VŠB - Technical university of Ostrava
Faculty of electrical engineering and computer science
Department of computer science

URI Transformer

Diploma thesis

2007 Jan Bednařík

I declare that I have elaborated this diploma thesis on my own. And I have named all literature sources and publications I have used.

In Ostrava May 2, 2007 .....

Thanks to Stefan Matheis for suggestions on the Typo3 extension, and to Derek J. Muir for proof-reading of this diploma thesis.

Abstract

This work is about transforming URIs, that lead to creating nice URIs, or so to say, CoolURIs. Nice URIs are bound with internet search engines optimization, which has recently become very common. It leads to getting a certain web page to the top spots in search results. A tool for transforming URI has been developed. The tool should be able make generating nice URIs easier. Most of this work is about this tool and describes its configuration. At the end, two case-studies of its usage are presented.

Keywords: URI, CoolURI, DirtyURI, user-friendly URIs, mod_rewrite, URI transformation, SEO

Abstrakt

Tato práce pojednává o transformaci URI, která vede k vytvoření hezkých URI neboli CoolURI. Hezké URI jsou úzce vázány s optimalizací pro internetové vyhledáváče, což je v poslední době populární disciplína směřující k posunutí určité webové stránky na čelní pořadí výsledků hledání. Pro transformaci URI byl vyvinut nástroj, jenž má generování hezkých URI usnadnit. Převážná část této práce se zabývá právě tímto nástrojem a popisuje jeho konfiguraci. V závěru jsou předvedeny dvě studie jeho praktického použití.

Klíčová slova: URI, CoolURI, DirtyURI, přátelské URI, mod_rewrite, transformace URI, SEO

List of used acronyms a symbols

ASCII	–	American Standard Code for Information Interchange
CMS	–	Content Management System
HTML	–	Hypertext Markup Language
HTTP	–	HyperText Transfer Protocol
IIS	–	Internet Information Server
ISAPI	–	Internet Server Application Program Interface
MVC	–	Model View Control
PHP	–	PHP Hypertext Preprocessor
SEM	–	Search Engine Marketing
SEO	–	Search Engine Optimization
SERP	–	Search Engine Result Page
SQL	–	Structured Query Language
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator
URN	–	Uniform Resource Name
WWW	–	World Wide Web

Table of contents

1 Introduction

Since the beginning, internet[1] has grown into a worldwide communication device. It has become a part of our everyday life. Many people cannot even imagine living without it. Actually, some people's existence depends on the internet. When the internet is down, they have nothing to do and cannot work.

Everything that we can find on the internet has to be identified somehow. In the same way as every house has its own unique address. The addresses, we use to locate things on the internet, are called URI. URI is a standard, created by the W3C group [4].

This work is about URIs (2.1) and its subset – URLs (2.2) and their rewriting or, if you like, transformation. The reason why we need to transform URIs will be described later on. If brief, when we surf the net, we can often find URLs like *http://www.example.com/index.php?id=1234&mod=99&catId=32&hash=sdf322zach2443*. There are many reasons why such URLs are bad (4.1) and why URLs without parameters (the stuff after ?) are better and preferred. But it is a problem for programmers not to use such URIs. Therefore we need some means of transformation, so programmers could work with these, but users would see different, nicer URLs.

This work may bring an unique thing to this area, as there is nothing in existence yet. We may find a few basically similar things, but they are always dependent on something other (such as a content management system). The results of this work should not depend on any other system, but should be so flexible, that there will not be any problems including it to any other existing system. To prove this, a part of this work's output will be an extension for the CMS Typo3[2] as well as a case-study of a dedicated server.

[1] The word "internet" is no longer a name. We do not speak about internet as there was only one. It is all around us, living with us. We do not need to call it with a name. That is why word "internet" wil not be written with a capital letter. It is not Internet anymore, it is just internet.

[2] http://www.typo3.org, http://www.bednarik.org

2 Uniform Resources definitions

At first, we should say what URIs, URLs and URNs are:

2.1 URI

URI provides a simple and extensible means for identifying a resource.

- *Uniform*: Uniformity provides several benefits. It allows different types of resource identifiers to be used in the same context, even when the mechanisms used to access those resources may differ.
- *Resource*: The specification does not limit the scope of what might be a resource; rather, the term "resource" is used in a general sense for whatever might be identified by a URI. Familiar examples include an electronic document, an image, a source of information with a consistent purpose, a service, and a collection of other resources.
- *Identifier*: An identifier embodies the information required to distinguish what is being identified from all other things within its scope of identification. Our use of the terms "identify" and "identifying" refer to this purpose of distinguishing one resource from all other resources, regardless of how that purpose is accomplished.

Examples of URI:

- http://www.example.com ;
- http://www.example.com/page/subpage?parameter=1 ;
- mailto:John.Doe@example.com ;
- urn:oasis:names:specification:docbook:dtd:xml:4.1.2 .

There are two classes of URIs – those that identify by specifying location (Uniform Resource Locators) and those that do so by naming the resource (Uniform Resource Names), such as purls (persistent URLs).

2.2 URL

The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location").

2.3 URN

A Uniform Resource Name is a Uniform Resource Identifier that uses the urn scheme, and does not imply availability of the identified resource. Uniform Resource Names are intended to serve as a persistent, location-independent resource identifiers, and are designed to make it easy to map other namespaces (that share the properties of URNs) into URN-space. Therefore, the URN syntax provides a means to encode character data in a form that can be sent in existing protocols, transcribed on most keyboards, etc. [16]

3 Search Engines

As we already know, internet consists of, besides other things, countless number of pages (which are identified by their URI, as we well know). Now imagine looking for information among all these pages. It is like looking for a needle in a giant haystack. Or like looking for a person, not knowing their address, in a city like New York[3]. This task may seem impossible, but if we had some clues, we may narrow the perimeters. Well, there are no clues of how to find a needle in a haystack, but to find a person in a city we can use e.g. a telephone book. Still, there can be dozens of John Does in the phonebook, if we do not have any more clues, we just need to pick one up and go see if he is the one we are looking for. Now there is a question, how do we pick them up? We can start by using alphabetical order, pick one at random, or choose by their addresses (we would definitely prefer Broadway to Bronx). See the analogy? Searching for a web page is like searching for a person. But instead of phonebooks we have search engines. Even further, the search engines do not search among all web pages[4] also phonebooks record only people who have telephones.

There are a lot of search engines[5], here is a short list of just a few of them:
International:

- Google: http://www.google.com (or its czech mutation http://www.google.cz) ;
- Microsoft Live Seach: http://search.msn.com ;
- Yahoo!: http://www.yahoo.com .

Czech:

- Seznam: http://www.seznam.cz ;
- Jyxo: http://www.jyxo.cz ;
- Centrum: http://www.centum.cz ;
- Altas: http://www.atlas.cz .

When we query a search engine, it gives us millions of webpages. Results are divided into SERPs, usually 10 results each. Pages are not sorted in an alphabetical order as in a phonebook. However, there are many rules which put one page ahead another. All these rules create a number, called the rank (Google's pagerank, Seznam's srank etc.). The higher the rank is, the higher the page get in SERP. It is obvious that every page would like to be in the top spot. There are ways of how to achieve it. These ways are called SEO (SEM) and we'll talk about it later on.

[3] It looks like we are comparing internet and its pages to New York and its houses. The idea is OK, but the scale is not. Maybe even looking for a person all over the world would not make the scale right.

[4] The term "web page" will be used in a meaning "page we can find on the internet, identified by a URI"

[5] Most of these search engines have two parts – catalogue and fulltext. We will refer mainly to the fulltexts further in this document.

3.1 Eye tracking in MSN Search: Investigating snippet length, target position and task types

Microsoft has published a study [5] in January 2006, where they examined the influence of various factors on search results usability. The study was performed upon SERPs of Microsoft's MSN Search (Live Search now) with 10 results per page.

Users fulfilled a few tasks and the goal was to find some specific information. Basically, these tasks were divided into two groups:

- *Navigational tasks* – e.g. "Find the main page of the World Football Championship 2006" (there was always only one right link),
- *Informational tasks* – e.g. "Find out the area of Oklahoma City ZOO" (there was always one most suitable link).

The purpose of the research was to find out what influence on speed and accuracy had the following factors:

- the target link position in the SERP,
- length of the description,
- type of the task.

Some of the results of this research were:

- Longer descriptions significantly raise successes while fulfilling the informational tasks and shorten the time of the fulfillment,
- No matter which link the user chose among the results on the SERP, they look over at least 3 or 4 links,
- Before the user decides to change the query, they look at at least 8 links in the SERP,
- When deciding whether a link is useful or not, users take into account the URI of the result.

Especially the last list item is quite important to this work, as it shows us, that nice[6] URIs are welcomed and people are more tend to looking at them. What the nice URIs are will be discussed further on.

3.2 Search Engine Optimization

As the name of this method implies, it is focused on optimizing web pages, so they are better for search engines. The most important thing to mention is that what is better for people is also better for search engines. SEO consists of many factors, which are divided into two groups – on and off page factors.

[6] We should say "cool", rather than "nice" in context with this text. But it may be a bit confusing right now.

3.2.1 On-page factors

Lots of web pages focused on describing on-page factors can be found for example via Google, [7]. The problem is, it is not possible to say which on-page factors are the best. The selection and order depends on the algorithm of a certain search engine, which can differ from case to case. One of the orders could be following, [11].

1. The page's title tag (include keywords),
2. Link anchor text (include keywords),
3. Use of keywords in the document,
4. The accessibility of the document (can search engines see it and read the content?),
5. Internal linking (linking within your site from one page to another),
6. Primary subject matter of your site (consistent use of your markets language, focus on content),
7. Links to external sites (quality links may reflect upon you, but dont overdo it),
8. Link popularity in topical community (do other sites about your topic link to you?),
9. Global link popularity (do all kinds of sites link to you?),
10. Keyword spamming (a negative factor, do not overuse your keywords).

3.2.2 Off-page factors

It is said that off-page factors are more important than on-page. It could seem obvious, off-page factors tells us about our web site what others think of it. One of them, the number of links linking to our page, shows how popular our site is. We can create a list of off-page factors importance, but again – it cannot be said, that it is a finite and valid. Many resources about it can be found over the internet, [8].

1. Link Anchor Text,
2. Inbound Links from Varying IP's,
3. Site Wide Inbound Links,
4. Directory Listings.

3.3 Search Engine Marketing

Search Engine Marketing is a sister of SEO. Sometimes it is hard, maybe impossible, to get to the top of the search results using only SEO. And that is when SEM steps in. We all know well so called "sponsored links" in search engines results pages. So, if we cannot make it to the top using SEO, we can pay for it. And that is called SEM.

SEM is most important for businesses who sell goods and services online or who use their websites to generate sales leads. Other goals for SEM including:

- building a brand,
- enhancing reputation with investors,
- generating media coverage,
- driving traffic to physical business locations.

Organizations such as non-profits, universities, governments, and political parties also use SEM to promote their ideas.

Basically SEM is oriented on buying sponsored links at search engines result pages. Usually a few words or phrases are bought and when a search, containing these words or phrases, is done, the sponsored link shows up.

4 URIs

Let us start with an example, which should make clear what is a CoolURI and what is a DirtyURI. A CoolURI is for example

http://www.example.com/furniture/chairs/wheelchair/

and on the other hand a DirtyURI for the same page could be for example

http://www.example.com/index.php?pageId=28&itemId=3485&catId=7492

URI should have some sort of semantics, so we could say what can be found under this URI. That is why CoolURIs are sometimes called user-friendly URIs. However, no one would input a whole URL to their browser[7], it is better when URI is readable and we can say, where we are or where we will get to. Therefore, it is not a big deal when a CoolURI is longer than DirtyURI (meaning both would point to the same page).

There has been always a flamewar[8] whether to include whole path to a document into URI or not. Some would prefer URIs *http://www.example.com/path/to/product/theproduct*, others however prefer *http://www.example.com/theproduct*.

Another flamewar has always been between followers of the directory approach and followers of the file approach. Directory approach is when a trailing slash is always included in a URI (*.../page/*), file approach is the opposite (*.../page*).

When a file approach is used, there is a question, if some sort of extension is included to a URI (*../page.html*) or not.

As we can see, there are no rules about CoolURIs. It always depends on a webmaster what is chosen.

4.1 DirtyURIs are bad

Complex, hard-to-read URIs are often dubbed dirty URIs because they tend to be littered with punctuation and identifiers that are at best irrelevant to the ordinary user. DirtyURIs are commonplace in today's dynamic web. Unfortunately, DirtyURIs have a variety of troubling aspects, including:

4.1.1 DirtyURIs are difficult to type

The length, use of punctuation, and complexity of these URIs makes typo error commonplace.

4.1.2 DirtyURIs do not promote usability

Because DirtyURIs are long and complex, they are difficult to repeat or remember and provide few clues for average users as to what a particular resource actually contains or the function it performs.

[7] Users prefer to click rather than type a long address.
[8] http://en.wikipedia.org/wiki/Flamewar

4.1.3 DirtyURIs are a security risk

The query string which follows the question mark (?) in a DirtyURI is often modified by hackers in an attempt to perform a front door attack into a web application. The very file extensions used in complex URLs such as .asp, .jsp, .pl, and so on also give away valuable information about the implementation of a dynamic web site that a potential hacker may utilize.

4.1.4 DirtyURIs impede abstraction and maintainability

Because DirtyURIs generally expose the technology used (via the file extension) and the parameters used (via the query string), they do not promote abstraction. Instead of hiding such implementation details, DirtyURIs expose the underlying "wiring" of a site. As a result, changing from one technology to another is a difficult and painful process filled with the potential for broken links and numerous required redirects. [14]

4.2 Recommendations on CoolURIs

It has been said already, that there are no rules on CoolURIs. However, there are still a few recommendations.

4.2.1 Short, clean, sweet

Parts of all CoolURIs (directories) should be meaningful. Obviously, */products* is much better than just */p*. However more is sometimes less, so the parts should be kept as short as possible. Changing */products* to */productcatalog* adds only more letters to the URI, but does not add anything useful. The parts, the URI is composed from, should be the shortest identifiers consistent with a general description of the page's (or directory's) contents or function.

4.2.2 Hyphen is better

The first recommendation cannot be always followed. Sometimes it is even better to include more words to a part of a URI. For example, URI of a single product detail page should include the name of the product. Since a space is not allowed in a URI and it should not be just left out, it has to be substitued by something else. There are only two possible choices[9] and that it an underscore or a hyphen. And because Google says, that "underscores are not treated as word separators" and also underscore is often difficult to notice and type, hyphen should be used as a word separator. Some may want to use camel-style URIs (*.../MyProductsNameIsCool/*), but ...

[9] Since a part of a URI can contain only: ALPHA / DIGIT / "-" / "." / "_" / "~"

4.2.3 Use lower case

Casing in URIs is troublesome because depending on the web server's operating system, file names and directories may or may not be case sensitive. For example, URIs such as *http://www.example.com/Products.html* and *http://www.example.com/products.html* are two different files on a UNIX system but the same file on a Windows system. Add to this the fact that *www.example.com* and *WWW.EXAMPLE.COM* are always the same domain, and the potential for confusion becomes apparent. The best solution is to make all file and directory names lowercase by default.

4.3 Advantages of CoolURIs

4.3.1 Idea what is behind

CoolURIs give a pretty, let's say, cool idea, as to what can be found there, on the other hand, the DirtyURI gives us hardly any idea.

4.3.2 CoolURIs do not change, [3]

> What makes a cool URI?
> A cool URI is one which does not change.
> What sorts of URI change?
> *URIs do not change: people change them.*

There may be tons of reasons to change URIs, such as "We just reorganized our website to make it better.", "Well, we found we had to move the files..." etc. but none of these reasons are good enough to be a reason to change a CoolURI. So no matter how old the URI is or how many times the page at this URI has been changed, the URI pointing at this page should be still the same. It should not matter what kind of technology, whether PHP, ASP or something else, has been used to generate a page.

4.3.3 Users like CoolURIs

Another advantage has been mentioned already (3.1). Users are more likely to click on a nice URI (CoolURI) rather than dirty. And this is probably the most important advantage. It brings visitors and visitors bring money.

4.3.4 Searchengines like CoolURIs

Although it is believed that search engines do not give much credit to keywords in a URI, it is no doubt that search engines results are mainly based on the frequency of key words in a web page and the URI could be, and should be, assumed as a part of a web page. At least some little bonus a page with a CoolURI should have in comparison with a DirtyURI page. So if a search for, from our example (4), word "chair" is held, regarding only URIs and not other on-page/off-page factors, the first page from the example should be higher in the results.

4.3.5 Google advices to use CoolURI

Some time ago, Google said, in its Webmaster Guidlines [17], that it did not index pages with a parameter "id" in an query string (parameter "id" is quite common when using DirtyURIs). The official statement was [6]:

> Do not use "&id=" as a parameter in your URLs, as we do not include these pages in our index.

This sentence has been removed from the Guidlines, but they still advice to use user-friendly, aka cool, URIs [6].

> Keep in mind, however, that dynamic URLs with a large number of parameters may be problematic for search engine crawlers in general, so rewriting dynamic URLs into user-friendly versions is always a good practice when that option is available to you.

4.4 Disadvantages of CoolURIs

There are certainly no disadvantages with regard to users, visitors or search engines. The disadvantages concern only web programmers at most.

4.4.1 Parameters

A dynamic web page is generated based on some parameters passed to this page via certain methods (that means GET[10], POST[11] etc.). Query strings in DirtyURIs always contain doubles key=value. It is very easy to access these parameters (e.g. in PHP we use `$_GET['key']`). Values are integers, and integers are always precise, cannot be mistaken and are used in databases as primary keys. But when CoolURIs are used, values of these parameters are replaced by some meaningful content and keys a thrown away. It is not easy to access variables hidden in CoolURIs and the meaningful content of URIs is not necessary always unique and precise as the integer keys.

4.4.2 Transformation

It is obvious, that we need to get parameters out of CoolURIs somehow. That means we need to transform CoolURIs into DirtyURIs, while the DirtyURIs are not visible on the outside (for visitors, users and search engines) of course. This transformation is not automatic, it needs to be somehow defined. There are some tools that can transform CoolURIs into DirtyURIs, those will be mentioned later. But there is nothing that can do it the other way. So programmers have to make up, what the CoolURIs will look like in their application and work with them from the begining. That can be a problem if they find out later, that the first design of the CoolURIs was wrong. It would mean rewriting all URIs used so far[12].

[10] GET method is a method of passing variables in a URI.

[11] POST variables are passed from a form usually.

[12] The aim of this work is to get rid of this disadvantage.

5 Rewriting Tools

Rewriting URIs means two transformations – parameters (DirtyURI) to CoolURIs and CoolURI to parameters. The only publicly available rewriting tools, however, provide only one way rewriting (CoolURIs to parameters). That means, we need to put CoolURIs into our scripts, because we have no way how to change parameters into a CoolURI. This has been discussed in 4.4.2.

The most popular and widely used tool is *mod_rewrite* implemented in the Apache web server. Microsoft's Internet Information Services has a ISAPI filter, which can be used for rewriting too. Universal method, which is available on almost every server, is "404 Not Found" page rewriting.

5.1 Mod_rewrite

There are two statements, which perfectly describe the mod_rewrite, [1].

> The great thing about mod_rewrite is it gives you all the configurability and flexibility of Sendmail. The downside to mod_rewrite is that it gives you all the configurability and flexibility of Sendmail.
>
> – Brian Behlendorf, Apache Group

> Despite the tons of examples and docs, mod_rewrite is voodoo. Damned cool voodoo, but still voodoo.
>
> – Brian Moore, bem@news.cmc.net

And the welcome sentence tells us, what mod_rewrite creators think of if:

> Welcome to mod_rewrite, the Swiss Army Knife of URL manipulation!

This module uses a rule-based rewriting engine (based on a regular-expression parser) to rewrite requested URIs on the fly. It supports an unlimited number of rules and an unlimited number of attached rule conditions for each rule to provide a really flexible and powerful URI manipulation mechanism. The URI manipulations can depend on various tests, for instance server variables, environment variables, HTTP headers, time stamps and even external database lookups in various formats can be used to achieve really granular URI matching.

Mod_rewrite operates on full URIs (including the path-info part) both in per-server context and per-directory context and can even generate query-string parts on result. The rewritten result can lead to internal sub-processing, external request redirection or even to an internal proxy throughput.

Obviously, mod_rewrite is very powerful and very complex thing. Complex is probably the best word to describe mod_rewrite. A synonym to it is *complicated*, on the other hand, another synonyms may be *full* or *total*.

The mod_rewrite works in a way, that it converts incoming (requested) URI to another, it may redirect, instead of silent rewriting, as well. This means, that mod_rewrite does not rewrite URIs in our source codes. Web creators have to put final URIs (CoolURIs) into their templates, but they work with another versions, the one rewritten by mod_rewrite. This can become quite unpleasant, especially when they find out they need to change the URIs and so have to rewrite them all manually. Or they just have to stick with these rules forever.

Let us present a simple example of rewriting URIs like *www.example.com/somepage.html* to *www.example.com/index.php?id=somepage*:

```
RewriteEngine On
RewriteRule ^(.*)\.html$ %{DOCUMENT_ROOT}/index.php?id=$1 [L,QSA]
```

Listing 1: Mod_rewrite example

In this case, index.php will be called with a GET parameter id, whilst URI remains the same.

To make a better picture about mod_rewrite, there are loads of examples along with a rewriting guide at [2].

5.2 404 page rewriting

The 404 or Not Found error message is a HTTP standard response code indicating that the client was able to communicate with the server, but the server either could not find the file that was requested, or it was configured not to fulfill the request and not reveal the reason why, [9]. When a web server is required to respond for a request for a web page and this page is not found on the server, response code 404's associated string is "Not Found" is sent. When sending this response, web server can send a HTML document. In other words, when a page is not found, another one can be presented instead.

This means that instead of a static page a script can be presented. This script can decide whether 404 status will be send or not. Basically it can send any page, even with correct status 200 OK. So, based on the "not found's" document URI, the script can send another one and that is the same as rewriting though.

This method is available with Apache server, IIS, Tomcat and probably even other webservers. As these three are the most used servers, it may be useful to show how this could be achieved.

5.2.1 Custom 404 with Apache

The following text needs to be added to a *.htaccess* file: `ErrorDocument 404 /notfound.html`.

5.2.2 Custom 404 with IIS

This is set in the *Administrative Tools*, *Internet Services Manager* and there are *Properties* of the server. These properties can be edited on the *Custom Errors* page, where our custom page can be set as 404 error page. It is necessary to change *Message Type* to *URI*.

5.2.3 Custom 404 with Tomcat

The best way is to add the following in the *web.xml* for the context.

```
<error−page>
   <error−code>404</error−code>
   <location>/notfound.html</location>
</error−page>
```

Listing 2: Setting custom 404 page in Tomcat

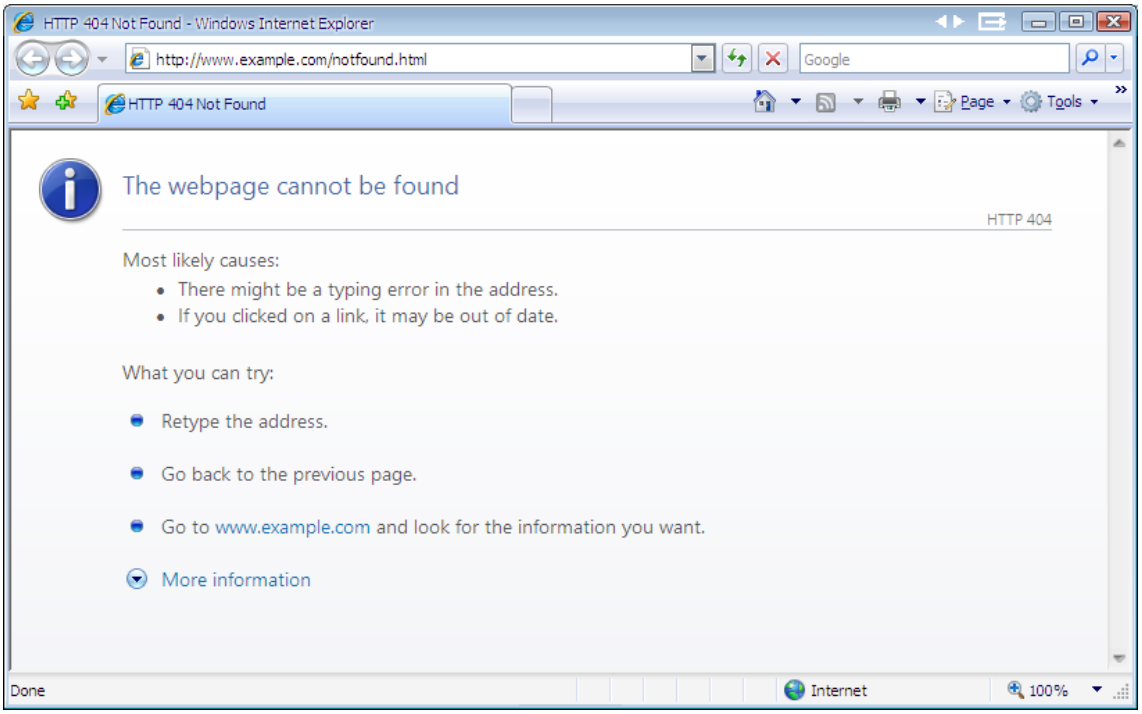

Figure 1: Internet Explorer comes configured to "Show friendly HTTP error messages" which replace the server's normal error messages.

5.3 ISAPI filter

ISAPI is a set of functions in Windows, which enables web applications to be created, [12]. Special type of ISAPI library is a ISAPI filter, which is used for accepting all HTTP requests. So it is possible to create a ISAPI filter for accepting every URIs. The disadvantage of ISAPI is that it is dependent on a used platform. Nowadays, only Windows servers (IIS and Personal Web Server), Apache servers running on Windows and a few others can work with ISAPI and ISAPI filters. They can be written in any programming language and have to be compiled with a compiler, which can create these libraries.

An example of a ISAPI filter can be found on [13].

5.4 .NET

The Microsoft's .NET platform has a special method for rewriting. At the beginning of every request it calls a special method protected void Application_BeginRequest (Object sender, EventArgs e) which is placed in the *Global.asax* file.

This function can be redefined and it can place a different URI into HttpContext's method. Best would be to show an example of such function [15].

```
protected void Application_BeginRequest(Object s,EventArgs e)
{

    HttpContext incoming = HttpContext.Current;
    string oldpath = incoming.Request.Path.ToLower();
    string pageid; // page id requested

    // Regular expressions to grab the page
    // id from the pageX.aspx
    Regex regex = new Regex(@"page(\d+).aspx", RegexOptions.IgnoreCase | RegexOptions.
        IgnorePatternWhitespace);
    MatchCollection matches = regex.Matches(oldpath);

    if (matches.Count > 0)
    {
        // Extract the page id and send it to Process.aspx
        pageid = matches[0].Groups[1].ToString();
        incoming.RewritePath("Process.aspx?pageid=" + pageid);
    }
    else
    {
        // Display path if it does not containt pageX.aspx
        incoming.RewritePath(oldpath);
    }
}
```

Listing 3: An example of the Application_BeginRequest method

5.5 Others

Previous methods are not the only ones, of course. There are a lot of other methods for URL rewriting. For illustration:

- Apache and PHP:
  http://www.tutorio.com/tutorial/php-alternative-to-mod-rewrite-for-se-friendly-urls;
- Isapi Rewrite: http://www.isapirewrite.com/;
- J2EE URL rewrite: http://tuckey.org/urlrewrite/;
- IIS Rewrite Engine: http://www.qwerksoft.com/products/iisrewrite/.

5.6 Dirty to Cool

The methods presented so far can do only one way transformation – cool to dirty. That means, if a URI is requested, it is transformed into a list of parameters and values. Basically the values are not transformed in any ways.

This one way transformation is quite a disadvantage, as we need to create our own way how to put CoolURIs into our source codes. That means we have two tools for one thing – we use something from above to get parameters from URIs and we use our own solution to get a URI from parameters.

It is really strange, that there is not any general-purpose tool that would do the two way transformation. There are a few CMSes which have tools that can do the two ways, but they are dependent on the CMS, so there cannot be any mention of a multi-purpose.

As has been said, there is no two way transformation tool, but we should add *so far*. The aim of this work is to change this thing.

6 URI transformer

This part of the thesis will be sort of a manual for the transformer. It will be focused on the PHP implementation, however the Java implementation is available as well, it will be mentioned in the end. The main goal is, there will not be any difference than necessary between PHP and Java usage.

6.1 Introduction

The basic idea is very simple here. When we create a web application, we need to create anchor tags with links, that would connect pages of our system together into one application. As programmers, we always prefer DirtyURIs to CoolURIs, because we can use integer identifiers rather than word representations. Another reason is, that when we create a link to a page with parameter *id=X*, we will get the parameter with the value on the linked page. When CoolURIs are used, we would need to create a link without parameters, but we would have parameters in the linked page and in addition the parameters would not be integer, but verbal. This sounds like every programmer's nightmare.

So, what is the idea and the goal here? It is to allow programmers to work with DirtyURIs, while on the output would be CoolURIs. That means if a link to a page with *id=X* is set, a parameter *id* with value *X* will be available on the page. But there will not be any link containing *id=X*, there will be only cool links without parameters.

6.1.1 Cool to Dirty

The URI transformer does not do things much differently than the tools, which have been mentioned before. It simply takes a CoolURI, extracts parameters from it somehow, and pass these parameters to a script, where they can be accessed.

But it would not be useful, just to add a new tool that would do the same as the others. The question is, what are the others missing and this transformer is not, so it can prove itself useful:

- *Database look-up* – doubles value and key, where value is a part of a CoolURI and key is a part of a DirtyURI, are often stored in a database. This tool can look-up the keys automatically;
- *Value maps* – a part of a URI can be an item of a set of values, this transformer can find a key to the value and therefore translate it;
- *Single ID from path look-up* – a tree structure is often store in a database where a single item stores its ID and its parent's ID. It should be useful to get a single ID from a page-path[13] in a URI;

[13] a page-path id for example: /electronics/tv/plasma/wide-screen/

6.1.2 Dirty to Cool

Common rewriting tools do not provide any means to transform URIs in this direction. The result is, programmers have to put the URI they want to display in the browser directly into their source codes. This is unwanted, as the URIs cannot be changed during the application development. Anyway, this has been already discussed.

Most programmers have their own tool for Dirty to Cool transformation, which is often built for and dependent on a concrete system. Afterwards, the situation is that a separate tool is used for each transformation direction.

If programmers use DirtyURIs in their scripts, we need to have some sort of method which would find out these URIs and transform them into Cool. There are possibilities such as:

- *Output filter* – parse outgoing data, find out URIs which point within the application and transform them;
- *URI function* – instead of writing a URI into a script, the programmer could call a function, pass it some sort of associative array and this function would return a CoolURI, which would be used in the page;
- *Template plugin* – templates engines are a favourite way for separating data from design, therefore a plugin which would transform URIs could be an option.

6.1.3 Additional features

- *Caching* – translation process may be a bit power and time consummating, so it may be useful to store the result into a cache;
- *Old links redirection* – despite that CoolURIs do not change, it is sometimes needed to change a URI. For example, if a name of a page is changed, it would seem ridiculous if the URI stayed the same. But the URI cannot be just thrown away and forgotten. The old URI should be redirecting to the new one. However, this redirection does not need to be forever, so some expiration can be set;
- *Link Manager* – sometimes other than logic URI can be required, that means, we would like to set another URI than the one automatically generated. Therefore, a tool where any transformation could be set may be useful. Beside that, this manager also displays cached URIs and old URIs which are redirected to new ones.
- *Comprehensible settings* – the mod_rewrite is a voodoo thing because of its settings, which are hard to understand. Settings of this transformer are stored in a XML file, which can be self-explanatory;

6.2 Requirements

6.2.0.1 Web application Since the transformer works only with parameters in a URI, it does not take into account a file. It is supposed that the application is controlled by a

single script (in PHP it is usually *index.php*). This approach – single script – has become quite popular because of the MVC pattern. MVC has usually only one controller, which delivers tasks to the models. And this controller is the single script.

6.2.0.2 PHP

- Apache server with mod_rewrite enabled or any other server where can be a custom 404 page set,
- PHP 5+ with SimpleXML extension enabled,
- MySQL 4.1+.

6.2.0.3 Java Java implementation uses a few common packages that need to be included:

- Dom4j – for parsing the XML configuration file,
- MySQL Connector – for connecting to a database

6.3 Installation

First of all we need to redirect all HTTP requests to the controller. Since the transformer is just a script in PHP (or Java), it cannot be achieved using it. We have to use a server based rewriting tool, such as those mentioned in (5). This may seem a bit awkward as it does not get us rid of the other transformers. This is true, but we just use them to do one simple thing, so their set up is fairly simple as well.

6.3.1 Redirecting to a controller

We can set either mod_rewrite rules or a custom 404 page:

6.3.1.1 Setting mod_rewrite If we have the mod_rewrite available, we just need to put a few lines into the *.htaccess* file:

```
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-l

RewriteRule .*$ index.php
# index.php is the controller here
```

Listing 4: Mod_rewrite settings

6.3.1.2 Setting 404 page We just set the controller as the 404 page. Therefore every request which will be targeted to a non-existing file/directory will be forwarded to the controller. But make sure to send a right status code back to the browser[14] otherwise server will return *404 Not Found* and serve the controller. Therefore, everything would look OK, but server would say that there are no pages and search engines would not index any.

6.4 Basic functions

6.4.1 Adding the URI transformer into the control script and accessing translated parameters

At first, the transformer classes need to be included into the script:

```
require_once 'link.Main.php';
```

Listing 5: Including transformer into a script

than we can get an instance of the *Link_Translate* class and call *cool2params* method. Because the class implements the pattern singleton[15] we cannot create an instance using *new*, but we need to get the instance using method *getInstance*:

```
$lt = Link_Translate::getInstance();
$lt->cool2params();
```

Listing 6: Transforming a CoolURI to parameters

6.4.1.1 Link_Translate->getInstance(): This singleton method accepts one optional parameter *xmlconffile* – location of a XML configuration file, default is *CoolUriConf.xml*. Returns instance of the *Link_Translate* class.

6.4.1.2 Link_Translate->cool2params(): The *cool2params* method accepts one optional parameter *uri* – a URI to be parsed, if empty the URI will be retrieved from server variables. Returns an array of parameters, or based on the configuration, it can save the parameters to any array.

In PHP we would like to save these parameters into the $_GET or $_REQUEST superglobal array probably. This could be achieved by either telling the transformer to do so in the configuration or saving it manually:

```
$_REQUEST = $lt->cool2params();
```

Listing 7: Saving parameters into an array manually

From now on, the translated parameters can be accessed as array items.

[14] If everything is OK, a *200 OK* status should be sent.

[15] http://en.wikipedia.org/wiki/Singleton_pattern

6.4.1.3 Link_Translate−>GET(),sGET(),GETAll(): Or we can retrieve them using these functions:

- GET($var) – returns a variable unslashed
- sGET($var) – returns a variable slashed
- GETall() – returns all parameters in an array

6.4.2 Transforming parameters to a CoolURI

6.4.2.1 Link_Translate−>params2cool() This is the main function, called by the other functions. It accepts 5 parameters, but last 4 are optional. First parameter *params* is an array of query string's parameters. Second parameter *file* is used only if CoolURIs are disabled, it is the file before the query string. Next boolean *entityampersand* tells whether to convert "&" into an entity or not. If the fourth *dontconvert* is true, no URI trasformation is done. And last one if true, will force the link to be updated in a cache, even when the cache is not expired.

6.4.2.2 Smarty plugin A plugin to the template system Smarty[16] is available. It is a function *link* with one parameter *href* which contains a DirtyURI. This DirtyURI is parsed and transformed into cool.

6.4.2.3 Link_Translate−>replaceAllLinks() If a web application is complete already, this last option may become handy. It can take any string as the parameter *string*, that means a whole web page too, find out links and transform those which does not start with "http" to cool. So, this can be used as a pre-output filter.

Note: If the application outputs data directly to the browser, ob_* functions can be used to capture data, so they can be processed.

6.5 URI

In this case, a URI is looked upon as a set of values separated by some characters or strings. If the URI is divided by these separators, it breaks up into a few kind of values:

- unwanted values,
- static values,
- predefined parts,
- valuemaps,
- pagepath,
- other parts.

[16] http://smarty.php.net/

6.5.1 Unwanted values

Parts that are not part of the CoolURI. That means they should not be there and if they are in the URI, they create duplications. If a page has a duplication, the page can be retrieved with more different URIs. A duplication is bad, because it shatters a rank the page has (3). Therefore if a duplication is found, it should be redirected to a default URI. To understand more what a duplication is, let's give an example: *http://www.example.com/* and *http://www.example.com/index.html* are 2 different URIs, but serve the same page.

If such part is found in the set of values, it is deleted and a visitor is optionally redirected to a URI without such parts.

6.5.2 Static values

Such values are always a part of a CoolURI. That means they are not translated in any way, they are just put at their place in the URI. For example, if a 3^{rd} level domain is part of a CoolURI[17] the 2^{nd} level (example.com) needs to be included in the URI.

6.5.3 Predefined parts

These parts are identified by their section. As if a numbering is used and pages are identified by "page-1", "page-2" .. "page-X" in a URI, it is obvious that the value – the page number – is stored in a URI part which starts with "page-" string, followed by a number.

6.5.4 Valuemaps

A parameter can acquire more values. If the values and their keys are known and storing them in a database would be superfluous, they can be put into a so called valuemap. This valuemap can be viewed as a common associative array.

6.5.5 Pagepath

Pagepath can be composed from more than one URI part. It is the part which shows a path to a document (or a page). This is the only exception when a single URI part is not translated into one parameter, but indefinite number of parts are translated into one parameter.

6.5.6 Other parts

This is what is left after those above are processed. In comparison with predefined parts this part is not identified in any way and against valuemaps it can take any value.

[17] This can be seen on many e-shops or advertisiment servers, where a main category is stored in the 3^{rd} level domain, such as http://cars.example.com, http://bikes.example.com.

6.6 Caching

Caching is optional, but recommended as it can significantly increase speed and decrease load of a server. URI transforming can take up to tens queries into a database, whilst retrieving a cached URI takes only a few. Besides that it has some advantages against the non-caching solution.

6.6.1 Duplicities

When a CoolURI is translated in the common way, some parameters are retrieved and a page is displayed. There is not any check whether transforming these parameters into a CoolURI would produce the same URI as the one in the request. The reason for this is performance, because it would be too much time and resources consumption to translate a CoolURI into parameters and back just because of a duplicity check.

Despite that, if the caching is enabled, URI is not found in the cache and some parameters are retrieved, there is a check if such parameter combination is in the cache present already. If so and cached URI is different to the one in the request, request is redirected to the cached URI and therefore no duplicity is possible.

6.6.2 More transforming options

Transforming a URI both ways without the caching is possible, but brings some limitations to the application and the configuration is a bit more difficult. When caching, it is possible to limit the transformation from a CoolURI into parameters to retrieving parameters from cache. With this option set, if nothing in the cache is found, the transformation ends and no page is displayed (what exactly happens depends on the configuration).

This means that CoolURIs are generated from parameters and stored in the cache then, so URIs have to be pre-generated before they really exist. An URI is generated when it is linked from another page, so if there is not any link linking to a page, the page will not have any CoolURI.

If we feel comfortable about this, the transformation config becomes a bit less complicated as only params to cool transformation options have to be set.

6.6.3 Aliases

Values such as page titles are stored in a database at most cases. If a title is used in a CoolURI, it cannot be used in a normal form – with national characters, spaces, interpunction etc. Therefore without caching, the same value as is used in a URI has to be saved with the title in the database. This value is called alias. But with the caching enabled, the alias is generated during the translation automatically and used in the URI.

6.6.4 More look up fields

In case an alias is required, the only database field, where a value is looked up is the alias field. On the other hand, without cache, there is no alias and no need to limit the fields

where the value is looked up to only one. When multiply fields are set, a value in the first non-empty is used in a URI.

6.6.5 Auto update

If links were retrieved from the cache every time, it would not be possible to change a link. Therefore it is needed to do a check whether the link has been changed or not sometimes. Doing this check every time parameters to link transformation is requested would suppress the sense of cache into retrieving a CoolURI only. So it is possible to set a number of days between checks.

6.6.6 Old links redirection

When a cached link is changed, the old one is removed from the cache and inserted to another database table, where such links are stored. Then, when a requested URI is not found in cache but in this "old links" table, the request is redirected to a new URI, as the old links save a reference to their new versions. It is possible to redirect only for a short period of time.

6.6.7 Sticky

Links in the cache have an attribute called *sticky* which says whether this link can be updated or not. This is useful in a case when a link has been changed manually and we do not want it to be changed back in its next update.

6.7 Link manager

This manager is a tool that allows us to manipulate with cache and old links. Therefore cache must be enabled in order to make this manager work. Since cache and old links are normal database tables, the manager is some sort of table administration. It can create, read, update and delete data from both tables.

The link manager is prepared to become a module of any CMS or some kind of application administration. A prove of this functionality is the CMS Typo3 BackEnd module.

6.7.1 Classes

The manager is in fact only one main class and one authentification class. But it use classes from the transformer such as the database layer or the functions class.

The authentization class LinkManager_UserAuth (file *linkmanager.UserAuth.php*) is meant to decide whether an user has rights to access the manager. If the user is authorized, the static function isLogged() returns true. This function has to be implemented based on each application needs. In the included demo it always returns true.

The main class is called LinkManger_Main and is placed in the *linkmanager.Main.php* file. It has only two public methods. The first one menu() returns menu tree and the second one main() returns contents. Both functions return directly HTML code.

Beside the two methods, the class has a constructor method with two parameters:

- *file* – file in which the class is included, it is important because of internal links to the manager's parts,
- *conf* – path to the XML configuration file, the same that is used in the transformer.

6.7.2 Demo

The demo is a simple application showing the Link manager working and can be also used as a reference for creating modules. It is placed in the *manager* directory along with the two manager classes.

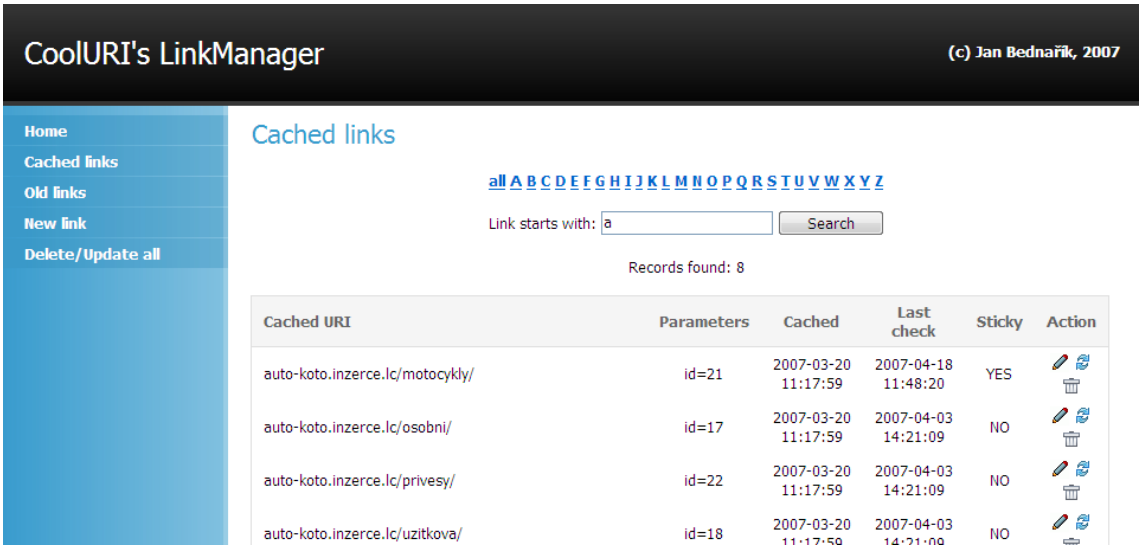

Figure 2: Included demo of the Link manager.

7 Configuration

The whole configuration is stored in a XML file. At default this file's name is *CoolUriConf.xml*, if the name is different, its name has to be passed to the instancing function (6.4.1.1) as a parameter.

Reasons for the configuration file to be a XML are quite obvious. It is portable – XML can be used on many platforms and with many programming languages. Anyway, this is being demonstrated here as a PHP solution and a Java solution are available, while both can use the same configuration file. Another reason for the XML is that XML is well-known standard and does not need to be brought closer.

The XML is defined by its XML Schema[18] which can, instead of DTD[19], be more descriptive. If the configuration XML follows the rules set in the XML Schema, that means the XML is valid, can be found out on *http://tools.decisionsoft.com/schemaValidate/* for example.

7.1 Composing the XML

Before we start to explain what elements the configuration XML contains, let's have a look at it from a wider point of view. Although the XML will be created by rewriting another one in most cases, it should be made clear where to start when creating it.

Let us begin with just a brief look at most important elements. A user-guide to these elements will follow immediately. This section should be useful to make a good idea about the XML at whole, so it would be understood how to put bits and pieces together.

Every XML must contain a root element. Here it is called cooluri.

Then the transformation has to be allowed by element cooluris. The name of the elements says that we really want CoolURIs.

If a CoolURI is not always passed to the cool2params function (6.4.1.2), we can specify in the element uri where the CoolURI can be taken from.

Array of parameters is always returned from the cool2params function (6.4.1.2). Using element savetranslationto can be also specified in which variable should the array be saved besides returning it.

Sections of a URI which should be removed before processing the URI are set in the removeparts elements. These are the sections discussed in 6.5.1.

So much for the preprocessing, postprocessing of a CoolURI before returning it from the params2cool function include appending prefix (urlprefix) or suffix (urlsuffix) to the URI or removing a trainling slash (removetrailingslash) so files would be used in URIs rather than directories (4).

Everything concerning cache is located in the cache element.

Characters which split the URI into an array of values are defined in pathseparators element. Even more that one separator can be defined.

Transforming options of the URI parts are defined for each type (2.1) in a separated way:

[18] http://www.w3.org/TR/xmlschema-0/

[19] http://www.w3schools.com/dtd/default.asp

Unwanted values – defined in removeparts element,

Valuemaps – element valuemaps,

Predefined parts – element predefinedparts,

Pagepath – element pagepath,

Other parts – other, but still parts of the URI, element uriparts,

Static values – these do not have their own element, but they are part of the uriparts element.

The final order of translated parts when generating a CoolURI from parameters can be specified too. Either order of part-types (2.1) or order of each parameter can be set. Element paramorder defines the order of each parameter. If this element is not set, or does not contain all parameters, part-types order is used to sort all or the rest. The default value can be overriden using element partorder.

7.2 Whitespace

If a *xs:token* value is required, whitespace should not be added before or after the value. When parsing the XML configuration, whitespaces are not trimmed and therefore it could cause trouble.

7.3 Datatypes

There are a few elements defined in the configuration, which are a sort of a datatype. These elements, or elements following rules of a datatype, can be found all over the configuration.

7.4 Boolean datatype

The boolean type is limited only to value 1 as true or 0 as false.

```
<xs:simpleType name="boolean">
  < xs:restriction  base="xs:string">
    <xs:enumeration value="1"/>
    <xs:enumeration value="0"/>
  </ xs:restriction >
</xs:simpleType>
```

Listing 8: XML Schema of the boolean datatype

7.5 Regexp and SQL datatypes

Regexp and SQL datatypes are renamed string only. XML Schema does not have built-in datatypes for regular expressions or SQL. Therefore, to indicate that a value should contain not any string, but a string following some rules, these two datatypes are set. However, no verification procedure that a string is a valid regular expression or SQL is run.

```
<xs:simpleType name="sql">
  < xs:restriction  base="xs:string"/>
</xs:simpleType>
<xs:simpleType name="regexp">
  < xs:restriction  base="xs:string"/>
</xs:simpleType>
```

Listing 9: XML Schema of the regexp and SQL datatype

7.6 Constraint datatype

Constraints are used to limit data to specific type, range or value. A value can be even matched against a regular expression[20].

An element of the constraint datatype has to contain an element match, type or (and) element sequence compare and value. The type element can be combined with the sequence. A value is matched against a regular expresion in the match element and a result is returned. When the type is used, a value's type is checked against a type in the element (types int, float and string are supported). Than the value is compared with the value in the value element. The comparison operator is defined in the element compare and can be one of these: lte (<=), gte (>=), lt (<), gt (>), eq (=), neq (<>).

```
<xs:complexType name="constraintType">
  <xs:all>
    <xs:element name="match" type="regexp" minOccurs="0"/>
    <xs:element name="type" minOccurs="0">
      <xs:simpleType>
        < xs:restriction  base="xs:string">
          <xs:enumeration value="int"/>
          <xs:enumeration value="float"/>
          <xs:enumeration value="string"/>
        </ xs:restriction >
      </xs:simpleType>
    </xs:element>
    <xs:element name="compare" minOccurs="0">
      <xs:simpleType>
        < xs:restriction  base="xs:string">
          <xs:enumeration value="lte"/>
          <xs:enumeration value="gte"/>
          <xs:enumeration value="lt"/>
          <xs:enumeration value="gt"/>
```

[20] preg_* functions are used in the PHP solution, therefore regular expressions must be Perl compatible (*http://www.php.net/manual/en/ref.pcre.php*).

```
            <xs:enumeration value="eq"/>
            <xs:enumeration value="neq"/>
          </ xs:restriction >
        </xs:simpleType>
      </xs:element>
      <xs:element name="value" type="xs:string" minOccurs="0"/>
    </ xs:all >
  </xs:complexType>
```

Listing 10: XML Schema of the constraint datatype

Example, when a value has to be a positive integer:

```
<type>int</type>
<compare>gt</compare>
<value>0</value>
```

Listing 11: Constraint datatype example

7.7 Lookindb datatype

This datatype is used when a value is looked up in a database. The element to is the only one required, as it is used when transforming from parameters *to* a CoolURI. The other direction, defined by the element from is not required because transformation from a CoolURI can be substituted by looking up parameters in cache only. Therefore, if cache is not enabled, this parameter is required.

Both elements must contain a SQL query which should return only one value or only the first value is used in the result. A value on which the query is based (value in a *where* clause) is identified in the SQL by $1 sign[21].

In order not to perform unnecessary queries and so not to uselessly load a SQL server, elements translatetoif and translatefromif can define constraints when the query can be performed and when not. This is decided so that the value which is being translated is passed to the constraint function. Therefore these two elements are of the constraint datatype (7.6).

Next available element in this datatype – urlize – can contain a boolean value (7.4). If true, a value retreived from a database is run through the URLize function, which transforms all non-ASCII character to ASCII, substitutes special characters and spaces with hyphens and so on, so the value can be used in a URI. Urlizing should not be carried out if cache is not enabled. If it was, transformed values would be looked up in the database, but would not be found of course.

It can happen that on some servers the URLize function does not work as expected. So there is an alternative function for urlizing called "sanitizing" activated by setting the element sanitize to 1.

```
<xs:complexType name="lookindbType">
  <xs:all>
    <xs:element name="from" type="sql" minOccurs="0"/>
```

[21] This sign is used because it is common in regular expressions as a representative for a first match.

```
    <xs:element name="to" type="sql"/>
    <xs:element name="translatetoif" type="constraintType" minOccurs="0"/>
    <xs:element name="translatefromif" type="constraintType" minOccurs="0"/>
    <xs:element name="urlize" type="boolean" minOccurs="0"/>
    <xs:element name="sanitize" type="boolean" minOccurs="0"/>
  </xs:all>
</xs:complexType>
```

Listing 12: XML Schema of the lookindb datatype

Example of the lookindb datatype, where a value from a database table *region* is looked up. But the transformation to a CoolURI is carried out only if the value matches the regular expression ^[0−9]+$.

```
<from>SELECT id FROM region WHERE urlname='$1'</from>
<to>SELECT urlname FROM region WHERE id=$1</to>
< translatetoif >
  <match>^[0−9]+$</match>
</ translatetoif >
```

Listing 13: Constraint lookindb example

7.8 Part datatype

Elements of this type are used when defining part transformations of a URI. Types of parts have been listed in 6.5. This datatype is quite complex, but not all of its elements and attributes are always used. A list of applicable elements and attributes will be given with each part's definition later.

```
<xs:element name="part">
  <xs:complexType mixed="true">
    <xs:all>
      <xs:element name="lookindb" type="lookindbType" minOccurs="0"/>
      <xs:element name="parameter" type="xs:token" minOccurs="0"/>
      <xs:element name="value" type="xs:token" minOccurs="0"/>
      <xs:element name="userfunc" type="xs:token" minOccurs="0"/>
    </xs:all>
    <xs:attribute  name="notrequired" type="boolean" use="optional" />
    <xs:attribute  name="static" type="boolean" use="optional" />
    <xs:attribute  name="key" type="regexp" use="optional" />
    <xs:attribute  name="regexp" type="boolean" use="optional" />
    <xs:attribute  name="pagepath" type="boolean" use="optional" />
    <xs:attribute  name="after" type="xs:token" use="optional" />
  </xs:complexType>
</xs:element>
```

Listing 14: XML Schema of the part datatype

Attributes not listed here are specific to a single element and therefore will be described with this element.

7.8.1 Parameter element

Name of a parameter which belongs to the part. Element with this parameter should be unique.

7.8.2 Value element

Value used in a CoolURI. If the value is looked up in a database, the value of this element is used in the database query. In some cases, instead of a static value, reference to a regular expression match is used.

7.8.3 Userfunc element

Sometimes out of all options available none are sufficent. If such thing happens, we can define our own function. But user functions can be used only when the cache is enabled.

Value of this element is the function's name. If the function is a class method, the method can be called as classname−>methodname. Two parameters are passed to this function, first one is the whole XML element, second one is a value of this element.

Userfunc element is used in the Typo3 extension, because of Typo3's database structure, pagepath needs to be looked up in 3 tables[22].

Depending on a location of this element it should return a single value or an array of values. This will be discussed later.

```
function getPageTitle($conf,$value) { ... }
```

Listing 15: Example of a function used in an userfunc element

7.8.4 After attribute

At default URI parts are separated with slash. In specific cases we may want to put something different than slash after a part. This attribute can specify this.

7.9 Root element

The root element has already been mentioned, but to keep order, let's repeat it: The root element is called cooluri and its XML schema is following:

```
<xs:element name="cooluri">
  <xs:complexType>
    <xs:all>
      <xs:element name="cooluris" type="boolean"/>
      <xs:element ref="uri" minOccurs="0"/>
      <xs:element name="savetranslationto" type="xs:token" minOccurs="0"/>
      <xs:element ref="pathseparators" minOccurs="0"/>
      <xs:element name="urlprefix" type="xs:token" minOccurs="0"/>
      <xs:element name="urlsuffix" type="xs:token" minOccurs="0"/>
```

[22]For those who are acquainted with Typo3 – these tables are pages, pages_language_overlay and sys_template.

```
      <xs:element name="removetrailingslash" type="boolean" minOccurs="0"/>
      <xs:element ref="removeparts" minOccurs="0"/>
      <xs:element ref="cache" minOccurs="0"/>
      <xs:element ref="uriparts" minOccurs="0"/>
      <xs:element ref="pagepath" minOccurs="0"/>
      <xs:element ref="predefinedparts" minOccurs="0"/>
      <xs:element ref="valuemaps" minOccurs="0"/>
      <xs:element ref="paramorder" minOccurs="0"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Listing 16: XML Schema of the root element cooluri

The minOccurs attribute suggest that all elements except one are optional. However, without at least half of them, the transformation would do nothing, or more precisely it would transform parameters not to a CoolURI but to a DirtyURI and that is not the best result.

7.10 Cooluris element

The only obligatory first level element turns on or off the transformation. On/off means that it can contain only a boolean value. Turning off the transformation can be useful during development or when an application is moved to a server, that does not allow rewriting.

```
<xs:element name="cooluris" type="boolean"/>
```

Listing 17: Inline XML Schema of the cooluris element

7.11 Uri element

This element defines from where the transformer should retrieve a URI for transformation. If the URI is passed to the transformer as a function parameter, this element does not need to be set. Otherwise it can define, using element var, in which variable can be the URI found. If this variable is an array, elements part can define this array's keys. If there are more than one parts, the URI is concatenated from the array's values in an order set be the order of the elements.

```
<xs:element name="uri">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="var" type="xs:token"/>
      <xs:element name="part" type="xs:token" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 18: XML Schema of the uri element

7.11.1 Var element

For description of this element see above.

In PHP, the value is search among all variables available in the $GLOBALS array. But most probably, the the URI will be composed from values in the $_SERVER array. Be aware, that the variable name is set without the leading dollar sign.

7.11.2 Part element

For description of this element see above.

REQUEST_URI is the most probable value of this element as $_SERVER['REQUEST_URI'] contains the whole request URI without a server name.

7.11.3 Example

Retrieving value from $_SERVER['HTTP_HOST'] and $_SERVER['REQUEST_URI'] variables. That means, for an example, from a request URI *http://www.example.com/some/page/* a value *www.example.com/some/page* will be retrieved.

```
<uri>
  <var>_SERVER</var>
  <part>HTTP_HOST</part>
  <part>REQUEST_URI</part>
</uri>
```

Listing 19: Uri element example

7.12 Savetranslationto element

Translated parameters array can be saved into a variable defined by this element. It is recommended to use one of the superglobal variables[23] such as $_REQUEST or $_GET. But again, the variable name must be set without the leading dollar sign.

If the variable contains some values, those will not be lost, as the traslated parameters array is merged with the variable, therefore overwriting only duplicate keys, others are left untouched.

```
<xs:element name="savetranslationto" type="xs:token" minOccurs="0"/>
```

Listing 20: Inline XML Schema of the Savetranslationto element

Example of saving parameters to the $_REQUEST array:

```
<savetranslationto>_REQUEST</savetranslationto>
```

Listing 21: Savetranslationto element example

[23] http://php.net/variables.predefined

7.13 Pathseparators element

This element contains a list of characters used to split a URI or only a path to a document, or even better the value retrived using the uri element, into parts. If this element is not present default value – slash "/" – is used.

```
<xs:element name="pathseparators">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="separator" type="xs:token" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 22: XML Schema of the Pathseparators element

7.13.1 Separator element

Pathseparators' subelements define the separators. These are concatenated into a single string which is passed to the preg_split function. Therefore it is possible that the separator will not be a single character, but a list of characters or even a character range (0−9). This is not recommended and even it is not useful at all.

7.13.2 Example

Separating a variable with "." and "/" characters:

```
<pathseparators>
  <separator>.</separator>
  <separator>/</separator>
</pathseparators>
```

Listing 23: Pathseparators element example

7.14 Urlprefix and urlsuffix elements

Strings in these elements are added to a URI just before returning. That means they are not cached, therefore be careful not to pass a URI containing these fixes to the cool2params function.

```
<xs:element name="urlprefix" type="xs:token" minOccurs="0"/>
<xs:element name="urlsuffix" type="xs:token" minOccurs="0"/>
```

Listing 24: Inline XML Schema of the fixes elements

Example of prefixing URI with http://:

```
<urlprefix>http://</urlprefix>
```

Listing 25: Urlprefix element example

7.15 Removetrailingslash element

The directory approach to URIs is used at default (4). But if the file approach is preferred, it is necessary to remove the trailing slash by setting a value of this element to 1. It can be used in combination with suffixing *.html* to every URI.

```
<xs:element name="removetrailingslash" type="boolean" minOccurs="0"/>
```

Listing 26: Inline XML Schema of the removetrailingslash element

7.16 Cache element

Cache element contains configuration of the cache. Because it is a bit complex, its description is divided into a few sections. Settings referring to transforming a CoolURI to parameters are set in the cool2params element. On the contrary, the other direction's settings are in the element called params2cool. What will happen when a URI is not found in the cache is defined in the element pagenotfound.

```
<xs:element name="cache">
  <xs:complexType>
    <xs:all>
      <xs:element name="usecache" type="boolean"/>
      <xs:element name="tablesprefix" type="xs:token" minOccurs="0"/>
      <xs:element name="cacheparams" type="boolean" minOccurs="0"/>
      <xs:element ref="params2cool" minOccurs="0"/>
      <xs:element ref="cool2params" minOccurs="0"/>
      <xs:element ref="pagenotfound" minOccurs="0"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Listing 27: XML Schema of the cache element

7.16.1 Usecache element

The cache can be easily turned on and off by the boolean element usecache. Again, this can be useful during an application's development when we tamper with settings, URIs etc. The same effect would be achieved by deleting the whole cache element, but setting 1 or 0 is a bit easier.

7.16.2 Tablesprefix element

To prevent name collisions between cache database tables and other tables, a table prefix can be set using subelement tablesprefix. Default value of this element is *link_*.

7.16.3 Cacheparams element

Element cacheparams enables caching of parameters (part of a URI after a question mark). This option is disabled at default and it is recommended to stay that way. It is not usual to the check number of parameters or their names, as it can vary.

7.17 Cool2params subelement

It is obvoius that this cache's subelement refers to the transforming direction from a CoolURI to a list of parameters.

```
<xs:element name="cool2params">
  <xs:complexType>
    <xs:all>
      <xs:element name="translateifnotfound" type="boolean"/>
      <xs:element name="oldlinksvalidfor" type="xs:unsignedInt"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Listing 28: XML Schema of the cool2params element

7.17.1 Translateifnotfound element

This option says whether the transformer should try to retrieve parameters when a URI is not found in the cache. It can be enabled only if the transformer is able to work also without the cache at all. That means, if getting parameters is dependent on the cache, this option cannot be turned on.

7.17.2 Oldlinksvalidfor element

As has been already said, old links are those which have been replaced by another CoolURI and have been moved to an old links database table. If a URI is not found in the cache, but in this table, a request is redirected to the new URI. But this redirection need not to be pernament. A number of days when this redirection happens since moving to the old links table can be set in this element. To make it pernament, just set it to a really big number.

7.18 Params2cool subelement

Element referring to parameters to a CoolURI transformation has only one subelement so far.

```
<xs:element name="params2cool">
  <xs:complexType>
    <xs:all>
      <xs:element name="checkforchangeevery" type="xs:unsignedInt"/>
    </xs:all>
  </xs:complexType>
```

```
</xs:element>
```

Listing 29: XML Schema of the params2cool element

7.18.1 Checkforchangeevery element

CoolURIs do not change but can change. If some values of a URI are composed from changed data, the URI does not change immediately, as it is retrieved from the cache. To make it possible, to change a URI, it is necessary to check sometimes. This element holds a number which says how many days should elaps between each check. When an interval in days from a URI's last check is longer than the number, the URI is not retrieved from the cache, but generated as if it was not even in the cache. If the URI is changed, the old value is moved to the old links table and updated by the new one.

If we want to skip the cache and generate a URI every time, just set the number to 0. On the other hand, if it is big enough, the check would never happen.

7.19 Pagenotfound subelement

What will happen when a page (URI) is not found in the cache defines this element. Basically two things need to be done. A right status code needs to be sent to the browser and some kind of message with it. Sometimes instead of sending a message a request can be redirected to another page.

```
<xs:element name="pagenotfound">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="status" type="xs:string"/>
      <xs:element ref="behavior"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 30: XML Schema of the pagenotfound element

7.19.1 Status element

A value of this element is a HTTP version and a complete status code with its definition as were defined by WWW Consorcium [10]. However, from all those possible status variants, only 3 are useful in this situation.

301 Moved Permanently – when redirecting,

303 See Other – when redirecting,

404 Not Found – when sending a message.

For more information on whether to use 301 or 303 see [10].

7.19.2 Behavior element

Behavior when a page is not found in the cache is defined as well as the status. The behavior type is defined by the attribute type which can contain one of three values:

message – a simple message, such as "Page not found";

page – a filename of a HTML page its contents will be echoed;

redirect – a URI to which a request will be redirected.

```
<xs:element name="behavior">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        < xs:attribute  name="type" type="behaviorType"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>

<xs:simpleType name="behaviorType">
  < xs:restriction  base="xs:string">
      <xs:enumeration value="message"/>
      <xs:enumeration value="redirect"/>
      <xs:enumeration value="page"/>
  </ xs:restriction >
</xs:simpleType>
```

Listing 31: XML Schema of the behavior element

7.19.3 Example

Sending 404 status code along with a simple message.

```
<pagenotfound>
  <status>HTTP/1.0 404 Not Found</status>
  <behavior type="message"><![CDATA[ <h1>Page not found!</h1> ]]></behavior>
</pagenotfound>
```

Listing 32: Example of the pagenotfound element

7.20 Cache example

```
<cache>
  <usecache>1</usecache>
  <params2cool>
    <checkforchangeevery>1</checkforchangeevery>
  </params2cool>
  <cool2params>
```

```
    <translateifnotfound>0</translateifnotfound>
    < oldlinksvalidfor >365</oldlinksvalidfor>
  </cool2params>
  <cacheparams>0</cacheparams>

  <pagenotfound>
    <status>HTTP/1.0 404 Not Found</status>
    <behavior type="message"><![CDATA[ <h1>Page not found!</h1> ]]></behavior>
  </pagenotfound>
</cache>
```

Listing 33: Example of the cache element

7.21 Removeparts element

Parts of a URI that have to be removed before the URI can be translated to parameters. These parts are listed in subelements part of this element. If something is removed, it is possible to redirected a request to a URI without the removed parts. This prevents duplicities, but can cause circular redirecting. Therefore the redirect has to be allowed using the boolean attribute redirect.

```
<xs:element name="removeparts">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="part" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute  name="redirect" type="boolean" use="optional" />
  </xs:complexType>
</xs:element>
```

Listing 34: XML Schema of the removeparts element

7.21.1 Part element

This element is of the part datatype, but uses only regexp attribute, that says a value is a regular expression.

The value is look up in a whole URI and if found is removed. In case of the regular expression, whole URI is matched against it and found matches are removed.

7.21.2 Example

Removing index.php from a URI.

```
<removeparts>
  <part>index.php</part>
</removeparts>
```

Listing 35: Example of the removeparts element

7.22 Valuemaps element

This element contains an unbounded number of valuemap elements. Valuemap is a list of values and theirs mapping to parameter values. That means when a parameter has a limited number of values and each value has a synonym in a CoolURI value, this parameter is a valuemap.

A value map seems to be a similarity in looking up values in a database. That is true, but valuemaps are meant for small sets of values, where a database table would be an overhead.

```
<xs:element name="valuemaps">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="valuemap" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="valuemap">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="parameter" type="xs:token"/>
      <xs:element ref="value" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 36: XML Schema of the valuemaps and valuemap elements

7.22.1 Parameter element

Parameter element defines a name of a parameter that identifies this valuemap.

7.22.2 Value element

Attribute **key** characterizes value used in a CoolURI which is a synonym to a value of this element.

```
<xs:element name="value">
  <xs:complexType mixed="true">
    <xs:attribute  name="key" type="xs:token"/>
  </xs:complexType>
</xs:element>
```

Listing 37: XML Schema of the value element

7.22.3 Example

Value of a parameter lang is an integer identifier of a selected language. But in a URI the language should be identified by its standardized code. Zero is a default value which is not identified in any way.

```
<valuemaps>
  <valuemap>
    <parameter>lang</parameter>
    <value key="">0</value>
    <value key="en">1</value>
    <value key="de">2</value>
    <value key="it">3</value>
  </valuemap>
</valuemaps>
```

Listing 38: Example of the valuemaps element

7.23 Predefinedparts element

It is an element which envelops a list of part elements. Predefined part is a part which can be recognized according to a key. The key can be a simple value or a regular expression. From this key is extracted a parameter. A value based on this parameter can be looked up in a database.

Element part is of a datatype part, it can contain all subelements of this datatype, but some of its attributes are useless here.

Key attribute can contain a regular expression, but only with one bracketed expression. If subelement value contains $1, it is replaced by the expression. If the key is a regular expression, it must be identified by attribute regexp set to 1.

User function in this element must return single value, which will be used in a CoolURI afterwards.

7.23.1 Example

- If a part of a URI is a word *hidden*, parameter *displayhidden* is set to 1;
- Parameter *n* is set to a number that is prefixed in a URI with *page-*;
- Values that begin with *page-* are looked up in a database table named *region*.
- A parameter which is thrown away

```
<predefinedparts>
  <part key="hidden">
   <parameter>displayhidden</parameter>
   <value>1</value>
  </part>
  <part key="page-([0-9]+)" regexp="1">
    <parameter>n</parameter>
    <value>$1</value>
  </part>
  <part key="region-(.*)" regexp="1">
    <parameter>region</parameter>
    <value>$1</value>
```

```
    <lookindb>
      <from>SELECT id FROM region WHERE urlname='$1'</from>
      <to>SELECT urlname FROM region WHERE id=$1</to>
      < translatetoif >
        <match>^[0-9]+$</match>
      </ translatetoif >
    </lookindb>
  </part>
  <part>
   <parameter>throwAway</parameter>
  </part>
</predefinedparts>
```

Listing 39: Example of the predefinedparts element

7.24 Pagepath element

This element defines how to retrieve a pagepath from a database table.

It is required that this table has a standard tree/hierarchical structure. That means each row has a unique identifier and an identifier of its parent row. Items on the first level has a parent identifier set to zero or some null value. If the cache is not used, a column in this table, which contains urlized value[24], is required.

For example, if there are triples *(id,parent_id,name)*: *(1,0,Electronics)*, *(2,1,Televisions)*, *(3,1,Portable)*, *(4,2,LCD)*, *(5,2,Plasma)*, *(6,3,MP3 players)* in a database, a pagepath can be *Electronics – Televisions – LCD*, or *Electronics – Portable – MP3 players*.

```
<xs:element name="pagepath">
  <xs:complexType>
    <xs:all>
      <xs:element name="saveto" type="xs:token"/>
      <xs:element name="default" type="xs:token"/>
      <xs:element name="table" type="xs:token" minOccurs="0"/>
      <xs:element name="id" type="xs:token" minOccurs="0"/>
      <xs:element name="alias" type="xs:token" minOccurs="0"/>
      <xs:element name="title" type="xs:token" minOccurs="0"/>
      <xs:element name="additionalWhere" type="sql" minOccurs="0"/>
      <xs:element name="connection" type="xs:token" minOccurs="0"/>
      <xs:element ref="start"  minOccurs="0"/>
      <xs:element name="idconstraint" type="constraintType" minOccurs="0"/>
      <xs:element ref="required" minOccurs="0"/>
      <xs:element ref="paramconstraints" minOccurs="0"/>
      <xs:element name="allparamconstraints" type="constraintType" minOccurs="0"/>
      <xs:element name="userfunc" type="xs:token" minOccurs="0"/>
      <xs:element name="sanitize" type="boolean" minOccurs="0"/>
    </xs:all>
  </xs:complexType>
</xs:element>
```

Listing 40: XML Schema of the pagepath element

Main part of the pagepath element defines SQL query for retrieving a pagepath.

[24] A value without punctuation, national charcters etc.

7.24.1 SQL query for a pagepath lookup

The table that contains the tree structure is set in the element table.

The name of a parameter that is translated to a pagepath is defined in the saveto element. On the other hand, the value of a parameter with a name set in saveto is retrieved from a pagepath when transforming from a CoolURI.

With the cache, field that contains a value for the pagepath can be defined in the element title . This element can contain multiple column names separated with comma. First non-empty column will be used for the pagepath. A found value will be run though the urlize function. Alternatively, if the element sanitize is set to 1, it will be run though the sanitizing function.

But without the cache, the column has to be set in the alias element and only one value is allowed. In addition this column must contain already urlized value.

The column containing the parent ids defines the element connection (because it connects nodes in the tree structure).

The first level of the tree, the point where the lookup gets to the tree root, defines the element start. It has 2 subelements – param contains column name and when in a query result this column has a value equals to value in the value subelement, the lookup ends. Or it may start here as well, it depends on the transformation direction.

If required, additional conditions to the *where* part of a pagepath query can be set using element additionalWhere.

```
<xs:element name="start">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="param" type="xs:token"/>
      <xs:element name="value" type="xs:token"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 41: XML Schema of the start element

7.24.2 Composing a final value

Looking up a pagepath from a parameter starts from a node of a tree where the node has the same identifier as is the parameter. Then the lookup continues based on the connection value until a composed query returns empty result or until the tree's first level defined in the start is reached. This method returns several values which are composed into a URI.

In the reverse direction – from a URI to parameters – sections of a pagepath are recognized (based on the contents of the uriparts element, which is not part of this pagepath element) at first. A way through the tree is found depending on the sections. A last node's identifier is the wanted value – value that identifies a current node in the tree.

If no parameter from a pagepath is retrieved, value in the default element is set to the saveto parameter.

7.24.3 Required element

The uriparts element will be described later, but required strongly depends on it. Parameters in the uriparts can be assigned with a value from a pagepath. Let us give an example: A CoolURI looks like *first.example.com/second/third*, where *first* to *third* are pagepath levels. Now a URI *example.com/some/value* does not clearly contain a pagepath because it lacks the first level and that is not possible. So a pagepath exists only if the first URI part contains a value. Which part is required in order to the pagepath exist can be set using the required element.

This element contains a list of parameters in param subelements, which have to exist in the uriparts element.

```
<xs:element name="required">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="param" type="xs:token" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 42: XML Schema of the required element

7.24.4 Constraining a pagepath

The lookup from parameters can end sooner than at the point defined by start. The translation from parameters does not even start if a saveto parameter does not pass a constraint qualified in the idconstraint element.

Sections of the pagepath can be saved to various parameters when retrieving them from a CoolURI. This parameters in which the pagepath is saved are defined in the uriparts element. A retrieval of the saveto parameter can be limited to a constrain when a uripart has to be set in order to the saveto parameter exist. This limitation is defined in required element which lists parameters from the uriparts element in part elements.

Composition of the final saveto value can be sucessful only if all parts of the looked up pagepath pass a constraint in the allparamconstraints element.

If several parts should not pass the allparamconstraints, but some other constraints, those can be defined in paramconstraints element. This element contains a list of paramconstraint elements, which are of the constraint datatype. Each element must have an attribute named param, which refers to a parameter in uriparts element.

When iterating through tree, looking for a pagepath, every identifier found has to pass a constraint defined in the idconstraint element.

```
<xs:element name="paramconstraints">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="paramconstraint" type="paramconstraintType" maxOccurs="unbounded" /
          >
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:complexType name="paramconstraintType">
  <xs:complexContent>
    <xs:extension base="constraintType">
      <xs:attribute  name="param" type="xs:token"/>
```

```
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Listing 43: XML Schema of the paramconstraints element

7.24.5 Userfunc element

When caching, the pagepath can be retrived using a user defined function. The function name is set in the userfunc element and the function has to return an array in which the first element is last page on the path. That means this array is reversed against the pagepath.

Elements saveto and **default** has to be set when a user function is defined.

7.24.6 Example

An example of this element would not be comprehensible at this point. It will be presented in the case-studies section (8).

7.25 Uriparts element

Uriparts are parts that are left after predefined and valuemaps are processed. That means their value is not from a small set of values and the parts can be recognized by any sort of a regular expression. Uriparts element is again just a wrap for a list of part elements, where each element defines a single uripart.

```
<xs:element name="uriparts">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="part" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 44: XML Schema of the uriparts element

The part is of the dataype part. In contradistinction to the other URI parts, this one can have most attributes and can contain all elements, which are defined in the datatype. Not used attributes are key and regexp as there two recognize a part in a URI, but these parts cannot be recognized, they have to be tested.

Subelements of this element, defined by their datatype, are well known and their meaning is the same here. Therefore there is no need to describe them again. However, some attributes are new here.

7.25.1 Static attribute

If a part is set to be static, by setting this attribute to true, the only subelement it needs is value. The value is used in a URI. Other elements are useless.

These parts are important when splitting the pagepath with some static values is needed. In the example in 7.24.3 static parts would be *example* and *com*.

7.25.2 Notrequired attribute

This attribute defines that this part is not required. That means it can be missing in a URI. However, element with this attribute set to true must be followed by an element with the attribute static set to true as well.

The example in 7.24.3 is appropriate here as well. The value *first* should be set to notrequired, because it does not have to be in a URI as presents the second URI in the example.

7.25.3 Pagepath attribute

Part with this attribute set to true is a part of a pagepath. That means, when a pagepath is found, this part will become a value of one level from it. On the other hand, when retrieving an identifier from a pagepath in a URI, this element will tell that on its position is probably a part of a pagepath.

7.25.4 Example

This example requires cache enabled, for an example on pagepath without cache, look at the case-studies section.

Here is a parameter *newsId* looked up in a database and put into a URI:

```
<uriparts>
  <part>
    <parameter>newsId</parameter>
    <lookindb>
      <to>SELECT title FROM news WHERE id=$1</to>
      < translatetoif >
        <match>^[0-9]+$</match>
      </ translatetoif >
      <urlize>1</urlize>
    </lookindb>
  </part>
</uriparts>
```

Listing 45: Example of the uriparts element

7.26 Partorder element

Order of the URI parts can be set using this element. It has to hold exactly 4 subelements part. Each of them must contain one value out of these 4:

- pagepath
- uriparts
- valuemaps
- predefinedparts

These values refer to elements of the same name and therefore define a succession of the parts in a URI.

```
<xs:element name="partorder">
  <xs:complexType>
    <xs:sequence>
```

```
      <xs:element name="part" type="xs:token" maxOccurs="4" minOccurs="4" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 46: XML Schema of the partorder element

If this element is not present, the default order is the same as defined by the following XML.

```
<partorder>
  <part>pagepath</part>
  <part>uriparts</part>
  <part>valuemaps</part>
  <part>predefinedparts</part>
</partorder>
```

Listing 47: Example of the partorder element

7.27 Paramorder element

When a neater ordering than the one provided by partsorder is required, element paramorder can do this. A list of its subelements param defines order upon parameters.

```
<xs:element name="paramorder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="param" type="xs:token" maxOccurs="unbounded" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Listing 48: XML Schema of the paramorder element

Not every parameter needs to be listed here. Those that are not listed are ordered upon the partorder settings and attached after listed.

Parameter with a name lang will be first value in a URI, followed by a parameter with a name newsId:

```
<paramorder>
  <param>lang</param>
  <param>newsId</param>
</paramorder>
```

Listing 49: Example of the paramorder element

7.28 Differences in the Java implementation

7.28.1 Saving parameters

Because Java does not have anything like the superglobal variables in PHP, there is no way how to set retrieved parameters for global access. So the cool2params method returns Map. Another way to access parameters is to use get method of the Translate class. This means that the savetranslationto element is useless in Java.

7.28.2 Retrieving URI

Java cannot access any variables that are in the global space of a script, so retrieving a URI is a bit limited. There is no use for the subelement var of the uri element, and subelements part of this element can containing only values QUERY_STRING, REMOTE_HOST, SCRIPT_NAME, SERVER_NAME, HTTP_* (where * stands for any string). Relevant methods of the *HttpServlet-Request* class are called in order to get a value.

7.28.3 User functions

Where a PHP function uses array, Java uses List. This means a pagepath's user function has to return List.

7.28.4 Sanitizing

As is supposed that urlizing function will work correctly on all Java servers, there is no need for alternative. So, there is no sanitizing function.

7.28.5 Status

Subelement of the element pagenotfound cannot containg a whole HTTP status message, but only the number code. It is because Java has these messages predefined and the method sending a status requires an interger parameter.

8 Case-studies

This section will present two examples of a real implementation of this work. One is a Czech advertisement server and second one is a Typo3 extension. Both use the same source codes, but with a different XML configuration. So, the configuration will be the main focus here.

8.1 Euroinzerce.cz

This server is about advertising and there is a huge competition among web servers in this field. Therefore every bit of a search engine optimization is needed in order to get the server to top places. So CoolURIs are very necessary.

8.1.1 CoolURI requirements

It was demanded that a first level of a tree of categories will create a 3^{rd} level domain, that means a page listing advertisements placed in the first level would have a URI like *http://computers.euroinzerce.cz*. Next levels would be placed right after *cz* – for example *http://computers.euroinzerce.cz/pc/intel/*. Number of levels is limited up to 4, but a slight change in the configuration could allow higher number.

But there are pages on this server, that not only display categories. Hence it was needed to create URIs pointing to registration, ad detail, documents etc. It was decided that such pages would have URIs like *http://euroinzerce.cz/module/action/*, for example *http://euroinzerce.cz/documents/contact/*. So another requirement was to differ whether part of a URI right after the domain is a 2^{nd} level category on a module name.

8.1.2 Configuration

8.1.2.1 Getting a CoolURI Because the CoolURIs are part of the domain, it is required to get the URI from the server's variables HTTP_HOST and REQUEST_URI. So the configuration looks like:

```
<uri>
  <var>_SERVER</var>
  <part>HTTP_HOST</part>
  <part>REQUEST_URI</part>
</uri>
```

Listing 50: Euroinzerce.cz – retrieving a CoolURI

It is the same as this PHP code:

```
$uri = $_SERVER['HTTP_HOST'].$_SERVER['REQUEST_URI'];
```

Listing 51: Euroinzerce.cz – retrieving a CoolURI in PHP

8.1.2.2 Splitting a CoolURI Once the CoolURI is aquired, it needs to be splitted into parts, but not only with slash, but with dot as well.

```
<pathseparators>
  <separator>.</separator>
  <separator>/</separator>
</pathseparators>
```

Listing 52: Euroinzerce.cz – splitting a CoolURI

8.1.2.3 Valuemaps There is only one valuemap defined. A type of advertisements that are displayed is set when words *prodam* (sell) or *koupim* (buy) are in a URI. In both cases, a parameter adtype is set to 1 or to 2.

```
<valuemaps>
  <valuemap>
    <parameter>adtype</parameter>
    <value key="koupim">2</value>
    <value key="prodam">1</value>
  </valuemap>
</valuemaps>
```

Listing 53: Euroinzerce.cz – Valuemap

8.1.2.4 Predefined parts Paging in advertisement lists is done by having value *strana-X* (page-X) in a URI, where X is a page number. The page number is set and saved in a parameter n.

The other part defines from which region visitors want to display advertisements. Region name is in a database, where the name is saved along with the urlized name. This urlized name is prefixed with *region-* in a URI.

```
<predefinedparts>
  <part key="strana-([0-9]+)" regexp="1">
    <parameter>n</parameter>
    <value>$1</value>
  </part>
  <part key="region-(.*)" regexp="1">
    <parameter>region</parameter>
    <value>$1</value>
    <lookindb>
      <from>SELECT id FROM region WHERE urlname='$1'</from>
      <to>SELECT urlname FROM region WHERE id=$1</to>
      < translatetoif >
        <match>^[0-9]+$</match>
      </ translatetoif >
    </lookindb>
  </part>
</predefinedparts>
```

Listing 54: Euroinzerce.cz – Predefined parts

8.1.2.5 Uriparts Every translated part of the pagepath is saved to a different parameter. When there is no pagepath in a URI, values on the positions of the pagepath are saved to these parameters. They are named in sequence cid, sid, lid , tid , kid (kid is not part of the pagepath). There is always .euroinzerce.cz between cid and sid.

```
<uriparts>
  <part notrequired="1" after="." pagepath="1">
    <parameter>cid</parameter>
```

```
  </part>
  <part static="1" after=".">
    <value>euroinzerce</value>
  </part>
  <part static="1">
    <value>cz</value>
  </part>
  <part pagepath="1">
    <parameter>sid</parameter>
  </part>
  <part pagepath="1">
    <parameter>lid</parameter>
  </part>
  <part pagepath="1">
    <parameter>tid</parameter>
  </part>
  <part>
    <parameter>kid</parameter>
  </part>
</uriparts>
```

Listing 55: Euroinzerce.cz – Uriparts parts

8.1.2.6 Pagepath Pagepath here is the most complicated part, as it is separated by the domain. The category list is saved in a database table category with a primary key id and foreign key pid to it is predecessor (parents). Because URIs are not cached, or do not need to be, category name has to be saved urlized in the column uriname. First level of the category tree is defined by having pid equals to 0. When transforming from a CoolURI, result pagepath identifier is saved in a parameter id.

```
<pagepath>
  <table>category</table>
  <id>id</id>
  <alias>uriname</alias>
  <connection>pid</connection>
  <start>
    <param>pid</param>
    <value>0</value>
  </start>
  <idconstraint>
    <match>^[0-9]+$</match>
  </idconstraint>
  <saveto>id</saveto>
  <default>0</default>
  <required>
    <param>cid</param>
  </required>
  <allparamconstraints>
    <match>^[0-9]+$</match>
  </allparamconstraints>
</pagepath>
```

Listing 56: Euroinzerce.cz – Pagepath parts

8.1.3 Examples

Table 1: Parameters to CoolURI examples

Parameters	CoolURI
id=4	pocitace.euroinzerce.cz/
id=9	pocitace.euroinzerce.cz/notebooky/do-10-tisic/
adtype=1&id=4&region=4	pocitace.euroinzerce.cz/prodam/region-praha-1/
adtype=2&id=6&n=5	pocitace.euroinzerce.cz/notebooky/koupim/strana-5/
sid=logout	euroinzerce.cz/logout/
lid=kontakt&sid=doc	euroinzerce.cz/doc/kontakt/

8.2 Typo3 extension

The extension[25] is a sort of a bridge between Typo3 and this URI transformer. Typo3 uses a specific function called typolink to generate all front-end links, therefore it was needed to create a hook which would change return value of this function. On the other hand, for retrieving parameters, a method that would be called before any other processing was required as well. Fortunately, Typo3 offers ways to create these hooks, so no modifications to its core were needed.

As the Typo3 does not have any alias fields in database tables, there is no other way for retrieving parameters from a URI than using cache. Thanks to that, the basic configuration of the transformer is quite simple.

8.2.1 Page path

The biggest problem was with the pagepath, because the transformer supports pagepaths which are stored only in one table. Beside pages table, Typo3 stores page title translations in the table called pages_language_overlay and the first level in the page tree is resolved using a connection to a sys_template record.

The best way out of this was to define a special function, which retrieves the pagepath and returns it in an array. So the XML configuration looks like this:

```
<pagepath>
  < title >alias, subtitle , title </ title >
  <saveto>id</saveto>
  <default>0</default>
  <userfunc>tx_cooluri->getPageTitle</userfunc>
</pagepath>
```

Listing 57: Typo3 – Pagepath

The title element defines an order of fields in which a page title is looked up. The other elements are required.

[25] Other content management systems call this a module

8.2.2 Discarding parameters

Typo3 sometimes use more parameters than is required or parameters that are not good enough to be in a URI. So we have a strange situation, that there are parameters which need to be discarded. The subelement part of the predefinedparts can be used for this task:

```
<predefinedparts>
    <part>
      <parameter>no_cache</parameter>
    </part>
</predefinedparts>
```

Listing 58: Typo3 – Discarding the parameter no_cache

9 Conclusion

The aim of this work has been generating CoolURIs and retrieving parameters from them and it has been successfully fulfilled. Using the XML configuration, the transformer provides vast scope for transforming options and the two implementations (PHP and Java) prove that URIs do not need to change even when an application platform is changed.

Deep search on the internet has revealed that no such universal reusable tool has ever been made. Therefore, this whole work could prove itself as a huge contribution, aimed in particular at the world of search engine optimization. About hundred downloads of the Typo3 extension prove that the transformer works and is already being used.

Future development of the transformer can include extensions of the transforming options, although it is not clear, what else could be transformed. And another language platform support, besides PHP and Java, could be added. An excellent candidate for this is some from the .NET platform, most likely C#.

10 Literature

[1] Apache module mod_rewrite.
http://httpd.apache.org/docs/1.3/mod/mod_rewrite.html

[2] Apache 1.3 URL Rewriting Guide.
http://httpd.apache.org/docs/1.3/misc/rewriteguide.html

[3] Barners-Lee T.: Hypertext Style: Cool URIs don't change.
http://www.w3.org/Provider/Style/URI, 1998.

[4] Barners-Lee T.: Uniform Resource Identifier (URI): Generic Syntax.
http://www.gbiv.com/protocols/uri/rfc/rfc3986.html, 2005.

[5] Eye tracking in MSN Search: Investigating snippet length, target position and task types.
http://research.microsoft.com/research/pubs/view.aspx?type=technical+report &id=1243

[6] Fox V.: Official Google Webmaster Central Blog: Update to our webmaster guidelines.
http://googlewebmastercentral.blogspot.com/2006/10/update-to-our-webmaster-guidelines.html, 2006

[7] Google. http://www.google.com/search?q=SEO+on-page+factors

[8] Google. http://www.google.com/search?q=SEO+off-page+factors

[9] HTTP 404 – Wikipedia, the free encyclopedia.
http://en.wikipedia.org/wiki/404_error

[10] HTTP/1.1: Status Code Definitions.
http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

[11] Internet Marketing Blog: On-Page Factors Category.
http://www.web1marketing.com/blog/index.php/archives/category/seo/on-page-factors/, 2007

[12] ISAPI Server Extensions and Filters.
http://msdn2.microsoft.com/en-us/library/aa279394(VS.60).aspx

[13] ISAPI_Rewrite – URL Rewrite engine for IIS, ISAPI Filter for URL Rewriting, mod_rewrite for IIS. http://www.isapirewrite.com/

[14] Powell T. A., Lima J.: URL Rewriting – Search Engine Friendly URLs.
http://www.seoconsultants.com/articles/1000/urls.asp

[15] Search Engine Friendly URLs using ASP.NET (C#.NET).
http://www.theukwebdesigncompany.com/articles/search-engine-friendly-urls-asp.php

[16] Uniform Resource Name – Wikipedia, the free encyclopedia.
http://en.wikipedia.org/wiki/Uniform_Resource_Name.

[17] Webmaster Help Center – Webmaster Guidelines.
http://www.google.com/support/webmasters/bin/answer.py?answer=35769