, 141 templates (2.44 %) were detected as candidates for the category and all these templates belong to it. Of all downloaded files, 56 templates (0.97 %) were found to belong to the category "RDF reader". However, only 18 templates (0.31 %) belong to it.

During manual check, we found out that the templates have only one speciality for categories "RDF generator" and "RDF reader". Using many different namespaces. Thus, we will create special tests for namespaces. Creating tests based on RDF is not necessary.

## 8.3 Summary

During the analysis of features of downloaded files, we found out that a typical XSLT document is very simple (see Chapter 8.1.10). Thus, we will create tests based on typical values of scanned features. Also, we will create tests for typical 16 elements used in XSLT documents.

During the analysis of categories of downloaded files, we found out that some categories only have a few representatives or are based on one particular pattern. Next, some categories are not presented in downloaded files. Finally, some categories are presented largely in the downloaded files. Thus, we will not create any test for some categories, we will create base test for some categories and we will use pattern for creating test for some category.

It would be possible to analyze many other categories (e.g. SVG [55]). Interestingly, the procedure could be an analysis of the namespaces used in the downloaded files. Also, it would be possible to analyze other features or their combinations. Some of them were discused in chapters about analysing features (see Chapter 8.1). For example, analysis of XPath expressions used in XSLT files. It would be very interesting and complex analysis.

# Chapter 9

# Test Environment

It was required to create test environment, which enables running of parameterized tests, their running in different XSLT processors and reporting of results. Next, it was required that test environment allows easy adding of non-parameterzied tests, adding more XSLT processors, running on different operating systems and monitoring of time and memory usage.

We implemented all requirements in our program called XSLT Benchmarking [65]. We will discuss individual parts of the program in next chapters. In addition, we will discuss possible extensions and improvements. Most of the functionaly was implemented by using drivers, so it is easy to add new features by just adding a new driver with distinct interface.

We choose PHP as a language for the implementation. The main advantage is easy portability between OSs and its ability to run from console. Moreover, it would be easy to add an extension for running from web browser. That would be very comfortable to run tests.

The user manual is in file `README.markdown`[1] [67]. Of course, we generated the API documentation [68] too. Note, the program, API documentation and reports of tests are saved on the attached CD (see Appendices A.3, A.4 and A.5).

[1]Manual is written in Markdown [66] format, that is a format for easy-to-read, easy-to-write plain text format and converting plaint text to HTML.

## 9.1 Parameterization

The parameterization was one of the natural basic requirements. Therefore, it is possible set many parameters in command line that influence running of tests (selecting tests to run, select tested processors and many others). On the other hand, simplicity was one of the basic requirements too. Therefore, most parameters have default values. Few basis options are sufficient for basic running of program. Setting parameters by the means of an configuration file could be possible extension.

Using should be easy for programmer too. Thus, we created PHP library PhpOptions [69] for working with command line parameters. Due to this, working with parameters of our program XSLT Benchmarking was easier, clearer and extensible.

The program can run on Windows and Linux OS. The script "`run.bat`" is for running on Windows and the script "`run.sh`" is for running on Linux. There is the list of some examples of running the program with descriptions. All examples are demostrated with Windows variant "`run.bat`". However, it would be same for Linux variant `run.sh`. The full summary of parameters of the program is in Appendix D.

1. `run.bat -h`
   Print the help of the program.

2. `run.bat -g`
   Generate tests from all tests templates (use the default directory for templates, tests and temporary files).

3. `run.bat -g --templates-dirs "elements-choose,rss-reader"`
   Generate tests from test temapltes "elements-choose" and "rss-reader" in the default directory.

4. `run.bat -r`
   Run all tests (use the default directory for tests, reports and temporary files).

5. `run.bat -r --tests-dirs "elements-choose-long,`
   `rss-reader-html"`
   Run tests "elements-choose-long" and "rss-reader-html" in the default directory.

6. `run.bat -grvc --repeating 10`
   Generate tests from all templates and run all generated tests. Verbose mode is on. Finaly convert the generated XML report into the HTML format. Additionally, all transformations will be 10 times repeated for the better measure of time and memroy usages.

## 9.2 Generating Tests

Possible ways of creating tests increase flexibility of the program. We can create individual tests manually or create templates of tests, from which tests are generated finally.

Each test (created manually or generated form test template) is in one directory. It includes XSLT template and couples of input and expected output files. The directory also contains XML configuration file that defines test name, individual couples of files (input and expected output) and the name of respective XSLT template. There is an example of definition of a simple test consisting of 2 couples, in Example 9.1. The full description of all possibilities of the XML file is in Appendix E.

```
<?xml version="1.0" encoding="UTF-8"?>
<test name="Modify element - Rename"
   template="test.xslt">
   <couple input="oneElement.xml"
      output="oneNewElement.xml"/>
   <couple input="twoElements.xml"
      output="twoNewElements.xml"/>
</test>
```

Example 9.1: **__params.xml**: Sample test definition. It contains the name of the test "Modify element - Rename", name of XSLT template "test.xslt" and two pairs of input and expected output files.

It is possible to generate tests based on tests templates. The tests template is a directory that includes a template for generated XSLT templates and templates for generated XML files used in the generated tests. Tests to be generated are defined in an XML configuration file. The configuration file is also in directory of tests template. There is an example of definition of a template for two tests in Example 9.2. The full description of all possibilities of the XML file is in Appendix E.

```
<?xml version="1.0" encoding="UTF-8"?>
<tests name="Modify element" template="test.tpl.xslt"
   templatingType="smarty">
   <files>
      <file id="zero">zeroElement.xml</file>
      <file id="one">oneElement.xml</file>
      <file id="oneNew">oneNewElement.xml</file>
      <generated id="many" generator="easy"
         output="manyElements.xml">
         <setting name="testName">20</setting>
         <setting name="testName2">3</setting>
      </generated>
      <generated id="manyNew" generator="easy"
         output="manyNewElements.xml">
         <setting name="newTestName">20</setting>
         <setting name="testName2">3</setting>
      </generated>
   </files>
   <test name="Rename">
      <file input="one" output="oneNew" />
      <file input="many" output="manyNew" />
      <setting name="action">rename</setting>
      <setting name="newName">newTestName</setting>
   </test>
   <test name="Remove">
      <file input="one" output="zero" />
      <file input="many" output="zero" />
      <setting name="action">remove</setting>
   </test>
</tests>
```

Example 9.2: **__params.xml**: The file that defines tests template named "Modify element". There are defined two tests to be generated called "Rename" and "Remove". Both generated tests use two input files ("oneElement.xml" and "manyElements.xml"), where the first one is prearranged and the Easy XML generator (attribute "/tests/files/generated/@generator") generates the second one. Attributes "/tests/test/file/@output" define the expected outputs of transformations for XSLT template and relevant inputs. The template of the generated XSLT templates is defined in attribute "/tests/@template" and it is common for all generated tests. The Smarty XSLT generator is used for generating XSLT template (attribute

"/tests/@templatingType"). Elements "settings" set settings of generated XML or XSLT files by generators.

Different templating types could be used for generating. They are implemented by drivers. Thus, it is very easy to add new templating systems. Let us mention two used important templating system.

The first system is Smarty [70]. It is projected for templating of HTML pages. Due to affinity of HTML, XML and XSLT languages, it is very easy to use it. Smarty includes constructs like cycles, conditions and many others. Thus, it is a very strong tool.

The second system is ToXgene [71]. It is projected for generating random XML files. It can be used to generate XSLT templates. However, the main purpose of its using was to generate random input and expected output XML files. It is not the only existing XML generator. New drivers could be added too (e.g. for XML generators VeXGene [72], MemBeR [73] or NiagDataGen [74]). We drew information mainly from report [1], which does not discuss only XML generators. ToXgene was rated as very powerful tool. On the other hand, it needs complex templates even for generating simple XML files. We tried to avoid similar complexity in our program XSLT Benchmarking.

We thought about templating system based on XSLT transformation. However, dependence on XSLT processor used for XSLT templating system could affects results. We wanted to avoid this dependence. Thus, we did not implement templating system based on XSLT. However, it is not problem to add it by a new driver as well.

In addition, we can compare both the mentioned drivers. The generated results are straightforward by Smarty, which is an advantage. On the other hand, almost all generated outputs have to be written by hand, which is a disadvantage. We can easily get different outputs from ToXgene by simple change of a random seed, which is an advantage. Disadvantage is the mentioned complexity of needed template for generating any outputs. From our experience, we recommend the Smarty driver for generating XSLT templates and ToXgene for generating XML files.

## 9.3 Running Tests

We can run the prepared tests in the next part of our program. Each test is run in each tested processor. A report of tests is saved in an XML file. One test includes one XSLT template and serial couples of files as mentioned in Chapter 9.2. Quaternion *processor- XSLT template- input- expected output* determines one record in the report. As we can see, one quaternion determines one transformation in one XSLT processor. The generated errors are written into reports too. The output of generated transformation is compared with *expected output*. The result of the comparison is written into reports too.

Normalization of generated output and expected output have to be done before comparing them, if they are XML or HTML files. In particular, excessive white spaces are eliminated, attributes are alphabetically ordered, empty elements are transferred on short form etc. As mentioned before, normalization in benchmark XSLTMark inspired us, see Chapter 5.1. Of course, normalization is not applied for text output. Also note that, HTML normalization has own specifications. It is more complicated than XML normalization, especially for white spaces. HTML normalization is not fully implemented yet. Nevertheless, it was sufficient for our usage. It would be extended for greater use of tests with HTML output.

Time and memory usages are measured for each transformation. It means how much time and memory each transformation takes. The measured values are saved in the report with each quaternion. It is possible to set the number of repeating of each transformation for more precise measurement of values by command line property of our program. Time and memory usages are measured for each transformation separately. Average values are reported. It is for reduction of deviation of machine on which the program is run.

Measurement of time usages was not too big problem. Measurement of memory usages was little more complicated. We had to distinguish between running of our program on Linux and Windows. We used command line program `time` [75] on Linux. It runs a certain command and returns information about process including the time usage. We found Windows alternative of program `time` [76].

We can select processors that will be tested in our program by command line parameter. There are different processors on Linux and Windows (e.g. MSXML, see Chapter 4.5, cannot be tested on Linux). It is possible to list available processors on actual machine. There is an example of possible output for command "`run.bat -a`" on Windows system in Example 9.3. In addition, it is possible to write drivers for new processors. After that, we can run tests for these processors separately. Note that it is usually necessary to implement different behavior of the processor for each OS.

```
Available processors:
---------------------
     SHORT NAME |    KERNEL | FULL NAME:
-----------------------------------------
 libxslt1123php |   libxslt | libxslt 1.1.23 - PHP
        msxml30 |     MSXML | MSXML 3.0
        msxml60 |     MSXML | MSXML 6.0
sablotron103cmd | Sablotron | Sablotron 1.0.3
       saxon655 |     Saxon | Saxon 6.5.5
    saxonhe9402 |     Saxon | Saxon HE 9.4.0.2
     xt20051206 |        XT | XT 20051206
       xalan271 |     Xalan | Xalana 2.7.1
   xsltproc1123 |   libxslt | xsltproc 1.1.23
   xsltproc1126 |   libxslt | xsltproc 1.1.26
```

Example 9.3: **Getting a list of available processors**: This is example of getting of a list of available processors for testing. This output is returned for command "`run.bat -a`" on Windows system.

## 9.4 Reports

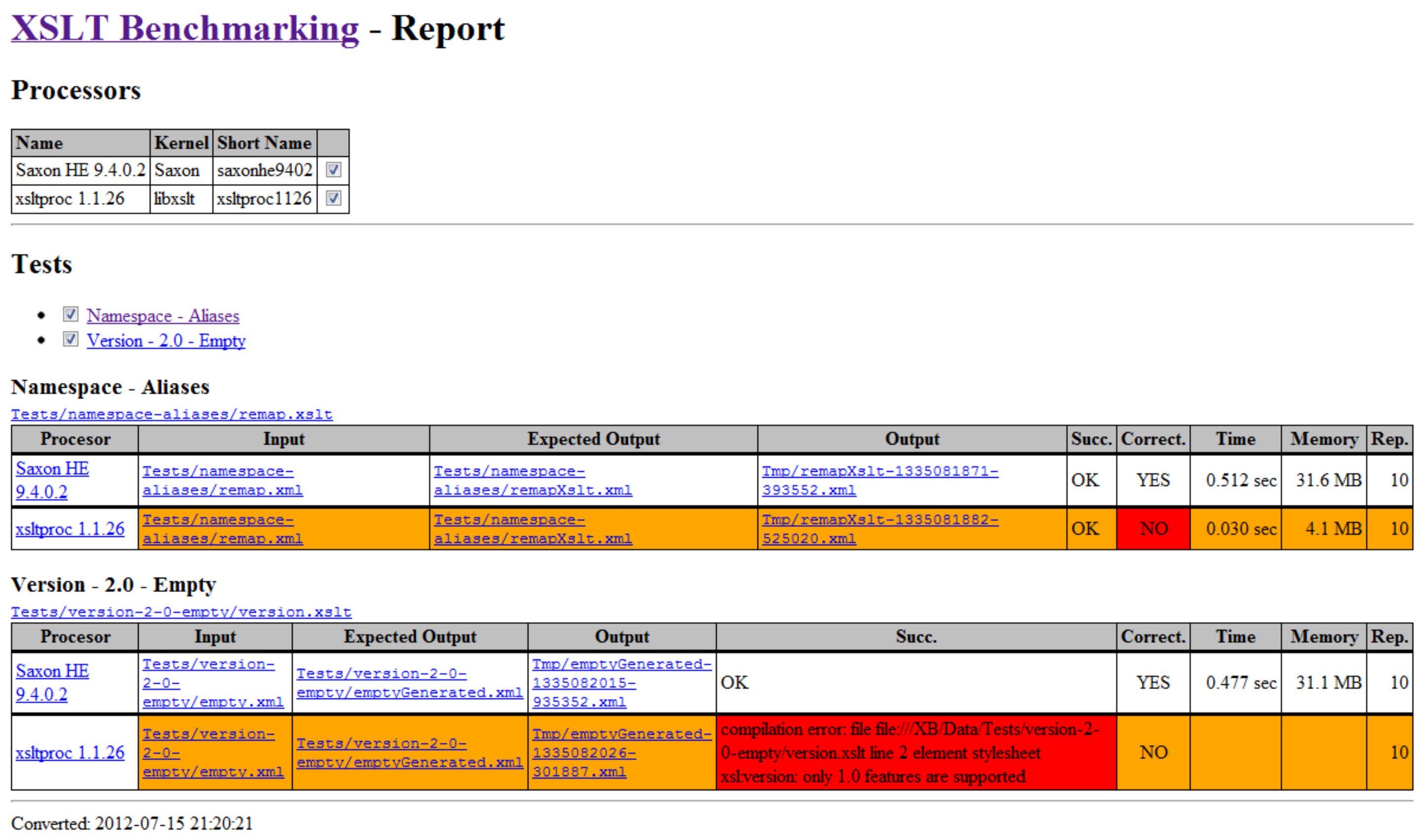
Reports about run tests are finally saved into an XML file. There is an example of XML report in Example 9.4. The full description of all possibilities of the XML file is in Appendix E. It is possible to convert the XML file into HTML format for better analyzing. There is an example of HTML report in Figure 9.1. Moreover, it is possible to create drivers for converting reports into other formats (e.g. CSV[2], TeX[3] table etc.). There are links to all relevant files (XSLT templates, input files, expected output files and generated output files) in HTML reports. This simplifies the analysis. Reports of our tests are presented on [65].

[2]A comma-separated values (CSV) file stores tabular data (numbers and text) in plain-text form.

[3]TeX is a typesetting system.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<reports>
  <global>
    <processors>
      <processor name="saxonhe9402" fullName="Saxon HE 9.4.0.2"
         kernel="Saxon"/>
      <processor name="xsltproc1126" fullName="xsltproc 1.1.26"
         kernel="libxslt"/>
    </processors>
  </global>
  <tests>
    <test name="Namespace - Aliases"
       template="/XB/Data/Tests/namespace-aliases/remap.xslt">
      <processor name="saxonhe9402">
        <input input="/XB/Data/Tests/namespace-aliases/remap.xml"
           expectedOutput="/XB/Data/Tests/namespace-aliases/remapXslt.xml"
           output="/XB/Tmp/remapXslt-1335081871-393552.xml" success="1"
           correctness="1" sumTime="5.125227" avgTime="0.512522"
           sumMemory="315596000" avgMemory="31559600" repeating="10"/>
      </processor>
      <processor name="xsltproc1126">
        <input input="/XB/Data/Tests/namespace-aliases/remap.xml"
           expectedOutput="/XB/Data/Tests/namespace-aliases/remapXslt.xml"
           output="/XB/Tmp/remapXslt-1335081882-525020.xml" success="1"
           correctness="0" sumTime="0.300028" avgTime="0.030002"
           sumMemory="40624000" avgMemory="4062400" repeating="10"/>
      </processor>
    </test>
    <test name="Version - 2.0 - Empty"
       template="/XB/Data/Tests/version-2-0-empty/version.xslt">
      <processor name="saxonhe9402">
        <input input="/XB/Data/Tests/version-2-0-empty/empty.xml"
           expectedOutput="/XB/Data/Tests/version-2-0-empty/emptyGenerated.xml"
           output="/XB/Tmp/emptyGenerated-1335082015-935352.xml" success="1"
           correctness="1" sumTime="4.773580" avgTime="0.477358"
           sumMemory="311328000" avgMemory="31132800" repeating="10"/>
      </processor>
      <processor name="xsltproc1126">
        <input input="/XB/Data/Tests/version-2-0-empty/empty.xml"
           expectedOutput="/XB/Data/Tests/version-2-0-empty/emptyGenerated.xml"
           output="/XB/Tmp/emptyGenerated-1335082026-301887.xml"
           success="compilation error: file
           file:///XB/Data/Tests/version-2-0-empty/version.xslt line 2 element
           stylesheet&#13;&#10;xsl:version: only 1.0 features are
           supported&#13;&#10;&#13;&#10;" correctness="0" sumTime=""
           avgTime="" sumMemory="" avgMemory="" repeating="10"/>
      </processor>
    </test>
  </tests>
</reports>
```

Example 9.4: **report.xml**: There are XML report of testing of two XSLT processors Saxon HE 9.4.0.2 and xsltproc 1.1.26 in two tests “Namespace - Aliases” and “Version - 2.0 - Empty”.



XSLT Benchmarking - Report

Processors

| Name | Kernel | Short Name | |
|---|---|---|---|
| Saxon HE 9.4.0.2 | Saxon | saxonhe9402 | ✓ |
| xsltproc 1.1.26 | libxslt | xsltproc1126 | ✓ |

Tests

- ✓ Namespace - Aliases
- ✓ Version - 2.0 - Empty

Namespace - Aliases

Tests/namespace-aliases/remap.xslt

| Procesor | Input | Expected Output | Output | Succ. | Correct. | Time | Memory | Rep. |
|---|---|---|---|---|---|---|---|---|
| Saxon HE 9.4.0.2 | Tests/namespace-aliases/remap.xml | Tests/namespace-aliases/remapXslt.xml | Tmp/remapXslt-1335081871-393552.xml | OK | YES | 0.512 sec | 31.6 MB | 10 |
| xsltproc 1.1.26 | Tests/namespace-aliases/remap.xml | Tests/namespace-aliases/remapXslt.xml | Tmp/remapXslt-1335081882-525020.xml | OK | NO | 0.030 sec | 4.1 MB | 10 |

Version - 2.0 - Empty

Tests/version-2-0-empty/version.xslt

| Procesor | Input | Expected Output | Output | Succ. | Correct. | Time | Memory | Rep. |
|---|---|---|---|---|---|---|---|---|
| Saxon HE 9.4.0.2 | Tests/version-2-0-empty/empty.xml | Tests/version-2-0-empty/emptyGenerated.xml | Tmp/emptyGenerated-1335082015-935352.xml | OK | YES | 0.477 sec | 31.1 MB | 10 |
| xsltproc 1.1.26 | Tests/version-2-0-empty/empty.xml | Tests/version-2-0-empty/emptyGenerated.xml | Tmp/emptyGenerated-1335082026-301887.xml | compilation error: file file:///XB/Data/Tests/version-2-0-empty/version.xslt line 2 element stylesheet xsl:version: only 1.0 features are supported | NO | | | 10 |

Converted: 2012-07-15 21:20:21

Figure 9.1: **report.html**: There are HTML report of testing of two XSLT processors Saxon HE 9.4.0.2 and xsltproc 1.1.26 in two tests "Namespace - Aliases" and "Version - 2.0 - Empty".

It is possible to merge more reports files together. Thus, we can add a driver for another XSLT processor, run tests only for this processor and merge generated reports with already generated reports. Similarly, we can add a new test, run it for all processors and merge generated reports with already generated reports.

At this moment, the reports do not include information about machine (OS, RAM[4] etc.) on which it is run. Creating of a separate reports for each machine was sufficient for our analysis. It could be an extension to add information about the machine into reports. After that, it will be possible to merge any reports. The design of HTML reports poses a very interesting problem. For example, potential merging of all our reports from all machines would be very long with very much information. It would be necessary to arrange sufficient clarity.

## 9.5 Extension Proposals

We already mentioned some possible extensions of our program XSLT Benchmarking. So far we mentioned these extensions: running the program using a web browser, setting of input parameters in configuration file, adding drivers for new templating systems for XSLT or XML files, upgrade normalization of HTML files, adding drivers for other XSLT processors, adding information about machine into reports and add drivers for converting reports to another formats.

Next possible extension would be new drivers for reading configuration files in other formats than XML (e.g. INI[5], JSON[6], YAML[7] etc.) for tests and templates of tests.

We also detected different levels of reports of warnings and errors by XSLT processors during creating of tests. Expected errors in tests would be a very interesting extension. For example, the analysis of the returned error would contain some words. Analysis of returned errors and warnings by XSLT processors would be very beneficial (their information value, precision etc.).

[4]Random Access Memory (RAM)

[5]INI files are simple text files with a basic structure composed of “sections” and “properties”.

[6]JavaScript Object Notation (JSON) is a lightweight text-based open standard designed for human-readable data interchange.

[7]YAML is a human-readable data serialization format that takes concepts from programming languages such as C, Perl, and Python, and ideas from XML.

Allowing to set more expected outputs would be next interesting extension. More expected outputs or a combination of expected outputs and potential errors is possible because of variability of XSLT W3C specification [2]. Both extensions are beyond the scope of this thesis. However, there are proposals of definition both extensions together for XML configuration file of template of tests, in Example 9.5.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tests name="Extended" template="test.tpl.xslt"
   templatingType="simple">
   <files>
      <file id="input1">input1.xml</file>
      <file id="input2">input2.xml</file>
      <file id="input3">input3.xml</file>
      <file id="input4">input4.xml</file>
      <file id="input5">input5.xml</file>
      <file id="input6">input6.xml</file>
      <file id="first">firstOutput.xml</file>
      <file id="second">secondOutput.xml</file>
   </files>
   <test name="Comlex example">
      <file input="input1" output="first" />
      <file input="input2" error="error substring" />
      <file input="input3" output="first" error="error
         substring" />
      <file input="input4">
         <error>error substring</error>
         <error>another error substring</error>
      </file>
      <file input="input5">
         <output>first</output>
         <output>second</output>
      </file>
      <file input="input6">
         <error>error substring</error>
         <error>another error substring</error>
         <output>first</output>
         <output>second</output>
      </file>
   </test>
```

```
32 </tests>
```

Example 9.5: **__params.xml**: The file, that defines the template of tests named "Extended". There is a draft definition of possibility of more expected outputs or expected errors. There is an example of one test with 6 input files. The definition of the first file has standard one expected output. The second definition has an expected error that contains string "error substring". The third one has an expected error or correct transformation with one expected output. The fourth one has an expected error that contains both set strings. The fifth one has an expected correct transformation the same as one of the set outputs. Finally, the sixth one has an expected error that contains both set strings or correct transformation the same as one of set outputs.

Testing of streaming of input XML file and generated output file would be very interesting. It is an up-to-date topic. However, most processors do not support it. In addition, it is a very complex topic and it is beyond the scope of this thesis. We recommended report [77] for more information about this issue.

## 9.6 Summary

We implemented all requirements in our program XSLT Benchmarking. We created parameterization test environment, which could be affected by many parameters. It allows for creating both simple and complex tests. Information about time and memory usages are saved in reports. It could be run on different OS. It is extensible by means of drivers (drivers for tested processors, templating systems etc.). We found many aspects of the program that could be extended. Thus, the program is shared on [65].

Interesting usage of our program would be to test applications based on XSLT transformations. Thus, it would be similar to PHPUnit [78], which is designed for testing of PHP applications. It would be useful to make a driver for conversion of reports, which would generate text summary. Finally, it would be useful to create the script, which would run required tests, generate summary of reports and print them on a console.

# Chapter 10

# Tests

We had to create XSLT templates, input XML files and expected output XML, HTML or text files during creating of tests. Most of tests were generated from templates of tests. Due to parameterization, it is possible to create new tests with other settings according to specific needs. We created tests mainly based on XSLT specification [2].

The main purpose of the test is to detect the corectness of outputs of transformations by all processors. Nevertheless, some tests have also other purposes. For example, some tests have big input data, big XSLT template or big expected output. These tests are sutable to track time and memory usages. Purposes of tests will be discussed in this chapter and in Chapter 11 that discusses results of the tests.

Some XSLT templates are designed as synthetic and some XSLT templates could be implemented more efficiently. Consistent testing of some XSLT elements was the main purpose for it. Of course, we created some tests based on real use or created tests that tested real XSLT templates used in practice. These tests are mainly tests of categories discussed in Chapter 8.2. The list of all tests are in Appendix F.

Note, that we tried to find tests used in XSLTMark (see Chapter 5.1) and use them in our testing. Unfortunately, only results were available on the Internet.

## 10.1 Features

Tests based on analyses of features (see Chapter 8.1) are mainly synthetic, because they test specific features. We did not create special tests for extreme values of researched features. We created tests focused on most frequently occurring elements (see Chapter 8.1.6).

We did not create test on XSLT elements “`value-of`”, “`copy-of`” and “`variable`”, because they are strongly linked with XPath [8], which we did not test in our thesis (see Chapter 8.1). Basic usages of these elements are covered in other tests. We created tests for other frequent XSLT elements. Note, that some tests covered more elements together. For example, tests with prefix “Elements element” covered XSLT elements “`element`” and “`attribute`” and tests with prefixes “Elements template” covered XSLT elements “`template`”, “`apply-templates`”, “`with-param`”, “`call-template`” and “`param`”.

We also created two tests that test XSLT 2.0. The first test is almost empty transformation with XSLT template, which only declares using of XSLT 2.0. It is test “Version - 2.0 - Empty”. The second test tests real support of XSLT 2.0 elements by processors. We used the most frequent XSLT 2.0 elements “`output-character`” and “`character-map`”. The test also tests XSLT 2.0 attribute “`use-character-maps`” from element “`output`”. It is test “Version - 2.0 - Character-map”. Both tests have empty input XML data. Expected outputs are different. The first test expectes empty XML data and the second test expects XML including listed data in XSLT template filtered by the “`character-map`”.

We assume that users of our program XSLT Bechmarking will create specialized tests for their XSLT applications.

We also decided to create a typical XSLT template (see Chapter 8.1.10), which contains the most frequent elements and has others features based on our analyses. We implemented these requirements in test “Typical - Budget”. The XSLT template expects an XML document with information about orders of companies. It generates an XML document including budgets of companies on an output. The input XML document is very big and it is generated by ToXgene. The XSLT template is very small.

## 10.2 Categories

Tests of analyzed categories are not synthetic. Either they are real XSLT templates, which were found in downloaded files (see Chapter 7) or they were created based on the most frequent elements and principles of XSLT templates belonging to categories. Each category has its own specifics, thus their tests are very different. It is very suitable for our testing.

We created two types of tests ("RSS reader" a "RSS generator") for category "RSS" (see Chapter 8.2.2). They are XSLT templates with procedural and also non-procedural approaches in analyzed files for "RSS generator". We created two tests, which have same input and expected output. Thus, we could compare both approaches (for more information see Chapter 11). We created two tests for "RSS reader". The first test contains very frequent template "`newsfeeds.xslt`". The second test was created based on typical elements and it generates HTML output. All input XML files for all these tests were generated by ToXgene and they are big. Input and expected output files were creted based on RSS specification [49] and expected XML elements and listed XML or HTML elements in analyses XSLT templates belonging to category "RSS".

The default template [51] was used for testing the category "Google Search Appliance" (see Chapter 8.2.3). The test XML document [79] was used as input XML document in the test. Expected output is HTML web page similar to results of searching on Google Search [40].

Templates from pages of the project "GraphML" [53] were used for tests of its category (see Chapter 8.2.4). The templates could be concatenated; that was advantage. It means, the first template generates random graph, the second template adds information for better visualisation in plane into it and the third template transforms the graph into the SVG format [55]. Input and expected output files are small.

We used one of the templates and test input file available on pages of the project "XGMML" [59] for testing its category (see Chapter 8.2.5). The input file is big and the expected output is small HTML file.

A prearranged set of templates available on pages of project "DocBook" [61] was used for testing its category (see Chapter 8.2.6). The set contains also test input XML file. The file contains different types of DocBook elements including references, footnotes etc. Thus, the test "Docbook - HTML" is not generated from any template, but it is prearranged manually.

## 10.3 Others

We created two tests based on analysis of templates that belong to the category "RDF". They test more namespaces in one XSLT template or in one XML file. The first test "Namespace - Aliases" tests aliases of namespaces by XSLT element "`namespace-alias`". The second test "Namespace - Rename" tests using of different prefixes of namespaces in XSLT template and input XML file for the same namespace. These tests have small synthetic input and expected output XML files and small synthetic XSLT templates. They focus mainly on correct implementation of functionality around namespaces in tested processors.

Next, we created test "Element include" based on analysis of the category "DocBook". It tests including of other templates into the main template by XSLT element "`include`". The test has small synthetic input and expected output XML files and small synthetic XSLT templates. It focuses mainly on correct behavior of element "`include`".

We detected problems with an encoding (set by XSLT attribute "`encoding`" of element "`output`") during creating tests. Thus, we also created tests with prefix "Encoding", for testing of functionality related to the encoding. We tested default usage, without using of attribute "`encoding`", and explicit setting different encoding. All tests have same set of input XML files. They are files with the same content saved in different encodings. The used encodings are adequately set in XML declarations.

## 10.4 Summary

We created synthetic tests, tests based on real XSLT templates and tests with real XSLT templates. Most tests were generated from templates. Few tests were created manually. The tests included small and also big input files/expected output files/XSLT templates. Diversity of tests presents great flexibility of our test environment.

Additional tests could be created based on further analysis of downloaded files (see Chapter 7) or our analysis (see Chapter 8). For example, it would be interesting to create tests testing more elements from XSLT 2.0 or even from XSLT 3.0 or recursion of named templates (it means XSLT element "`template`" with attribute "`name`"). However, we assume that users of XSLT Bechmarking will create specialized tests for their XSLT applications.

# Chapter 11

# Results

We will discuss results of our tests from different points of view in this chapter. We will compare different versions of processors, different types of processors (e.g. library and program, see Chapter 4), running of tests on different OSs etc. Complete reports of tests are available on [65]. HTML reports were used for major part of our analysis. They enabled us to filter lists of processors or tests on the output. In addition, they enable us to easily show input and output files of tests. These reports can serve as groundwork for the next analysis. For example, if we create other tests or test other processors.

Additionaly, we had to make some special tables for better analysis of some view of the results. This tables are available on [80] as Google Docs Spreadsheet. We mention source of results in next chapters. There are main HTML reports (*HTML reports*) or name of the sheet in the Google Docs Spreadsheet (e.g. *GDS - Small vs. Large Files*).

All unnecessary programs were turned off on machines on which the tests run. In addition, machines were disconnected from the Internet, so that the machine was used only by our tests. Tests were run with setting "`--repeating 10`" for more accurate results (see Chapter 9).

## 11.1 Different OS, RAM or CPU

The created test were run on different machines with different OS, RAM or CPU. Reports of these tests enabled comparative analysis. Basis properties of used machines are in Table 11.1. This results were created based on *GDS - avgMemory - win1 vs. win2*, *GDS - avgMemory - lin1 vs. lin2*, *GDS -*

*avgTime - win1 vs. lin1*, *GDS - avgMemory - win1 vs. lin2*, *GDS - Averages of processors - win1* and *GDS - Averages of processors - lin1*.

| **Shc.** | **System** | **Addr.** | **CPU** | **RAM** |
|---|---|---|---|---|
| win1 | Windows 7 Profesional, SP1 | 64bit | Intel(R) Core(TM) i5 CPU M560, 2.67GHz | 8GB |
| win2 | Windows 7 Home Premium, SP1 | 32bit | AMD Athlon(tm) II X4 635 Processor, 2.90GHz | 4GB |
| lin1 | Ubuntu 11.10 | 64bit | Intel(R) Core(TM) i5 CPU M560, 2.67GHz | 7GB |
| lin2 | Debian 6.0.4 | 64bit | AMD Opteron(TM) Processor 6234, 2.70GHz | 132GB |

Table 11.1: **Machines properties**: There are basis properties of machines, on which tests run. The first column contains shortcuts of machines used in next text.

Checking of reports showed that errors are reported equally on all machines. Thus, there was no test, which did transformation on one machine correctly and on another machine with an error. Thus, processors available on both OS (Windows and Linux) have the same functionality.

The next finding is about available RAM. Comparison of *win1* and *win2* or *lin1* and *lin2* showed that on systems with more available RAM more memory was actually used. Due to this, the transformation was faster.

The most interesting was the comparison of time and memory usages on different OSs. Transformations on *lin1* have worse time and memory usages than on *win1*. About 49 % of transformations on *lin1* were slower by at least 0.1 seconds. Next, about 92 % of transformations on *lin1* used at least 1 MB RAM more than transformations on *win1*. We found out that mostly worse time and memory usages were for Java processors (e.g. Saxon or XT) on *lin1*. On the other hand, command line processors (e.g. xsltproc or Sablotron) were faster on *lin1* in global. We can assume that worse time usages were caused by JVM on *lin1*. Tests of JVM are not included in our thesis. However, it is possible make drivers for processors, which use other JVM, in our test environment (see Chapter 9). This would allow to compare tests for one Java processor run under different JVM.

## 11.2 Different Processors Versions

Next, we compared different versions of XSLT processors. This results were created based on *HTML reports*.

The first examined processor was xsltproc (see Chapter 4.4). We examined versions 1.1.23 and 1.1.26. These two versions were the same in time and memory usages. The only difference was in test “Docbook - HTML”. The older version 1.1.23 did not do the transformation completely correctly. The newer version 1.1.26 did the transformation perfectly. There were no differences in other tests.

The next examined processor was MSXML (see Chapter 4.5). We examined versions 3.0 and 6.0. These two versions did all transformations identically and had about same memory usages. Time usages were different. Transformations had same time usages for small input data. However, the newer version 6.0 was about 1.5 times faster than version 3.0 for large input data (e.g. tests “Elements choose - Long” or “Elements foreach - Long with not presented”).

The last researched processor was Saxon (see Chapter 4.1). We researched versions 6.5.5 and HE 9.4.0.2. These versions have some main differences. The first, older version 6.5.5 is about 2.5 times faster and uses about 1.5 times less RAM than the new version HE 9.4.0.2. Maybe, commercal versions PE 9.4 or EE 9.4 would be better in testing of time or memory usages. Unfortunately, we could not test them for financial reasons. Moreover, it is better to compare versions available for the same price, thus free for this case. The second, newer version HE 9.4.0.2 cannot use Java functions in XSLT templates opposite older version 6.5.5. This fact is described in documentation and we checked it in tests “GraphML - GNM - Random” a “GraphML - Spring - Random and Example”. Versions PE and HE should support it.

The next difference was a support of XSLT 2.0. Version HE 9.4.0.2 supported it, but version 6.5.5 supported only XSLT 1.0. Support of XSL 2.0 will be discussed in detail in the following text. The last difference was reporting of warnings by processors. Version HE 9.4.0.2 informed about a multiple inclusion of one XSLT file during transformation in the test “Docbook - HTML”. This information could be helpful in many cases. Version 6.5.5 ignored this fact as all others tested processors.

## 11.3 Different Languages

We also compared different language versions of processor libxslt 1.1.23 (see Chapter 4.4). This results were created based on *HTML reports*. We compared PHP library and command line program xsltproc. The console variant had smaller time and memory usages in all tests. Probable reason was using of CPU and RAM by PHP.

Bigger difference was in reporting of warnings and errors. The PHP variant did not report errors that were reported by the console variant in general. Moreover, the PHP variant generated an empty XML file without report of error in some tests. Thus, console variant has better error reporting.

## 11.4 Encoding

In tests with prefix "Encoding", we tested if processors can generate output in different encodings and if they can process input XML files in different encodings. All processors, except for XT (see Chapter 4.3), had no problem with different encodings. Processors read input XML in different encodings and they respected set output encoding. If output encoding had not been set, then they used UTF-8, UTF-16 or none encoding in declaration of output XML. None encoding in XML declaration means UTF-8 or UTF-16 encoding, as per W3C XML specification [3].

Only XT processor failed in these tests. The processor could read input XML only with UTF-8 or UTF-16 encoding. It reported errors for another encodings[1], which is all right (see W3C XSLT specification [2]). However, it used UTF-8 encoding for generating output also when UTF-16 encoding was set, which is wrong.

Note, that this results were created based on *HTML reports*.

## 11.5 Categories

The tests of categories examined real transformations. Due to this, we found out several interesting information. This results were created based on *HTML reports*.

[1]The example of a reported error for the encoding Windows-1252 for processor XT: "org.xml.sax.SAXParseException: unsupported encoding"

One of the mentioned problems was dependence of transformation on Java functions for category "GraphML". The transformation runs correctly only for Saxon 6.5.5. Using of Java functions was not supported in other processors. Java functions are supported in new Saxon versions for commercial variants (not for tested free version HE). As we can see, our test environment could be used for finding XSLT processor useful for required XSLT template.

The template for category "Google Search Appliance" was without problem for most processors. Only processor XT did not generate any output. We found out that the template included an unsupported construct. It used "`disable-output-escaping`" for generating an attribute, which is not supported by XSLT specification [2]. All processors (except for XT) had a correct behavior. They ignored using of "`disable-output-escaping`" for generated attribute. Only processor Sablotron reported a warning about it. It could be considered as an advantage.

Processors libxslt 1.1.23 (PHP library also xsltproc), MSXML (3.0 also 6.0) and Sablotron 1.0.3 failed in test of category "DocBook". Other processors generated the expected output. We detected positive behavior of processor Saxon HE in this test. It reported multiple inclusions of one of included XSLT templates. It may not be wrong behavior of set of XSLT templates. Nevertheless, it could warn on potential drawback in performance.

Templates for categories "RSS" and "XGMML" were without problem for all processors.

## 11.6 Small vs. Large Files

This chapter reports about time and memory usages based on the size of input XML files or XSLT templates. This results were created based on *GDS - Small vs. Large Files.*

Tests with prefixes "Elements choose", "Elements foreach" and "Elements if" were created for comparison of transformations with different sizes of XSLT templates or input XML files. These tests confirmed the natural assumption, that for a bigger XSLT template time and memory usages are bigger. Differences were very significant. Increases were between 35 % and 50 % for time usages and between 5 % and 25 % for memory usages. In this chapter, increases are understood as increases of time or memory usage for bigger data (XML files of XSLT templates) against smaller data.

These tests also confirmed the assumption, that for bigger input XML data time and memory usages are bigger. Increases were about 8.7 % for memory usages. Differences were more significant for time usages. In addition, it is necessary to distinguish time usages for small and big XSLT templates. Increases were about 17 % for small XSLT templates for time usages. However, they were about 200 % for big XSLT templates. Processor Sablotron had the biggest increases. Increases were about 1058 % for big XSLT templates and 57 % for small XSLT templates for time usages. Moreover, its increases were the biggest for memory usages compared to other processors. Processor Saxon HE had the smallest increases for time also memory usages. Its increases were 17 % for big XSLT templates and 7 % for small XSLT templates for time usages. Moreover, its increases were the smallest for memory usages compared to other processors.

Tests "Elements element - Indent template" and "Elements element - Not indent template" were created for detection of influence of indentation in XSLT templates (thus needless whitespaces). We assumed that time usages have minimal differences between both approaches or not indent XSLT temlates has better time usages. The assumption was correct for memory usages. Differences were insignificant for major part of processors. The assumption was almost correct for time usages. Time usages were better up to 11 % for non-indent XSLT templates. However, processors MSXML 6.0 and XT had better time usages up to 8 % for indented XSLT template.

## 11.7 Procedural vs. Non-Procedural Approach

This chapter reports about tests focused on differences between procedural and non-procedural access. Thus, using of elements "`template`" mainly with attributes "`name`" (procedural access) or "`match`" (non-procedural access). This results were created based on *GDS - (Non)procedural*. The first couple of tests were tests "Elements template - Nonprocedural" and "Elements template - Procedural". The second couple of tests were tests "RSS generator - apply-templates" and "RSS generator - for-each". All transformations run correctly for all processors. Detection of time and memory usages was the main aim of these tests. Both tests in each couple transformed the same XML input files and expected the same XML output files. Only tranformations were written by procedural or non-procedural approach.

Differences were less than 2.5 % between procedural and non-procedural access for memory usages for almost all processors. Thus, memory usages were not much affected with these different accesses. An exception was processor Saxon 6.5.5 for tests with prefixes “Elements template”. It had lower memory usage for non-procedural access almost upon 10 %.

The results were more complicated for time usages. Differences higher than 5 % were considered as significant. Non-procedural access was more effective in terms of time usages for processors MSXML 3.0, Sablotron, Saxon 6.5.5., Saxon HE and Xalan. Thus, we recommend non-procedural access for these processors. Almost all differences were smaller than 5 % for time usages for processors with kernel libxslt (PHP also xsltproc, all versions). Thus, both accesses have the same efficiency for these processors. Finally, the results were different for processors MSXML 6.0 and XT. Procedural access was efficient for these processors in tests with prefix “RSS generator”. On the other hand, non-procedural access was efficient for these processors in tests with prefix “Elements template”, but not too much. Thus, we recommend procedural access for these processors.

## 11.8 XSLT Version

All processors refer to used XSLT version. Only Saxon HE supports XSLT 2.0. Other processors support only XSLT 1.0. This was confirmed by test “Version - 2.0 - Character-map” that tested elements from XSLT 2.0. Only processors libxslt, Sablotron and xsltproc warned about using of unknown element. Other processors ignored elements from XSLT 2.0 and done transformation without its application. Ignoring unsupported elements (e.g. from XSLT 2.0) could be considered as a flaw, because the user is not informed that the transformation failed.

Moreover, only processor xsltproc warned about unsupported XSLT 2.0 in test “Version - 2.0 - Empty”. There is an empty XSLT template with XSLT version 2.0 in XSLT declaration, in the test. This fact could be considered as positive behavior of xsltproc and negative behavior of other processors.

Note, that this results were created based on *HTML reports*.

## 11.9 XSLT Namespace

Test "Namespace - Rename" was the first test for using of namespaces. It tested working with more namespaces and their different naming in XSLT template and input XML files. All processor passed this test.

The second test "Namespace - Aliases" tested using of XSLT element "`namespace-alias`". Expected generated file was XSLT template. All processors generated valid XSLT templates, which would generate required output. However, only processors MSXML 3.0, MSXML 6.0 and Saxon HE generated expected output. Other processors used namespace prefix used in XSLT template (before output elements) instead of required namespace set by element "`namespace-alias`". In addition, processor Saxon 6.5.5 generated declaration (in root element "`stylesheet`") of both namespaces. Thus, only processors MSXML 3.0, MSXML 6.0 and Saxon HE passed the test.

Note, that this results were created based on *HTML reports*.

## 11.10 Average Time and Memory Usages

We discuss average time and memory usages of all tested processors in this chapter. The compared values for time usages are demostrated in Graph 11.1 and the compared values for memory usages are demonstrated in Graph 11.2. The exact measured values are in Table G.1. This results were created based on *GDS - Averages of processors - win1* and *GDS - Averages of processors - lin1*.

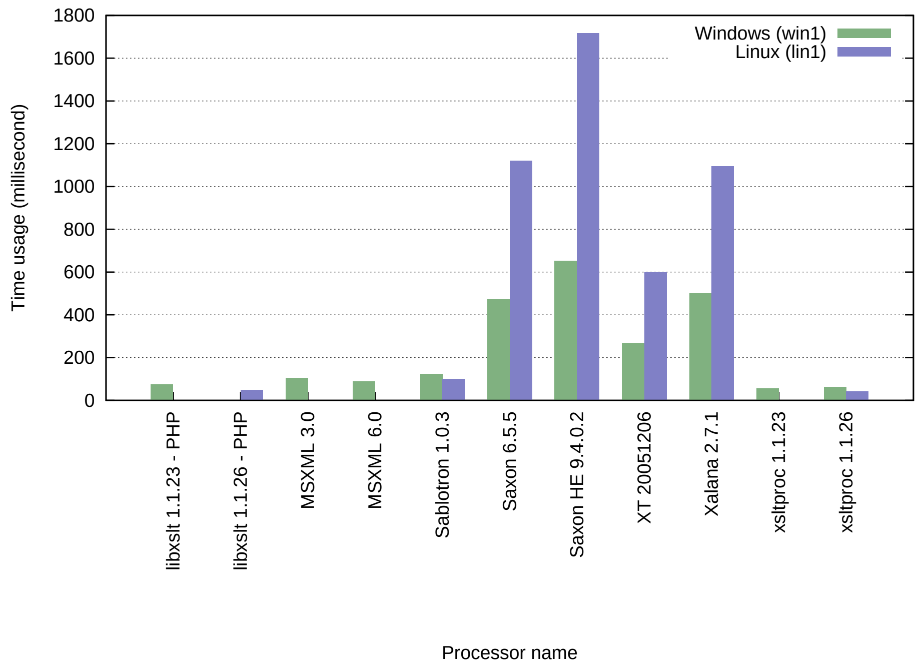
As we can see, Java processors (Saxon, XT and Xalan) have the highest time also memory usages. Command line processors (xsltproc and Sablotron) have the smallest memory usages. Finally, processors with kernel libxslt (libxslt - PHP and xsltproc) have the smallest time usages.

Processor xsltproc 1.1.23 has the smallest time and also memory usages and processor Saxon HE 9.4.0.2 has the biggest time and also memory usages.

Note, that processors MSXML 3.0 and MSXML 6.0 have average time and also memory usage.

## 11.11 Summary of Processors

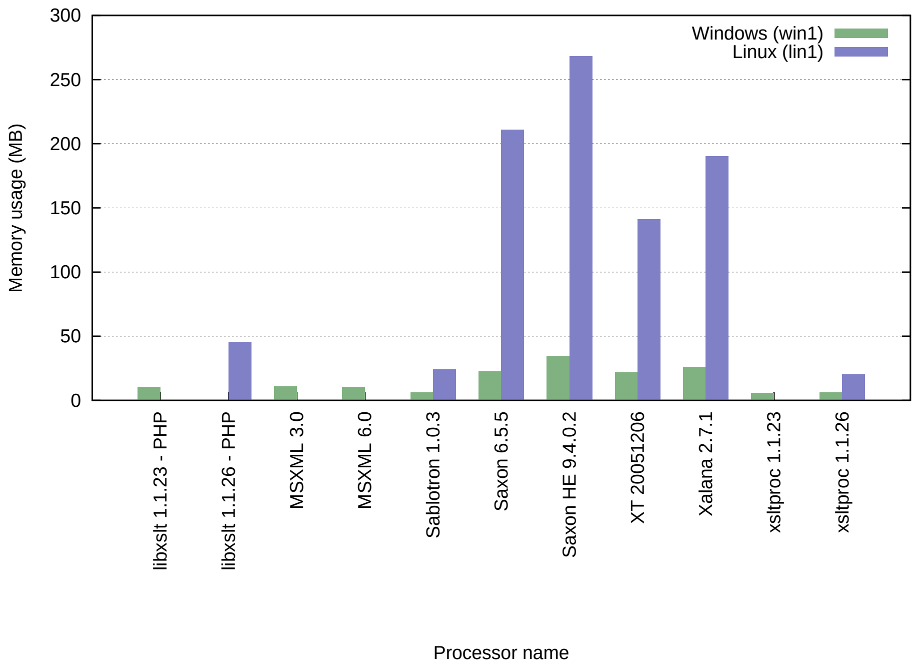
We summarize results from previous chapters for each processor in this chapter. We focus on interesting and important features of tested processors. The



Graph 11.1: **Average time usages for processors**: There are average time usages for all tested processors in the graph. Averages are showed separately for Windows and Linux. Not rendered boxes mean that the given processors were not tested for the given OS.

summary of all discussed features for all tested processors are in Tables 11.2 and 11.3. There is the list of features with their little explanations:

1. **Version** – The maximum supported XSLT version.
2. **Encoding** – The list of supported encodings. The flag “all” means all tested encodings.
3. **Speed** – The speed of a processor in verbal expression.
4. **Memory** – The memory usage of a processor in verbal expression.
5. **Bigger Inputs has Worse Speed** – A verbal expression for increase of the speed of a processor after increase of an input XML file.
6. **Indented XSLT is Better** – The flag, if an indented XSLT template is better than a non-indent XSLT template.



Graph 11.2: **Average memory usages for processors**: There are average memory usages for all tested processors in the graph. Averages are showed separately for Windows and Linux. Not rendered boxes mean that the given processors were not tested for the given OS.

7. **(Non-)Procedural for Better Speed** – The flag, if it is better procedural or non-procedural approach for speed of a processor.
8. **# Errors (from 43)** – The number of failed tests from all 43 tests.
9. **Java func.** – The flag of supporting of Java functions in templates.
10. **DocBook** – The flag of passing the test of the category DocBook.
11. **Namespace Aliases** – The flag of passing the tests on namespace aliasing.

| **Processor** | **Version** | **Encoding** | **Speed** | **Memory** | **Bigger Inputs has Worse Speed** | **Indented XSLT is Better** |
|---|---|---|---|---|---|---|
| **libxslt 1.1.23 - PHP** | 1.0 | all | fast | middle | very | no |
| **libxslt 1.1.26 - PHP** | 1.0 | all | fast | middle | very | no |
| **MSXML 3.0** | 1.0 | all | middle | middle | middle | no |
| **MSXML 6.0** | 1.0 | all | middle | middle | little | yes |
| **Sablotron 1.0.3** | 1.0 | all | middle | small | extremaly | no |
| **Saxon 6.5.5** | 1.0 | all | slow | big | little | no |
| **Saxon HE 9.4.0.2** | 2.0 | all | slow | big | little | no |
| **XT 20051206** | 1.0 | UTF-8 | slow | big | middle | yes |
| **Xalana 2.7.1** | 1.0 | all | slow | big | middle | no |
| **xsltproc 1.1.23** | 1.0 | all | fast | small | very | no |
| **xsltproc 1.1.26** | 1.0 | all | fast | small | very | no |

Table 11.2: **Summarize of features for processors - part 1**: The list of tested processors with their discussed features.

| **Processor** | **(Non-)Procedural for Better Speed** | **# Errors (from 43)** | **Java Func.** | **DocBook** | **Namespace Aliases** |
|---|---|---|---|---|---|
| **libxslt 1.1.23 - PHP** | same | 5 | no | no | no |
| **libxslt 1.1.26 - PHP** | same | 4 | no | yes | no |
| **MSXML 3.0** | non-proc | 4 | no | no | yes |
| **MSXML 6.0** | proc | 4 | no | no | yes |
| **Sablotron 1.0.3** | non-proc | 5 | no | no | no |
| **Saxon 6.5.5** | non-proc | 2 | yes | yes | no |
| **Saxon HE 9.4.0.2** | non-proc | 2 | no | yes | yes |
| **XT 20051206** | proc | 11 | no | yes | no |
| **Xalana 2.7.1** | non-proc | 4 | no | yes | no |
| **xsltproc 1.1.23** | same | 5 | no | no | no |
| **xsltproc 1.1.26** | same | 4 | no | yes | no |

Table 11.3: **Summarize of features for processors - part 2**: The list of tested processors with their discussed features.

In general, Java processors have the biggest time and memory usages. Conversely, command line processors are the fastest ones. Moreover, non-procedural access and non-indented XSLT templates have better time and memory usages than procedural access and indented XSLT templates. Next, all processors support only XSLT 1.0. Of course, exceptions exist and they will be mentioned for individual processors.

Processors with kernel libxslt have a problem with namespaces aliases. The advantage are good warnings about using of unsupported elements in XSLT template. Moreover, command line variants of xsltproc have better warnings than PHP variants. Version 1.1.26 is slower than 1.1.23 regardless the variant (command line or PHP library). On the other hand, version 1.1.26 is more reliable, version 1.1.23 failed in the test of the category "DocBook".

Processors MSXML 3.0 and MSXML 6.0 failed on the test of the category "DocBook". Processor MSXML 6.0 had better time usages for procedural access for some cases, which is interesting. Moreover, both processors have better time usages for indented XSLT templates, which is interesting too. Version 6.0 is faster than version 3.0 for big input files.

Processor Sablotron 1.0.3 failed on the test of category "DocBook". On the other hand, it has good reports of warnings and errors (report of unsupported used element, unsupported XSLT 2.0 by declaration etc.). A disadvantage is wrong support of namespace aliases and big slowdown with bigger input XML files.

Processors with kernel Saxon have the most passed tests. They failed only in 2 tests. Version 6.5.5 has better time also memory usages than version HE 9.4.0.2. Moreover, version 6.5.5 allows for using of Java functions in XSLT templates as only one processor from all tested. On the other hand, version HE 9.4.0.2 supports XSLT 2.0 as only one processor from all tested. In addition, version HE 9.4.0.2 is the least affected by bigger input XML files (from all tested processors) and has better warnings than version 6.5.5.

Processor XT 20051206 has the most failed tests (total 11). It supports only encoding UTF-8, does not support namespace aliases and using of Java functions in XSLT templates. In addition, it failed in the test of the category "Google Search Appliance". It has better time usage for procedural access in some cases, which is interesting.

Processor Xalan 2.7.1 does not support using of Java functions in XSLT templates and namespaces aliases. It is an average XSLT processor.

# Chapter 12

# Conclusions

The main purpose of this thesis was to create an XSLT benchmark, so we would be able to compare existing XSLT processors. The aim was to create flexible testing environment.

At first, we had to research existing XSLT benchmarks for possible inspiration. Unfortunately, there are not many XSLT benchmarks available on the Internet. Only XSLTMark is sufficient. However, this benchmark is from 2001 and it is not supported now. Only reports are available on the Internet. Thus, we could not use tests of XSLTMark in our benchmark and compare their results with XSLTMark results.

Next, we had to determine monitored criteria of XSLT processors. We determined *price*, *correctness*, *speed*, *memory usage*, *support* (timeliness, liveliness, availability, documentation), *OS* and *UX* (simplicity of installation, user friendliness, running in scripts). We also discussed types of XSLT processors before describing existing XSLT processors. Next, we described processors, which were being tested, in detail. We determined types not only for tested XSLT processors. In addition, we determined their criteria *price*, *support*, *OS* and *UX*.

We created a detailed analysis for criteria *correctness*, *speed* and *memory usage* and we created a test environment to make tests and to discuss their results. At first, we had to collect enough real XSLT templates for creating usable tests. We analyzed the collected files. The analysis had two main parts. The first part analyzes common features of XSLT templates. The second part analyzes categories of XSLT templates, thus focuses their usages.

One of the main benefits of our thesis was creating of the test environment. We created program XSLT Benchmarking. The program allows for generating of tests from templates of tests, running tests, generating XML reports, transforming reports into HTML format and testing different types of XSLT processors. In addition, it allows for many extensions. We can add other tests, templates of tests, tested processors and transformations of reports into other formats. Running of the program could be affected by many parameters. Nevertheless, only few parameters are sufficient for basic running. Thus, its using is very simple. Possibility of running it on different operating systems is a big advantage too. In addition, it is a command line program, thus it is possible to run it as a component of others scripts. We discussed possible extensions too. Thus, the program is freely available on [65] for possible upgrade.

We created tests based on analysis of collected XSLT templates in our program. Most tests were generated from our templates of tests. Smarty and ToXgene were used for generating some XSLT templates and XML files in the program. Of course, it is possible to add other generators. We created tests with synthetic XSLT templates, tests with XSLT templates based on collected files and tests with XSLT templates directly from collected files.

Much information and numbers in many tables were generated as results of tests. We discussed them from different points of view and discussed different features of XSLT processors. We discussed differences between operating systems, versions of some XSLT processors, average time and memory usages and many others. We cannot determine the best or the worst XSLT processor. Individual requirements of users of processors are very important and they could be very different. For example, only Saxon HE 9.4.0.2 supports XSLT 2.0, but it is very slow and uses very much memory. Next example, xsltproc, is the fastest and uses the least memory, but it works incorrectly with namespace aliases. The last example, MSXML, works with namespace aliases correctly, but it is supported only on Windows.

In brief, we created an XSLT benchmark, which is unique after long time. As a proof of the concept, we created the program XSLT Benchmark including tests for testing of XSLT processors. It is possible to add tests and tested processors into the program. We created summary of features of XSLT processors based on results of tests and discussed them.

# Chapter 13

# Future Works

The first possible extension of our system is an analysis of XPath expressions used in the collected XSLT files. It would by suitable to take into account elements, in which XPath is used, complexity of expressions, which functions are used and many others. New tests could be created based on this analysis.

The second possible extension is an analysis of other categories of collected XSLT files. Categories could be detected based on used namespaces in declarations of XSLT templates. Moreover, percentage usage of namespaces in templates could be also included in analysis.

In addition, other tested processors or additional tests and following extension of our results are interesting extensions too. It is possible to test Java processors running in different JVMs. It would be possible to detect the most appropriate JVMs for individual processors.

Additional tests could test recursive calling of named templates, more elements from XSLT 1.0, XSLT 2.0 or even XSLT 3.0. Other tests could be created based on further analysis of collected XSLT files. It would be possible to research the depth of nesting of XSLT elements "`if`" and "`choose`" together. Next, it would be possible to detect not using of elements from XSLT 2.0 in XSLT file that declare of using of XSLT 2.0. Or even, it would be possible to detect using of elements from XSLT 2.0 in XSLT file that do not declare of using of XSLT 2.0.

A nontrivial problem is taking into the account including of other templates into main template by XSLT elements "`include`" or "`import`".

Our program XSLT Benchmarking could be also extended. Most possible extensions are described in Chapter 9.5. Let us enumerate the most interesting: It is possible to add XSLT and XML generator. Input parameters of the program could be set by configuration file instead of console parameters. A large extension would be to enable to use more expected outputs of transformations. Alternatively, the extension could be to enable termination of transformations by errors and checking of content of errors.

# Bibliography

[1] I. Mlýnková: *XML Benchmarking: Limitations and Opportunities (Technical Report)*, Charles University 2008. `http://www.ksi.mff.cuni.cz/~mlynkova/doc/tr2008-1.pdf`.

[2] J. Clark: *XSL Transformations (XSLT) Version 1.0.* W3C, November 1999. `http://www.w3.org/TR/xslt`.

[3] T. Bray, J. Paoli et al.: *Extensible Markup Language (XML) 1.0 (Fifth Edition).* W3C, November 2008. `http://www.w3.org/TR/xml/`.

[4] I. Mlýnková, M. Nečaský, J. Pokorný, K. Richta, K. Toman, V. Toman: *XML Technologie, Principy a aplikace v praxi.* Grada Publishing, Prague, Czech republic, September 2008. ISBN 978-80-247-2725-7.

[5] D. Raggett, A. Le Hors and I. Jacobs: *HTML 4.01 Specification.* W3C, December 1999. `http://www.w3.org/TR/html401/`.

[6] M. H. Kay: *Saxon.* Saxonica Limited, December 2011. `http://saxon.sourceforge.net/`.

[7] J. Fleck: *xsltproc - command line XSLT processor.* 2002. `http://xmlsoft.org/XSLT/xsltproc.html`.

[8] J. Clark and S. DeRose: *XML Path Language (XPath) Version 1.0.* W3C, November 1999. `http://www.w3.org/TR/xpath/`.

[9] M. Kay: *XSL Transformations (XSLT) Version 2.0.* W3C, January 2007. `http://www.w3.org/TR/xslt20/`.

[10] A. Berglund, S. Boag et al.: *XML Path Language (XPath) 2.0.* W3C, January 2007. `http://www.w3.org/TR/xpath20/`.

[11] H. Zongaro: *XSLT and XQuery Serialization 3.0*. W3C, December 2011. http://www.w3.org/TR/xslt-xquery-serialization-30/.

[12] *Xalan*. The Apache Software Foundation. http://xalan.apache.org.

[13] J. Clark: *XT, Version 20051206*. December 2005. http://www.blnz.com/xt/xt-20051206/.

[14] D. Veillard: *libxslt - The XSLT C library for GNOME*. 2009. http://xmlsoft.org/XSLT/.

[15] *MSXML*. Microsoft Corporation. http://msdn.microsoft.com/en-en/data/bb190600.aspx.

[16] P. Hlavnicka, P. Cimprich, P. Gühring: *Sablotron*. February 2010. http://www.gingerall.com/charlie/ga/xml/p_sab.xml.

[17] J. Kloth, M. Brown, M. Olson, U. Ogbuji: *4Suite*. December 2006. http://foursuite.sourceforge.net/.

[18] *TransforMiiX - XSLT Processing Engine*. Mozilla Developer Network, 2005. https://developer.mozilla.org/en/Using_the_Mozilla_JavaScript_interface_to_XSL_Transformations.

[19] J. Burkard: *xslt.js version 3.2*. August 2008. http://johannburkard.de/software/xsltjs/.

[20] S. Meschkat, D. Fabulich, H.-B. Chai: *ajaxslt, version 0.8.1*. January 2008. http://code.google.com/p/ajaxslt/.

[21] *Unicorn XSLT Processor*. Unicorn Enterprises SA, 2001. http://www.unicorn-enterprises.com/products_uxt.html.

[22] *AltovaXML - XSLT 1.0/2.0 Engine, XQuery Engine, XML Validator*. Altova, 2012. http://www.altova.com/altovaxml.html.

[23] *XmlPrime*. Clinical  Biomedical Computing Ltd., 2012. http://www.xmlprime.com/xmlprime/download.htm.

[24] J. Clark: *XP - an XML Parser in Java, Version 0.5*. 1998. http://www.jclark.com/xml/xp/.

[25] D. Megginson: *SAX, Simple API for XML, Version 1.0.* http://www.saxproject.org/sax1-roadmap.html.

[26] D. Veillard: *libxml2 - The XML C parser and toolkit of Gnome.* November 2010. http://xmlsoft.org/.

[27] U. Drepper: *iconv, Version 1.9.2.* 2010. http://www.gnu.org/software/libiconv/.

[28] J.-L. Gailly and M. Adler: *zlib, Version 1.2.5.* March 2010. http://www.zlib.net/.

[29] *Microsoft Windows.* Microsoft Corporation, July 2009. http://windows.microsoft.com.

[30] *Windows Internet Explorer.* Microsoft Corporation, 2001. http://windows.microsoft.com/en-US/internet-explorer/products/ie/home.

[31] *cscript.* Microsoft Corporation. http://technet.microsoft.com/en-us/library/bb490887.aspx.

[32] J. Clark: *Expat - The Expat XML Parser.* June 2007. http://expat.sourceforge.net/.

[33] E. Kuznetsov, C. Dolph: *XSLTMark, XSLT Processor Benchmarks.* http://www.xml.com/pub/a/2001/03/28/xsltmark/index.html. March 2001.

[34] Caucho Technology, Inc.: *Caucho, XSLT Benchmark.* 2001. http://www.caucho.com/resin-3.0/features/xslt-benchmark.xtp.

[35] D. Parshley: *David Parshley, XSLT Benchmarks.* 2005. http://www.davidpashley.com/articles/xslt-benchmarks.html.

[36] J. Chamas: *Hello World.* April 2003. http://www.chamas.com/bench/.

[37] The Apache Software Foundation: *Apache - HTTP server project.* 2011. http://httpd.apache.org/.

[38] D. Fancellu: *XSLT is Way Faster Using Java 5.* http://www.softwarereality.com/programming/java5xslt_speedup.jsp. February 2005.

[39] H. Niksic, M. Cowan: *Wget - The non-interactive network downloader.* GNU Wget version 1.11.4. http://www.gnu.org/software/wget/.

[40] Google Inc.: *Google.* http://www.google.com.

[41] N. Blachman: *Google Guide.* http://www.googleguide.com/advanced_operators.html.

[42] A. ITO: *w3m - a text based Web browser and pager.* Version w3m/0.5.2. http://w3m.sourceforge.net/.

[43] Google Inc.: *Google Code.* http://code.google.com.

[44] J. Fleck, Z. Sherwin, H. Rupp: *xmllint - command line XML tool.* Using libxml version 20705. http://xmlsoft.org/xmllint.html.

[45] P. Eggert, M. Haertel, D. Hayes, R. Stallman and L. Tower: *diff - compare files line by line.* GNU diffutils version 2.8.1. http://www.gnu.org/software/diffutils/.

[46] The PHP Group: *PHP: Hypertext Preprocessor - Command Line Interface.* Version 5.2.10-2ubuntu6.9 with Suhosin-Patch.7 (cli). http://www.php.net/.

[47] D. Megginson: *SAX - Simple API for XML.* http://www.saxproject.org/.

[48] T. Granlund, D. MacKenzie, P. Eggert and J. Meyering: *du - estimate file space usage.* Version 7.4.

[49] RSS Advisory Board: *RSS 2.0 Specification.* March 30, 2009. http://www.rssboard.org/rss-specification.

[50] Google Inc.: *Google Search Appliance.* Version 6.12, August 2011. http://www.google.com/enterprise/search/gsa.html.

[51] Google Inc.: *XSL to format the search output for Google Search Appliance*. May 2008.
http://code.google.com/p/gsa-xhtml-stylesheet/source/browse/trunk/gsa-xhtml.en.xslt.

[52] GraphML Team: *The GraphML File Format*. April 2007.
http://graphml.graphdrawing.org/.

[53] GraphML Team: *The GraphML File Format - Download*. April 2007.
http://graphml.graphdrawing.org/download.html.

[54] W. Li, P. Eades, N. Nikolov: *Using Spring Algorithms to Remove Node Overlapping*. National ICT Australia Ltd, University of Sydney NSW, 2006, Australia

[55] O. Andersson, P. Armstrong, H. Axelsson et al.: *Scalable Vector Graphics (SVG) 1.1 Specification*. W3C, April 2009.
http://www.w3.org/TR/2003/REC-SVG11-20030114/.

[56] J. Punin, M. Krishnamoorthy: *XGMML (eXtensible Graph Markup and Modeling Language) - XGMML 1.0 Draft Specification*. August 2001.
http://www.cs.rpi.edu/research/groups/pb/punin/public_html/XGMML/draft-xgmml.html.

[57] M. Himsolt: *GML: A portable Graph File Format*. Universität Passau, 94030 Passau, Germany.
http://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf.

[58] J. Punin, M. Krishnamoorthy, G. Uffelman: *LOGML (Log Markup Language) - LOGML 1.0 Draft Specification*. August 2001.
http://www.cs.rpi.edu/research/groups/pb/punin/public_html/LOGML/draft-logml.html.

[59] J. Punin, M. Krishnamoorthy: *XGMML (eXtensible Graph Markup and Modeling Language) - XGMML 1.0 Draft Specification*. August 2001.
http://www.cs.rpi.edu/research/groups/pb/punin/public_html/XGMML/XSL/XSL_BL/.

[60] N. Walsh: *The DocBook Schema Version 5.0*. OASIS, March 2008.
http://www.docbook.org/specs/docbook-5.0-spec-cd-03.html.

[61] N. Walsh, J. Kosek, S. Ball: *Prearranged templates for DocBook, version 1.76.1.* The DocBook Project, November 2010.
http://docbook.sourceforge.net/release/xsl/1.76.1/.

[62] F. Manola, E. Miller: *RDF Primer.* W3C, February 2004.
http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

[63] D. Brickley, R.V. Guha, B. McBride: *RDF Vocabulary Description Language 1.0: RDF Schema.* W3C, February 2004.
http://www.w3.org/TR/rdf-schema/.

[64] J. Punin, M. Krishnamoorthy: *RDF Graph Modeling Language (RGML) - RGML 1.0 Draft Specification.* August 2001.
http://www.cs.rpi.edu/research/groups/pb/punin/public_html/RGML/draft-rgml.html.

[65] V. Mašíček: *XSLT Benchmarking, version 1.0.0.* April 2012.
http://xsltbenchmarking.masicek.net/.

[66] J. Gruber: *Markdown.* December 2004.
http://daringfireball.net/projects/markdown/.

[67] V. Mašíček: *XSLT Benchmarking, version 1.0.0 - README.markdown.* April 2012.
https://github.com/masicek/XSLT-Benchmarking/blob/master/README.markdown.

[68] V. Mašíček: *XSLT Benchmarking, veriosn 1.0.0 - API documentation.* April 2012. http://xsltbenchmarking.masicek.net/api/.

[69] V. Mašíček: *PhpOptions, version 1.3.0.* February 2012.
http://phpoptions.masicek.net/.

[70] M. Ohrt, U. Tews, R. Rehm: *Smarty - template engine, version 3.1.4.* New Digital Group, Inc., October 2011. http://www.smarty.net/.

[71] D. Barbosa, A. Mendelzon, J. Keenleyside: *ToXgene - the ToX XML Data Generator - version 2.3.* University of Toronto, February 2005.
http://www.cs.toronto.edu/tox/toxgene/.

[72] H. J. Jeong, S. H. Lee.: *A Versatile XML Data Generator.* International Journal of Software Effectiveness and Efciency, 1(1):21–24, 2006.

[73] L. Afanasiev, I. Manolescu, P. Michiels: *MemBeR XML Generator.* `http://ilps.science.uva.nl/Resources/MemBeR/member-generator.html`.

[74] A. Aboulnaga, J. F. Naughton, C. Zhang: *Generating Synthetic Complex-Structured XML Data.* In WebDB'01: Proc. of the 4th Int. Workshop on the Web and Databases, pages 79–84, Washington, DC, USA, 2001

[75] D. MacKenzie: *time - time a simple command or give resource usage.* December 1999. `http://www.gnu.org/software/time/`.

[76] A. Fingerhut: *clojure-benchmarks.* July, 2009. `https://github.com/jafingerhut/clojure-benchmarks/blob/master/bin/timemem.exe`.

[77] J. Dvořáková, F. Zavoral: *Using Input Buffers for Streaming XSLT Processing.* 2009. `http://www.computer.org/portal/web/csdl/doi/10.1109/DBKDA.2009.25`.

[78] S. Bergmann: *PHPUnit.* April 2012. `https://github.com/sebastianbergmann/phpunit/`.

[79] T. Burkholder: *gsa-parser, 1.xml.* February 2011. `http://code.google.com/p/gsa-parser/source/browse/GSA/xslt/1.xml?r=2`.

[80] V. Mašíček: *Google Docs - XSLT Benchmarking - Additional Reports.* April 2012. `http://code.google.com/p/gsa-parser/source/browse/GSA/xslt/1.xml?r=2`.

# Appendix A

# Content of CD

The description of content of attached CD.

1. /CollectingData/ – scripts for download and analysis of XSLT documents
   /CollectingData/download/ – scripts for download of XSLT documents from the Internet
   /CollectingData/download/addresses_search – the list of addresses of XSLT files found on Google Search
   /CollectingData/download/addresses_googlecode – the list of addresses found on Google Code used as seeds
   /CollectingData/download/addresses_hand – the list of addresses created manually and used as seeds
   /CollectingData/download/addresses_nonrecursive – the list of not downloaded addresses found in the log file generated during downloading
   /CollectingData/clean/ – scripts for merging and cleaning XSLT documents downloaded from the Internet

2. /AnalysisData/ – scripts for analysing of downloaded XSLT documents
   /AnalysisData/scan/ – scripts for scan of content of downloaded XSLT documents
   /AnalysisData/scan/scan.php – main script that scans all input files
   /AnalysisData/category/ – scripts for detection of categories of downloaded XSLT documents

`/AnalysisData/category/individual/common` – parameterization scripts for detection of properties of caterories of downloaded XSLT documents
`/AnalysisData/reports/` – scripts generating graps and tables (in the CVS format and in the TeX format) from reported data that show results of analysing of downloaded XSLT documents

3. `/XSLT-Benchmarking/` – our project for generating, runnig and reporting XSLT Benchmarking
   `/XSLT-Benchmarking/Data/Tests/` – prepared tests
   `/XSLT-Benchmarking/Data/TestsTemplates/` – prepared templates of tests for generating

4. `/API/` – the API documentation of the program XSLT Benchmarking

5. `/Reports/` – results of our tests on different machines

6. `/XSLT-Benchmarking.pdf` – this thesis

7. `/README.markdown` – the Markdown description of the CD
   `/README.html` – the HTML description of the CD generated form README.markdown

# Appendix B

# Scan Data - Features

This appendix includes tables with exactly measured values of scanned features of collected XSLT fiels. Graphs and tables in Chapter 8.1 are created based on these tables. There are list of tables:

1. Table B.1: **Maximum depth of files**
2. Table B.2: **Depth of nesting of selected elements**
3. Table B.3: **Maximum fan-out**
4. Table B.4: **Number of elements rate**
5. Table B.5: **Size of files**
6. Table B.6: **Number of recursions**
7. Table B.7: **Lengths of the longest recursion**

| **Maximal depth** | **# files** |
|---:|---:|
| 0 | 140 |
| 1 | 215 |
| 2 | 240 |
| 3 | 382 |
| 4 | 397 |
| 5 | 1 163 |
| 6 | 560 |
| 7 | 540 |
| 8 | 422 |
| 9 | 478 |
| 10 | 404 |
| 11 | 213 |
| 12 | 173 |
| 13 | 118 |
| 14 | 72 |
| 15 | 64 |
| 16 | 32 |
| 17 | 20 |
| 18 | 14 |
| 19 | 39 |
| 20 | 22 |
| 21 | 18 |
| 22 | 8 |
| 23 | 7 |
| 24 | 5 |
| 25 | 4 |
| 26 | 2 |
| 27 | 7 |
| 28 | 3 |
| 29 | 1 |
| 30 | 1 |
| 32 | 1 |
| 33 | 3 |
| 60 | 1 |

Table B.1: **Maximum depth of files**: In the table there are measured maximum depths of scanned files and numbers of files with the relevant maximum depths.

| **Depth of nesting** | **Foreach** | **Choose** | **If** |
|---:|---:|---:|---:|
| 0 | 3 639 | 3 147 | 2 795 |
| 1 | 1 594 | 1 608 | 1 984 |
| 2 | 381 | 713 | 756 |
| 3 | 86 | 220 | 185 |
| 4 | 46 | 50 | 39 |
| 5 | 7 | 26 | 8 |
| 6 | 3 | 3 | 0 |
| 7 | 4 | 1 | 2 |
| 8 | 2 | 0 | 0 |
| 9 | 1 | 1 | 0 |
| 10 | 2 | 0 | 0 |
| 11 | 1 | 0 | 0 |
| 12 | 0 | 0 | 0 |
| 13 | 2 | 0 | 0 |
| 14 | 0 | 0 | 0 |
| 15 | 0 | 0 | 0 |
| 16 | 0 | 0 | 0 |
| 17 | 0 | 0 | 0 |
| 18 | 0 | 0 | 0 |
| 19 | 0 | 0 | 0 |
| 20 | 0 | 0 | 0 |
| 21 | 0 | 0 | 0 |
| 22 | 0 | 0 | 0 |
| 23 | 0 | 0 | 0 |
| 24 | 0 | 0 | 0 |
| 25 | 0 | 0 | 0 |
| 26 | 0 | 0 | 0 |
| 27 | 1 | 0 | 0 |

Table B.2: **Depth of nesting of selected elements**: In the table there are measured depth of nesting of scanned files and numbers of files with the relevant depths of nesting separately for selected elements "`for-each`", "`choose`" and "`if`".

| Fan-out | # files |
|---:|---:|
| 0 | 111 |
| 1 | 67 |
| 2 | 283 |
| 3 | 657 |
| 4 | 619 |
| 5 | 435 |
| 6 | 371 |
| 7 | 261 |
| 8 | 252 |
| 9 | 228 |
| 10 | 252 |
| 11 | 125 |
| 12 | 149 |
| 13 | 148 |
| 14 | 111 |
| 15 | 102 |
| 16 | 114 |
| 17 | 90 |
| 18 | 74 |
| 19 | 62 |
| 20 | 64 |
| 21 | 61 |
| 22 | 43 |
| 23 | 55 |
| 24 | 67 |
| 25 | 40 |
| 26 | 58 |
| 27 | 80 |
| 28 | 51 |
| 29 | 37 |
| 30 | 13 |
| 31 | 21 |
| 32 | 23 |
| 33 | 16 |
| 34 | 18 |
| 35 | 12 |
| 36 | 15 |
| 37 | 14 |
| 38 | 19 |
| 39 | 27 |
| 40 | 15 |
| 41 | 9 |
| 42 | 10 |
| 43 | 20 |
| 44 | 9 |
| 45 | 10 |
| 46 | 2 |
| 47 | 4 |
| 48 | 6 |
| 49 | 8 |
| 50 | 19 |
| 51 | 12 |
| 52 | 11 |
| 53 | 14 |
| 54 | 12 |
| 55 | 4 |
| 56 | 4 |
| 57 | 7 |
| 58 | 3 |
| 59 | 9 |
| 60 | 5 |
| 61 | 4 |
| 62 | 12 |
| 63 | 8 |
| 64 | 7 |
| 65 | 4 |
| 66 | 2 |
| 67 | 6 |
| 68 | 4 |
| 69 | 1 |
| 70 | 2 |
| 71 | 8 |
| 72 | 13 |
| 73 | 10 |
| 74 | 2 |
| 75 | 1 |
| 76 | 12 |
| 77 | 2 |
| 78 | 3 |
| 79 | 2 |
| 80 | 5 |
| 83 | 1 |
| 84 | 4 |
| 85 | 1 |
| 86 | 2 |
| 87 | 1 |
| 89 | 1 |
| 90 | 4 |
| 91 | 2 |
| 93 | 2 |
| 96 | 4 |
| 97 | 2 |
| 98 | 2 |
| 99 | 3 |
| 101 | 1 |
| 103 | 4 |
| 104 | 1 |
| 105 | 1 |
| 107 | 1 |
| 108 | 1 |
| 110 | 6 |
| 112 | 6 |
| 113 | 2 |
| 114 | 2 |
| 115 | 2 |
| 117 | 3 |
| 119 | 1 |
| 120 | 7 |
| 121 | 2 |
| 122 | 1 |
| 123 | 1 |
| 124 | 3 |
| 125 | 1 |
| 129 | 3 |
| 132 | 2 |
| 133 | 2 |
| 134 | 8 |
| 135 | 1 |
| 136 | 1 |
| 137 | 2 |
| 138 | 5 |
| 139 | 4 |
| 140 | 5 |
| 141 | 4 |
| 143 | 1 |
| 144 | 2 |
| 146 | 3 |
| 150 | 2 |
| 153 | 1 |
| 154 | 5 |
| 155 | 2 |
| 157 | 1 |
| 159 | 1 |
| 160 | 1 |
| 161 | 1 |
| 165 | 1 |
| 166 | 4 |
| 169 | 3 |
| 172 | 1 |
| 173 | 1 |
| 174 | 1 |
| 175 | 2 |
| 176 | 2 |
| 178 | 1 |
| 179 | 1 |
| 180 | 2 |
| 181 | 1 |
| 182 | 1 |
| 183 | 1 |
| 184 | 4 |
| 185 | 3 |
| 186 | 8 |
| 187 | 2 |
| 190 | 2 |
| 193 | 1 |
| 196 | 2 |
| 200 | 1 |
| 202 | 1 |
| 204 | 1 |
| 205 | 2 |
| 219 | 1 |
| 223 | 1 |
| 233 | 2 |
| 252 | 1 |
| 256 | 1 |
| 260 | 1 |
| 261 | 1 |
| 275 | 2 |
| 276 | 3 |
| 281 | 1 |
| 290 | 1 |
| 303 | 2 |
| 308 | 2 |
| 309 | 1 |
| 312 | 2 |
| 325 | 1 |
| 326 | 1 |
| 343 | 1 |
| 391 | 1 |
| 397 | 2 |
| 407 | 2 |
| 451 | 1 |
| 515 | 2 |
| 550 | 2 |
| 583 | 1 |
| 615 | 1 |
| 779 | 1 |
| 813 | 1 |
| 1 821 | 2 |
| 6 652 | 1 |
|  |  |

Table B.3: **Maximum fan-out**: In the table there are numbers of scanned files with their relevant measured maximum fan-out.

| **Element name** | **XSLT** | **# rate** |
|---:|---:|---:|
| value-of | 1.0 | 100 500 |
| text | 1.0 | 74 029 |
| apply-templates | 1.0 | 61 755 |
| when | 1.0 | 60 017 |
| template | 1.0 | 59 280 |
| with-param | 1.0 | 48 814 |
| if | 1.0 | 43 182 |
| call-template | 1.0 | 42 809 |
| attribute | 1.0 | 39 846 |
| variable | 1.0 | 35 906 |
| choose | 1.0 | 26 843 |
| for-each | 1.0 | 26 361 |
| param | 1.0 | 25 205 |
| otherwise | 1.0 | 23 962 |
| copy-of | 1.0 | 8 364 |
| element | 1.0 | 7 784 |
| output-character | 2.0 | 6 678 |
| stylesheet | 1.0 | 5 447 |
| output | 1.0 | 3 565 |
| message | 1.0 | 2 170 |
| attribute-set | 1.0 | 2 106 |
| include | 1.0 | 1 979 |
| number | 1.0 | 1 838 |
| sort | 1.0 | 1 825 |
| copy | 1.0 | 1 476 |
| import | 1.0 | 1 087 |
| comment | 1.0 | 895 |
| key | 1.0 | 555 |
| strip-space | 1.0 | 357 |
| function | 2.0 | 342 |
| apply-imports | 1.0 | 215 |
| transform | 1.0 | 188 |
| eval |  | 183 |
| processing-instruction | 1.0 | 172 |
| result-document | 2.0 | 150 |
| sequence | 2.0 | 131 |
| preserve-space | 1.0 | 98 |
| character-map | 2.0 | 89 |
| decimal-format | 1.0 | 80 |
| for-each-group | 2.0 | 79 |
| namespace | 2.0 | 52 |
| namespace-alias | 1.0 | 49 |
| matching-substring | 2.0 | 40 |
| analyze-string | 2.0 | 40 |
| document | 2.0 | 38 |
| next-match | 2.0 | 37 |
| script |  | 13 |
| node-name |  | 13 |
| non-matching-substring | 2.0 | 12 |
| entity-ref |  | 10 |
| fallback | 1.0 | 6 |
| loop |  | 3 |
| import-schema | 2.0 | 3 |
| perform-sort | 2.0 | 1 |
| elsif |  | 1 |
| else |  | 1 |
|  |  |  |

Table B.4: **Number of elements rate**: In the table there are all found elements in scanned files from namespace “`xsl`” with their rates and XSLT versions.

| **Size (kB)** | **# files** |
|---:|---:|
| 0-10 | 4 308 |
| 10-20 | 723 |
| 20-30 | 256 |
| 30-40 | 138 |
| 40-50 | 107 |
| 50-60 | 57 |
| 60-70 | 28 |
| 70-80 | 36 |
| 80-90 | 31 |
| 90-100 | 17 |
| 100-110 | 8 |
| 110-120 | 16 |
| 120-130 | 8 |
| 130-140 | 6 |
| 140-150 | 8 |
| 150-160 | 5 |
| 160-170 | 2 |
| 170-180 | 10 |
| 180-190 | 2 |
| 190-200 | 1 |
| 200-210 | 1 |
| 210-220 | 2 |
| 220-230 | 1 |
| 230-240 | 2 |
| 240-250 | 2 |
| 250-260 | 3 |
| 260-270 | 2 |
| 270-280 | 1 |
| 450-460 | 1 |
| 510-520 | 1 |
| 540-550 | 1 |
| 760-770 | 1 |
| 2 200-2 210 | 1 |
| 7 830-7 840 | 1 |

Table B.5: **Size of files**: In the table are numbers of scanned files with their relevant measured sizes. Sizes of files are grouped for better visualization.

| # recursions | # files |
|---:|---:|
| 0 | 5 058 |
| 1 | 414 |
| 2 | 121 |
| 3 | 72 |
| 4 | 26 |
| 5 | 32 |
| 6 | 11 |
| 7 | 16 |
| 8 | 5 |
| 9 | 1 |
| 10 | 1 |
| 11 | 3 |
| 13 | 3 |
| 14 | 2 |
| 15 | 2 |
| 22 | 1 |
| 8 799 | 1 |

Table B.6: **Number of recursions**: In the table there are numbers of scanned files with their relevant measured numbers of recursions.

| Max way of recursions | # files |
|---:|---:|
| 0 | 5 058 |
| 1 | 677 |
| 2 | 25 |
| 3 | 3 |
| 4 | 3 |
| 6 | 1 |
| 9 | 1 |
| 18 | 1 |

Table B.7: **Lengths of the longest recursion**: In the table there are numbers of scanned files with their relevant measured lengths of longest recursion cycle.

# Appendix C

# Scan Data - Criteria for Categories

This appendix includes tables with lists of criteria for each detected category in collected XSLT files. Categories are disccussed in Chapter 8.2. There are list of tables:

1. Table C.1: **Criteria for category "RSS reader"**
2. Table C.2: **Criteria for category "RSS generator"**
3. Table C.3: **Criteria for category "Google Search Appliance"**
4. Table C.4: **Criteria for category "GraphML"**
5. Table C.5: **Criteria for category "XGMML"**
6. Table C.6: **Criteria for category "LOGML"**
7. Table C.7: **Criteria for category "DocBook reader"**
8. Table C.8: **Criteria for category "DocBook generator"**
9. Table C.9: **Criteria for category "RDF reader"**
10. Table C.10: **Criteria for category "RDF generator"**
11. Table C.11: **Criteria for category "RDFS reader"**
12. Table C.12: **Criteria for category "RDFS generator"**

13. Table C.13: **Criteria for category "RGML reader"**

14. Table C.14: **Criteria for category "RGML generator"**

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| Template match substring | rss | 10 |
| Template match substring | rdf | 10 |
| Template match substring | channel | 3 |
| Template match substring | item | 1 |
| File name substring | rss | 8 |
| File name | navlist.xslt | 13 |
| File name | newsfeeds.xslt | 13 |

Table C.1: **Criteria for category "RSS reader"**: The list of criteria with their input values and weights for category "RSS reader"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| File name substring | rss | 1 |
| Element name | rss | 10 |
| File substring | ¡rss | 10 |
| Element name | rdf | 10 |
| File substring | <rdf:rdf | 10 |
| Element name | channel | 3 |
| File substring | <channel | 3 |
| Element name | item | 1 |
| File substring | <item | 1 |

Table C.2: **Criteria for category "RSS generator"**: The list of criteria with their input values and weights for category "RSS generator"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| Name of variable | *10 different names* | 1 |
| File substring | XSL to format the search output for Google Search Appliance | 10 |
| File substring | Google Search Appliance | 5 |

Table C.3: **Criteria for category "Google Search Appliance"**: The list of criteria with their input values and weights for category "Google Search Appliance"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| File name substring | graphml | 8 |
| File substring | graphml | 3 |
| Template name substring | generatenodes | 2 |
| Template name substring | generateedges | 2 |
| Template match substring | graphml | 8 |
| Template match substring | graph | 1 |
| Template match substring | edge | 1 |

Table C.4: **Criteria for category "GraphML"**: The list of criteria with their input values and weights for category "GraphML"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| File name substring | xgmml | 8 |
| File substring | xgmml | 5 |
| Template match substring | graph | 5 |
| Template match substring | node | 3 |
| Template match substring | edge | 3 |
| Template match substring | graphic | 1 |
| Template match substring | att | 1 |

Table C.5: **Criteria for category "XGMML"**: The list of criteria with their input values and weights for category "XGMML"

| Criterion | Value | Weight |
|---|---:|---:|
| File name substring | logml | 12 |
| File substring | logml | 4 |
| Template match substring | logml | 8 |
| Template name substring | logml | 4 |
| Template match substring | hosts | 1 |
| Template match substring | domains | 1 |
| Template match substring | directories | 1 |
| Template match substring | userAgents | 1 |
| Template match substring | referers | 1 |
| Template match substring | keywords | 1 |

Table C.6: **Criteria for category "LOGML"**: The list of criteria with their input values and weights for category "LOGML"

| Criterion | Value | Weight |
|---|---:|---:|
| File name substring | docbook | 5 |
| File name substring | docbook.sourceforge.net | 6 |
| File substring | docbook.dtd | 9 |
| File substring | This file is part of the XSL DocBook Stylesheet distribution | 11 |
| Template match substring | chapter | 1 |
| Template match substring | abstract | 1 |
| Template match substring | section | 1 |
| Template match substring | bibliography | 1 |
| Template match substring | footnote | 1 |

Table C.7: **Criteria for category "DocBook reader"**: The list of criteria with their input values and weights for category "DocBook reader"

| Criterion | Value | Weight |
|---|---:|---:|
| Element name | chapter | 1 |
| File substring | <chapter | 1 |
| Element name | abstract | 1 |
| File substring | <abstract | 1 |
| Element name | section | 1 |
| File substring | <section | 1 |
| Element name | bibliography | 1 |
| File substring | <bibliography | 1 |
| Element name | footnote | 1 |
| File substring | <footnote | 1 |
| Element name | literallayout | 1 |
| File substring | <literallayout | 1 |
| Element name | editor | 1 |
| File substring | <editor | 1 |
| Element name | bibliosource | 1 |
| File substring | <bibliosource | 1 |
| Element name | acronym | 1 |
| File substring | <acronym | 1 |
| Element name | footnoteref | 1 |
| File substring | <footnoteref | 1 |
| Element name | sect1 | 1 |
| File substring | <sect1 | 1 |
| Element name | sect2 | 1 |
| File substring | <sect2 | 1 |
| Element name | sect3 | 1 |
| File substring | <sect3 | 1 |
| Element name | sect4 | 1 |
| File substring | <sect4 | 1 |
| Element name | sect5 | 1 |
| File substring | <sect5 | 1 |
| Element name | glossary | 1 |
| File substring | <glossary | 1 |
| Element name | glosslist | 1 |
| File substring | <glosslist | 1 |

Table C.8: **Criteria for category "DocBook generator"**: The list of criteria with their input values and weights for category "DocBook generator"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| File name substring | rdf | 1 |
| Template match substring | rdf | 5 |
| Template match substring | description | 5 |
| Template match substring | bag | 2 |
| Template match substring | seq | 2 |
| Template match substring | alt | 2 |

Table C.9: **Criteria for category "RDF reader"**: The list of criteria with their input values and weights for category "RDF reader"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| Element name | rdf | 10 |
| File substring | <rdf:rdf | 10 |
| Element name | Description | 4 |
| File substring | <rdf:Description | 4 |

Table C.10: **Criteria for category "RDF generator"**: The list of criteria with their input values and weights for category "RDF generator"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| File name substring | rdfs | 1 |
| Template match substring | rdf | 1 |
| Template match substring | rdfs | 2 |
| Template match substring | description | 5 |
| Template match substring | Class | 3 |
| Template match substring | subClassOf | 3 |
| Template match substring | bag | 2 |
| Template match substring | seq | 2 |
| Template match substring | alt | 2 |

Table C.11: **Criteria for category "RDFS reader"**: The list of criteria with their input values and weights for category "RDFS reader"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| Element name | rdf | 10 |
| File substring | <rdf:rdf | 10 |
| Element name | Class | 4 |
| File substring | <rdfs:Class | 4 |
| Element name | subClassOf | 2 |
| File substring | <rdfs:subClassOf | 2 |
| File substring | http://www.w3.org/2000/01/rdf-schema#Class | 2 |

Table C.12: **Criteria for category "RDFS generator"**: The list of criteria with their input values and weights for category "RDFS generator"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| File name substring | rgml | 8 |
| File substring | rgml | 1 |
| Template match substring | rdf: | 1 |
| Template match substring | rdf:seq | 2 |
| Template match substring | rdf:bag | 2 |

Table C.13: **Criteria for category "RGML reader"**: The list of criteria with their input values and weights for category "RGML reader"

| **Criterion** | **Value** | **Weight** |
|---|---:|---:|
| File name substring | rgml | 1 |
| File substring | rgml | 1 |
| Element name | bag | 2 |
| File substring | <rdf:bag | 2 |
| Element name | seq | 2 |
| File substring | <rdf:seq | 2 |

Table C.14: **Criteria for category "RGML generator"**: The list of criteria with their input values and weights for category "RGML generator"

# Appendix D

# Test Environment - Description of Parameters

There is the list of available parameter of the program XSLT Benchmarking.

1. `-g` or `--generate`
   Generates all tests from tests templates placed in the directory set by the option `--templates` and save them into the directory set by the option `--tests`.

2. `-r` or `--run`
   Run all prepared tests in the directory set by the option `--tests` and generate XML reports into the directory set by the option `--reports`.

3. `--templates`
   Set the directory containing tests templates for generating tests. The default value is "`../Data/TestsTemplates`".

4. `--templates-dirs`
   Set the subdirectories of directory set by the option `--templates` containing tests templates for generating, separated by the character ",". If this option is not set (or is set without value), then all tests templates are selected (all subdirectories that are considered as tests templates).

5. `--tests`
   Set the directory containing tests. Generated tests are generated into this directory. Running tests are get from this directory. The default value is "`../Data/Tests`"

6. `--tests-dirs`
   Set the subdirectories of directory set by the option `--tests` containing tests for runnig, separated by character ",". If this option is not set (or is set without value), then all tests are selected (all subdirectories that are considered as tests).

7. `--tmp`
   Set the temporary derectory. This directory is used for generating temporary files and for saving files generate by XSLT tranformations. The default value is "`../Tmp`".

8. `--reports`
   Set the directory for generating reports of tests. The default value is "`../Data/Reports`".

9. `--repeating`
   Set the number of repeating of each XSLT transformation. It means, how much times each XSLT transformation is repeated. The default value is "`1`".

10. `-p` or `--processors`
    Set the list of tested XSLT processors. If this option is not set (or is set without value), then all available processors are tested.

11. `-e` or `--processors-exclude`
    Set the list of tested XSLT processors, that we want exclude form the list of tested processors. If this option is not set (or is set without value), then any processors are excluded.

12. `-a` or `--processors-available`
    Print the list of short names of available XSLT processors (possible used in options `--processors`, `--processors-exclude`) and their kernels and full names.

13. `-m` or `--merge-reports`
    Merge the set list of XML reports in the directory set by the option `--reports`. The generated XML report has the suffix "`-merge`". If this option is set without value, then all available XML reports (without suffix "`-merge`") are mergered. XML reports are mergered in set order by the option `--order-reports`.

14. `-o` ot `--order-reports`
    Set the type of ordering for merging XML reports. Possible values are "`asc`" (for ascending ordering by names), "`desc`" (for descending ordering by names) and "`set`" (for ordering by set order of reports in the option `--merge-reports`). The default value is "`set`".

15. `-c` or `--convert-reports`
    Convert the set XML report of tests into the selected output set by the option `--convert-type`. If the file is not set, latest report in the direcotry set by the option `--reports` is converted. In case of empty directory, the report file have to be set. The generated converted report are saved into the directory set by `--reports`.

16. `--convert-type`
    Set the required type of report after converting of the XML report set by the option `--convert-reports`. Now, only the value "`html`" is available. Thus, it is also the default value.

17. `-h` or `--help`
    Show the help of the program.

18. `-v` or `--verbose`
    Print additional informations during running the program.

# Appendix E

# Test Environment - Deffinitions of XML Files

There are the list of XPath expressions defining parts of an XML file that defines one tests template.

1. `/tests` – the root element defines all tests to generating
2. `/tests/@name` – the prefix of names of generated tests
3. `/tests/@template` – the file name of a template of a XSLT templates that will be used in each generated test
4. `/tests/@templatingType` – the name of a templating driver for generating XSLT templates used in tests
5. `/tests/files` – the list of files (XML, HTML etc.) used in generated tests as an input or an expected output files
6. `/tests/files/file` – the deffinition of one prearranged file used in generated tests as an input or an expected output file
7. `/tests/files/file/@id` – the identifier of the prearranged file
8. `/tests/files/generated` – the deffinition of a file that will be generated and used in generated tests as an input or an expected output file
9. `/tests/files/generated/@id` – the identifier of the generated file

10. `/tests/files/generated/@generator` – the name of a driver used for generating the file

11. `/tests/files/generated/setting` – one setting used by the selected driver for generating the file

12. `/tests/files/generated/setting/@name` – the identifier of the setting

13. `/tests/test` – the deffinition of one test

14. `/tests/test/@name` – the suffix of the test name

15. `/tests/test/file` – one couple of files used as an input and an expected output files

16. `/tests/test/file/@input` – the identifier of the file (prearranged or generated) used in the generated test as an input file

17. `/tests/test/file/@output` – the identifier of the file (prearranged or generated) used in the generated test as an expected output for the relevant input

18. `/tests/test/setting` – one setting used by the selected templating driver for generating the XSLT template used in the test

19. `/tests/test/setting/@name` – the identifier of the setting

There are the list of XPath expressions defining parts of an XML file that define one test.

1. `/test` – the root element defines one test

2. `/test/@name` – the name of the test

3. `/test/@template` – the file name of the XSLT template used for transformations

4. `/test/couple` – one couple of files names of an input and expected output files

5. `/test/couple/@input` – the file name of an input file

6. `/test/couple/@output` – the file name of an expected output file

There are the list of XPath expressions defining parts of one report XML file.

1. `/reports` – the root element defines a collection of reports
2. `/reports/global/processors` – the list of XSLT processors that were tested
3. `/reports/global/processors/processor` – one tested XSLT processor
4. `/reports/global/processors/processor/@name` – the identifier of one tested XSLT processor
5. `/reports/global/processors/processor/@fullName` – the full name of one tested XSLT processor
6. `/reports/global/processors/processor/@kernel` – the kernel of one tested XSLT processor
7. `/reports/tests` – the list of reports of all tests
8. `/reports/tests/test` – the report of one test
9. `/reports/tests/test/@name` – the name on the test
10. `/reports/tests/test/@template` – the XSLT template used in the test
11. `/reports/tests/test/processor` – the report of testing of one processor
12. `/reports/tests/test/processor/@name` – the identifier of tested processor
13. `/reports/tests/test/processor/input` – the report of one transformation
14. `/reports/tests/test/processor/input/@input` – the path of the input file of the transformation

15. `/reports/tests/test/processor/input/@expectedOutput` – the path of the expected output file of the tranformation

16. `/reports/tests/test/processor/input/@output` – the path of the generated file of the transformation

17. `/reports/tests/test/processor/input/@success` – the flag of correctness, it contains '1' or a returned error

18. `/reports/tests/test/processor/input/@correctness` – the flag if the output and the expected output files are same (after normalization)

19. `/reports/tests/test/processor/input/@repeating` – the number of repeating of the transformation (for better measure of a time and a memory usages)

20. `/reports/tests/test/processor/input/@sumTime` – the sum of time usages of all transformations (transformation is repeated because of "`//@repeating`")

21. `/reports/tests/test/processor/input/@avgTime` – the average of time usages of all transformations (transformation is repeated because of "`//@repeating`")

22. `/reports/tests/test/processor/input/@sumMemory` – the sum of memory usages of all transformations (transformation is repeated because of "`//@repeating`")

23. `/reports/tests/test/processor/input/@avgMemory` – the average of memory usages of all transformations (transformation is repeated because of "`//@repeating`")

# Appendix F

# The List of Tests

The list of tests based on analysis of features of downloaded files.

1. Elements choose - Long
2. Elements choose - Long with not presented
3. Elements choose - Long with not presented and otherwise
4. Elements choose - Short
5. Elements choose - Short with not presented
6. Elements choose - Short with not presented and otherwise
7. Elements element - Indent template
8. Elements element - Not indent template
9. Elements foreach - Long
10. Elements foreach - Long with not presented
11. Elements foreach - Short
12. Elements foreach - Short with not presented
13. Elements if - Long
14. Elements if - Long with not presented

15. Elements if - Short
16. Elements if - Short with not presented
17. Elements template - Default
18. Elements template - Empty
19. Elements template - Nonprocedural
20. Elements template - Procedural
21. Elements text - text - output
22. Elements text - xml - output
23. Typical - Budget
24. Version - 2.0 - Character-map
25. Version - 2.0 - Empty

The list of tests based on analysis of categories of downloaded files.

1. Docbook - HTML
2. Google Search Appliance - Main
3. GraphML - GNM - Random
4. GraphML - Spring - Random and Example
5. GraphML - SVG - Random and Example
6. RSS generator - apply-templates
7. RSS generator - for-each
8. RSS reader - HTML
9. RSS reader - newsfeeds
10. XGMML - HTML

The list of other tests.

1. Element include
2. Encoding - Default
3. Encoding - ISO-8859-2
4. Encoding - UTF-16
5. Encoding - UTF-8
6. Encoding - Windows-1252
7. Namespace - Aliases
8. Namespace - Rename

# Appendix G

# Reports

This appendix includes table with exact measured time and memory usages of all tested processors. Graphs in Chapter 11.10 are created based on this table.

| **Processor** | **Windows (win1)** | | | | **Linux (lin1)** | | | |
|---|---|---|---|---|---|---|---|---|
| | **Time** | | **Memory** | | **Time** | | **Memory** | |
| | **sec** | # | **MB** | # | **sec** | # | **MB** | # |
| **libxslt 1.1.23 - PHP** | 0.074 | 3. | 10.4 | 5. | | | | |
| **libxslt 1.1.26 - PHP** | | | | | 0.049 | 2. | 45.6 | 3. |
| **MSXML 3.0** | 0.104 | 5. | 10.8 | 6. | | | | |
| **MSXML 6.0** | 0.088 | 4. | 10.4 | 4. | | | | |
| **Sablotron 1.0.3** | 0.123 | 6. | 6.1 | 2. | 0.099 | 3. | 24.0 | 2. |
| **Saxon 6.5.5** | 0.471 | 8. | 22.6 | 8. | 1.121 | 6. | 211.1 | 6. |
| **Saxon HE 9.4.0.2** | 0.653 | 10. | 34.6 | 10. | 1.717 | 7. | 268.2 | 7. |
| **XT 20051206** | 0.267 | 7. | 21.8 | 7. | 0.599 | 4. | 141.1 | 4. |
| **Xalana 2.7.1** | 0.501 | 9. | 26.1 | 9. | 1.095 | 5. | 190.4 | 5. |
| **xsltproc 1.1.23** | 0.056 | 1. | 5.9 | 1. | | | | |
| **xsltproc 1.1.26** | 0.062 | 2. | 6.2 | 3. | 0.041 | 1. | 20.2 | 1. |

Table G.1: **Average time and memory usages for processors**: There are average time and memory usages for all tested processors in the table. Averages are counted separately for Windows and Linux. Each column marked with '#' contains the order of values in column to the left. Blank boxes mean that the given processors were not tested for the given OS.