Department of Informatics Faculty of
Mathematics, Physics and Informatics
Comenius University, Bratislava

Modular redesign of the blog.matfyz.sk portal

(Master thesis)

Martin Rejda

Thesis advisor: RNDr. Martin Homola

Bratislava, 2010

Modular Redesign of The blog.matfyz.sk Portal

Master thesis

Martin Rejda

COMENIUS UNIVERSITY, BRATISLAVA, SLOVAKIA
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
DEPARTMENT OF INFORMATICS

9.2.1 Informatics

Thesis advisor: RNDr. Martin Homola

BRATISLAVA 2010

Abstract

This work concern about analysis of original code of blog.matfyz.sk portal, proposition of new architecture and its implementation. In proposition of new architecture we used modular programming and several principles from software engineering domain. We have to create modular architecture regarding to different types of languages and technologies which are used in this web application.

During analysis we discuss about biggest issues of portal such as improper separation of classes or extended functionality implemented directly into portal in complicated way. In our proposition we introduce new architecture which contains key paradigms from modular programming domain applied onto different programming languages such as XSLT, XQuery or XPath and technologies like XML which does not allow easily modularize data or code.

At the end we presents implementation with class diagrams and methods of connection technologies and programming languages.

keywords: Weblogs, Modular design, XSLT, XML, XML Database

Contents

Chapter 1

Introduction

1.1 Background

Today, the weblogs are social phenomenon. It is medium which allows people to present their personal opinions and observations to the public. In spring of 2007, the blog portal `blog.matfyz.sk` was developed. Portal had two primary goals. First was research of usability of XML, XND and XSLT in real life with testing of several ranking algorithms. Second was to create portal for community at our faculty. This community is made of students, teachers and other employees. Blog portal is independent of school from beginning and content is not censored or modified until it counteract with netiquette or authors rights.

In autumn 2007, blog engine was firstly used in learning process at course *Modern Approaches to Web Design*. Students had to create their own blog with emphasis on valid HTML and CSS, usability, accessibility and understanding XML and XSL technologies. Today blog portal is used also in other courses for different purposes. However, it is still research platform and students added complex support for tags with ontology in behind. In future is planned support for extended collaborative editing over many attributes, not just an articles, suggestions for many actions and many other algorithms based on artificial intelligence and knowledge systems.

1.2 Motivation

During development and testing of new algorithms, we had many regressions. Regression happens when some function breaks in new version, however it was working in previous. Development was slow because of inappropriate code structure which was oriented by action type. We spent more time fixing bugs, learning and fixing structure than implementing new algorithms. Initially we had to implement support for extend collaboration. After few months we decided to rewrite portal from beginning and try to preserve old functionality while new structure would allow us to easily implement new features.

1.3 Problem

We had to analyse root cause of problems arose during development. Based on analysis we had to propose abstract architecture model of new application also usable for planned features using modular programming over many types of languages. We also had to preserve current functionality and try to reuse as much as possible from existing code. Finally we had to reimplement original code into new architecture.

Chapter 2

Weblogs domain

Weblogs are a web applications designed to express opinions or observations of a person, company or community in form of articles or comments. There exists many types of blogs: tumblogs, videoblogs, microblogs, team/project blogs. We can also classify blogs according to topic: blogs about cooking, promoting new videos, personal blogs and so on.

Typical blog user enter on his personal blog, write an article about particular subject to share with community. Usually everyone can react to article by comment or by its own articles on written one to give feedback to author. Blogs also contains features like tagging, searching, voting, charts of top 10 users, archive list, most commented articles and most visited articles for some period.

For our goal we need to analyze structure of blog and separate it into logical parts, describe its actions and interactions.

2.1 Objects

On weblogs we have following objects we need to store informations for:

- **Articles** - contains main text, title, creation time. It is associated with author and its blog, votes, comments, visits, tags, categories and much more metadata
- **Blog** - associated with single user (however some blogs allow collaborative editing and many users may contribute on single blog)
- **Comments** - every article may have comments which are stored in tree structure. So comments are associated with article or another comment. Every comment has

author, time and may have title, rating, ...

- **Users** - users of portal with many personal informations associated
- **Categories** - each category contain articles with similar topic
- **Portlets** - part of page where are showed informations related to current blog or article. Each shows different kind of informations
- **Tags** - keywords associated with article. Tag describe an item and allows it to be found again by browsing or searching
- **Rating** - express opinion of visitors and other users on rated item. It is represented by single number
- **Layout** - deals with arrangement of elements like header, footer and portlets on blog

2.2 Actions

Each user is assigned to role. Each role has a set of available actions. These are common actions:

- **Reading** - reading article or comment
- **Writing, editing, deleting** - writing article and comments, editing and deleting own articles
- **Commenting** - leave a comment responding to article or another comment
- **File uploading** - upload photos, videos or any other content used in own article
- **Filtering** - filter articles according to its attribute (language, text match, tag, category)
- **Voting** - each user may rate articles and potentially comments
- **Recommendations** - we are using recommendation mark for articles. This mark can be attached only by administrators

Each portlet usually defines its own set of actions available to users. Many blog portals support almost all from actions above and many portals support even more actions. Actions are main logic of portal and controls flow of application.

2.3 blog.matfyz.sk

First version of blog portal `blog.matfyz.sk` had been developed by Anton Kohutovič. Portal is aimed at community of faculty of Mathematics, Physics and Informatics. This community is made of students, teachers, graduate, future students and other employees. We are using this portal with three main goals:

1. Communication channel in our community for sharing informations mostly related with life at university

2. In process of learning. We are using portal many courses (Modern approaches to web design, DVUi, ...) to learn students work with new technologies and understand many principles.

3. Research platform for graduating students. We are testing new or modified algorithms (ex. EigenRumour), testing new technologies (ex. XML databases) and principles of usability, accessibility with focus on SEO. SEO - Search engine optimization is a process of web application output optimization using meta tags, proper use of headings and other properties which search engines takes into account during indexing. SEO optimized web is placed higher in search engines results. Higher place usually leads to more visitors.

We are using own implementation of blog portal instead of existing one because:

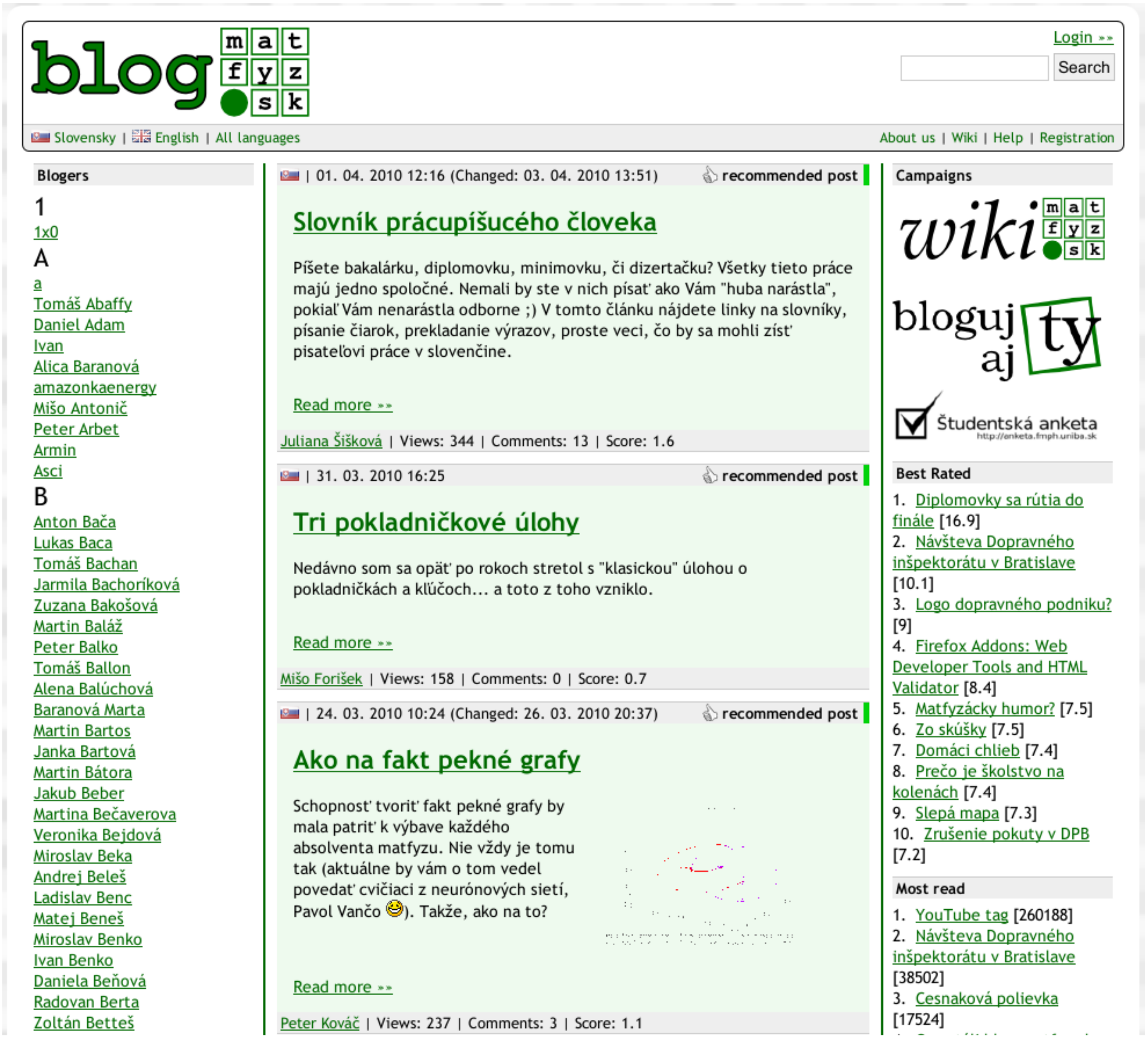
- There is not known blog system which is using XML database, XML and XSLT processing

- We already have many algorithms developed which uses current infrastructure and data formats

- There is not known system which is ready enough for our future plans (collaboration infrastructure)

- In case of existing one we would need to keep its core up to date to avoid potential security problem and track changes after update to maintain full functionality of our portlets
- Implement backward compatibility for links and user entered data
- Transform current data to another format

In addition we have also `wiki.matfyz.sk` with common informations for community where everyone can read and write.


blog matfyz.sk
Login »»
Search
Slovensky | English | All languages
About us | Wiki | Help | Registration

Blogers

1
1x0
A
a
Tomáš Abaffy
Daniel Adam
Ivan
Alica Baranová
amazonkaenergy
Mišo Antonič
Peter Arbet
Armin
Asci
B
Anton Bača
Lukas Baca
Tomáš Bachan
Jarmila Bachoríková
Zuzana Bakošová
Martin Baláž
Peter Balko
Tomáš Ballon
Alena Balúchová
Baranová Marta
Martin Bartos
Janka Bartová
Martin Bátora
Jakub Beber
Martina Bečaverova
Veronika Bejdová
Miroslav Beka
Andrej Beleš
Ladislav Benc
Matej Beneš
Miroslav Benko
Ivan Benko
Daniela Beňová
Radovan Berta
Zoltán Betteš

| 01. 04. 2010 12:16 (Changed: 03. 04. 2010 13:51) recommended post

Slovník prácupišucého človeka

Píšete bakalárku, diplomovku, minimovku, či dizertačku? Všetky tieto práce majú jedno spoločné. Nemali by ste v nich písať ako Vám "huba narástla", pokiaľ Vám nenarástla odborne ;) V tomto článku nájdete linky na slovníky, písanie čiarok, prekladanie výrazov, proste veci, čo by sa mohli zísť pisateľovi práce v slovenčine.

Read more »»

Juliana Šišková | Views: 344 | Comments: 13 | Score: 1.6

| 31. 03. 2010 16:25 recommended post

Tri pokladničkové úlohy

Nedávno som sa opäť po rokoch stretol s "klasickou" úlohou o pokladničkách a kľúčoch... a toto z toho vzniklo.

Read more »»

Mišo Forišek | Views: 158 | Comments: 0 | Score: 0.7

| 24. 03. 2010 10:24 (Changed: 26. 03. 2010 20:37) recommended post

Ako na fakt pekné grafy

Schopnosť tvoriť fakt pekné grafy by mala patriť k výbave každého absolventa matfyzu. Nie vždy je tomu tak (aktuálne by vám o tom vedel povedať cvičiaci z neurónových sietí, Pavol Vančo 😃). Takže, ako na to?

Read more »»

Peter Kováč | Views: 237 | Comments: 3 | Score: 1.1

Campaigns

Best Rated

1. Diplomovky sa rútia do finále [16.9]
2. Návšteva Dopravného inšpektorátu v Bratislave [10.1]
3. Logo dopravného podniku? [9]
4. Firefox Addons: Web Developer Tools and HTML Validator [8.4]
5. Matfyzácky humor? [7.5]
6. Zo skúšky [7.5]
7. Domáci chlieb [7.4]
8. Prečo je školstvo na kolenách [7.4]
9. Slepá mapa [7.3]
10. Zrušenie pokuty v DPB [7.2]

Most read

1. YouTube tag [260188]
2. Návšteva Dopravného inšpektorátu v Bratislave [38502]
3. Cesnaková polievka [17524]

Figure 2.1: Title page of `blog.matfyz.sk`

Chapter 3

Modular Programming and Principles

Modular programming is a programming technique which splits application code into several logical parts. Each part is called module. Modules represents principle of separation of concerns. Each module usually contain one or more classes which implements methods defined by known interface. Each module should have no dependencies or as few as possible. Main application mostly consists of decision tree and logic which calls modules methods.

Modularization have following advantages:

- easier maintainability
- better code clarity
- allow split project into many smaller ones
- modules are reusable in other projects
- programmers may focus on specific sub-tasks without knowledge of other parts or whole system

Definition 3.0.1 Module *implements basic functions for manipulation with abstract object implementing predefined interface.*

example: User (info about specified user), Sedna (query interface for database), Session (data related to active session), XSLT (basic operations with XSL templates), ...

Definition 3.0.2 Library *implements set of algorithms related to specific task.*

example: RankManager (may provide several rating algorithms of users or articles), Tag (manipulation with ontology in many different ways)

During modularization we will try to follow several principles.

Definition 3.0.3 Single responsibility principle *states that every object should have a single responsibility, and that responsibility should be entirely encapsulated by the class.*

Robert C. Martin [Mar03]

This principle tells that reason to change class should be only one. Let's have a module that compiles input data and print a report. This module have two reasons for its change. First, content of report may change. Second, format of report may change.

Definition 3.0.4 Separation of concerns *is a process of separating program code into distinct features that overlap in functionality as little as possible.*

Application should have separated processes of working with databases (data access), output results (presentation), program flow decisions (logic) into layers. Main goal is to establish a well organized system where each part fulfills a meaningful and intuitive role while maximizing its ability to adapt to change. This is achieved by logical or physical constraint which delineates a given set of responsibilities. There exists two main approaches.

Horizontal Separation

Main application is separated into three main layers. Presentation, business and resource access layers. Presentation layer contain components and all program all logic directly responsible for user interface. Business layer contain all logic responsible for application domain processes. Resource access layer provide abstract access to database, files on disk and other resources.

Vertical Separation

Vertical separation means that application is separated to modules. Each module relate to the same feature and implements presentation, logic and data resource part. It usually implements interface for each of these three main tasks. This method is useful for delegation between teams.

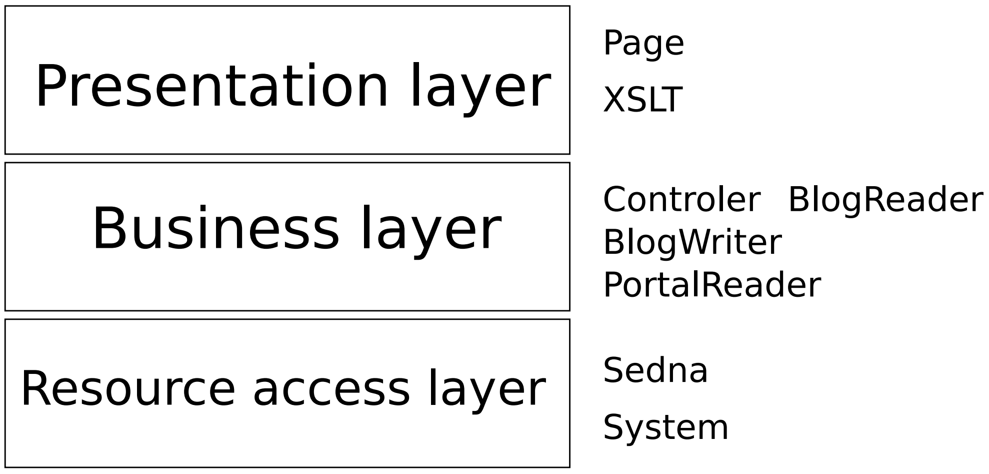


Figure 3.1: Horizontal separation

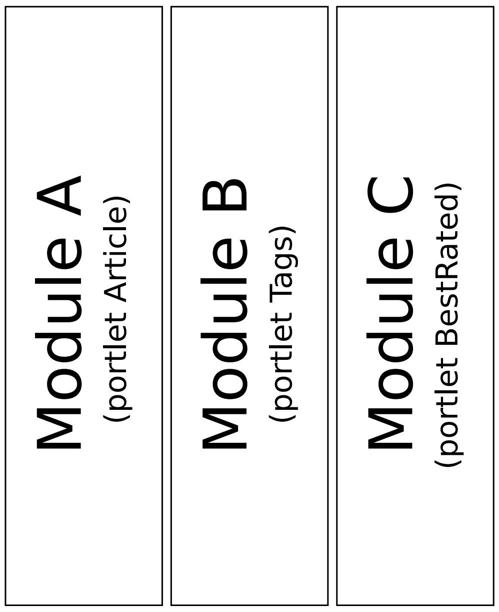


Figure 3.2: Vertical separation

However vertical separation groups a set of concerns based on their relevance in application, we can still use other separation methods. We can still separate *module A* on Figure 3.2 into three layers like on Figure 3.1.

During separation process we have to try to:

- eliminate the duplication of functionality

- restrict the scope of work to a maintainable size
- restrict the scope of work to the description of the containing boundary
- restrict the scope of work to the inherent behavior of the containing boundary
- minimize external dependencies
- maximize the potential for reuse

Definition 3.0.5 KISS *(keep it small and simple) states that simplicity should be a key goal in design, and that unnecessary complexity should be avoided.*

We used these principles and techniques during proposition of new application model.

For modularization of source code we need (optionally) language support for several paradigms which help us separate code into modules:

- source code include from multiple files
- procedures, functions or templates
- classes
- source code generation at run-time

3.1 Modular Programming in Weblogs Domain

It was separated into three layers - presentation, business and resource access. Business layer was separated into modules that processed users input and had written data into database. One functionality was placed in many classes. Each member of our team has a responsibility for some features (see Chapter 9). Thus we needed to separate it into modules where each module has responsibility for a single feature.

Portlet

Current blog systems are separated into libraries implementing basic functionality like working with database, abstraction of user management and many more. Another types of modules are portlets. Portlets are mainly used by presentation layer and provides output for part of page. Portlets often works as a plugins into many blog systems. Most common blog system is *WordPress*.

> Portlets are pluggable user interface software components that are managed and displayed in a web portal. Portlets produce fragments of markup code that are aggregated into a portal page. Typically, following the desktop metaphor, a portal page is displayed as a collection of non-overlapping portlet windows, where each portlet window displays a portlet. Hence a portlet (or collection of portlets) resembles a web-based application that is hosted in a portal.
>
> *Wikipedia* [Wik10]

Definition 3.1.1 Portlet *contain code logic for several use cases and provides interface for basic operations required by core (mainly for output generation).*

example: Portlet article handles logic: what to do if required action is adding new article, replace existing, vote for existing.

Libraries are used by portlets. Example of portlets is on Figure 2.1. There are portlets *Blogeri*, *Najlepšie hodnotené*, *Kampane* and list of articles. There are two logical states of portlet. One is side placed portlet and second is main content placed portlet.

Hooks

Hooking is set of techniques for altering or extending program behaviour using external code. Many web application frameworks contains hooks as a method used by plugins for extending application functionality. Such applications are *MyBB*, *WordPress*, *pmWiki*. Hook is an method for extending application functionality using callBack method.

If we would like to use hooks and extend functionality of remote application it must also support hooks. Such application contain call for particular hook. If we want to extend application functionality we have to register our function for such hook. Once the target application reach point of call for hook, our function will be called.

For example, let's have an application with authentication interface and several points in source code where the call for hooks is invoked. We would like to extend such application with notification of administrator for each failed login. We need to register our notification function for particular hook which is invoked by authentication system in case of authentication failure. If we do it, once authentication fail, our function will be called by the system, because we have made its registration.

Hooks would allow us to extend application functionality transparently in modular way.

Chapter 4

Used technologies

In this chapter we present most important technologies we are using on portal, with respect to modularization we were doing. Many of these technologies support some level of key features we needed. Since we need to split code into several modules, we need support for including (or merging source files at some level), functions or classes support, respectively some kind of paradigm which allow us to separate code for users registration from code creating output. Since we are using imperative (PHP), declarative (XSL), functional (XQuery) and query (XPath) languages, each with different level of support of key features, it is not easy to fully modularize code into semantically separate modules.

4.1 PHP

PHP is imperative object-oriented scripting language, designed for development of web applications. We are using PHP for our main web application and most of logic is written in PHP. On the side of webserver, when request is received, web server runs PHP interpreter with parameters gathered from request on requested script file. PHP interpret script file and prints output which is sent back to user by web server. Request contain all form data including address of requested page.

PHP has a support for almost every feature that can be used for good code separation. We can use following features for separation process:

- classes
- interfaces

- inheritance
- sources across multiple files (support for includes)
- passing objects, classes and functions as parameter
- namespaces

Namespaces were added in the latest version of PHP (5.3) and support is still in experimental state. PHP has a great support for different platforms, include support for many databases (including Sedna), many data formats (including XML) and external processors (like XSLT). It is free, open-source, widely adopted on the web and used by many big companies like Facebook for many years.

We are using PHP also for data and code generation of other languages we are using on our portal. We could use PHP for processing input, storing informations and generate output, but it would be ineffective and complicated. In case of storing informations we use database which could handle fast large amount of data. Output generation could by done using existing templates systems but it still require checking for valid output date to prevent XSS attacks. XSS attack consists in injection of web browser executable to database. Once this code is sent to output without proper checks, it is executed in visitors browser.

4.2 XML

XML is a language derived from SGML. XML is used for description, sharing and exchanging data between different systems. Main advantages of XML are simplicity and flexibility. XML is text format with support of unicode. XML itself does not specify type of content, it only specify syntax for data and optionally description of data structure.

Physical structure of XML consists of XML header listed in Listing 4.1, comments, elements, attributes and their values, entities and content inside elements. XML has exactly one root element. Element may contain recursively more elements. Each element may contain attributes and for each attribute there must be value. Element itself is pair of opening and closing tag.

Listing 4.1: XML declaration

```
<?xml version="1.0" encoding="utf-8" ?>
```

Listing 4.2: XML example

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE post SYSTEM "post.dtd">
<post lang="en" private="yes">
  <title>Hello world!</title>
  <subtitle/>
  <content><p>Welcome to my blog...</p></content>
  <tags>
    <tag>debug</tag>
  </tags>
</post>
```

In case of Listing 4.2 there is also (optional) pointer to DTD document. DTD describe constraints for structure and data type for each part of document. For example, there may be constraint, that inside `post` element there must be elements `title`, `subtitle`, `content`, `tags` and optionally element `comments`. There exists more formats for XML structure description, like XML Schema, which is XML file describing structure of another XML file. Therefore even complex XML document can be checked against DTD for validity.

There exists two levels of XML documents:

- **Well-formed** - XML document that satisfy all syntax requirements according to [W3C06].
- **Valid** - well-formed XML with DTD (or another structure description format) where all elements are defined.

Advantages of XML:

- Easily readable and parsable
- Hierarchical structure makes it appropriate for many types of documents and data
- Support for namespaces
- Easily extendible without problems with backward compatibility
- Built-in support for validity checking

- Support for Unicode
- Widely used and deployed

Disadvantages of XML:

- Overhead, as each tag has to be paired
- Slower data processing compared to other formats
- No support for built-in includes of multiple documents (support has to be written in processor itself)

We are using XML on our portal as a main data format. It allow us to easily check input documents for validity and in conjunction with HTML avoid several security problems like XSS attacks.

4.3 XSL

The Extensible Stylesheet Language Family (XSL) is declarative language used for transformation of one XML document into another XML document. XSLT stylesheet has form of XML document. It describe how to translate input XML. XSLT has a form of XML template with instructions for XSLT processor to let him known desired form of output XML.

XSL has these basic programming constructions:

- iterations through nodes
- conditions
- one-time definable variables (local and global)
- built-in XPath processor that allow further transformations over processing data
- ”functions”
- multiple documents import

Basic XSL stylesheets can be extended by more constructs, but this require support of this constructs by XSLT processor. For our purposes the most important features are "functions" (in XSLT we mean templates) and import of external documents. However, second feature could be simulated in higher language, we are using it. In case of our portal, we are using XSL for transformation of data from database to HTML output. We don't need to handle input for injection attacks since XSLT processor handle this correctly itself.

Today XSL proccesors are slow so we should avoid use them for complex computations and processing of large documents. Moreover since XSL is XML it has its disadvantages. In this case XSL has a code overhead and each instruction has a long notation. For the last, XSL has less features usable for modularization than PHP and thus it is more difficult to maintain.

Listing 4.3: Example of XSLT

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:import href="functions.xsl"/>
<xsl:param name="userID"/>
<xsl:template name="main" match="/">
  <user id="$userID">
    <name>
      <xsl:value-of select="person[@ID=$userID]"/>
    </name>
  </user>
</xsl:template>
</xsl:stylesheet>
```

4.3.1 XSLT Processing

Processing of XML using XSLT in PHP consists of these steps:

1. in PHP we load XSLT from file or from String to DOM object (used for storing generic XML documents)

2. in PHP we create object of type XSLTProcessor and let it load XSLT document from DOM object, created in previous step
3. (optionally) we set input parameter for XSLT processor which can be accessed from loaded template
4. call processing function with DOM object, where is input XML. Result is processed XML

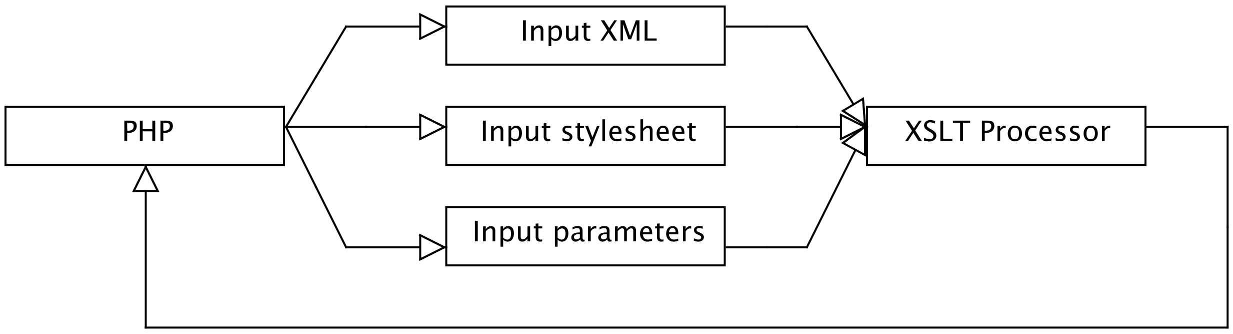


Figure 4.1: XSLT processing

In first step we may load XSLT from file or we can create dynamically one. In third, if we will create dynamic stylesheet or input XML, we can encode input parameter into it. So there exists possibility of writing fully optimized XSLT stylesheet, generated in runtime. This is how works Gallium3D with combination of LLVM. LLVM is low-level virtual machine. Gallium3D is a new framework for graphic drivers. It takes source code, compile it to its own bytecode and then this bytecode is transformed to specific instruction set of target hardware. It can take code for graphics shaders and recompile it to binary form with optimizations for specific graphic card. In our case, it could be optimization for specific XSLT processor.

4.4 XML Database

We are using XML database Sedna. Basically it is storage for XML documents that allow manipulating with these documents using XQuery language. Basic unit of information in XML databases is XML document. Documents are organized in collections. For each collection we may define specific schema that describe structure of XML documents inside one collection (like DTD).

There are some differences against relational databases:

- In relational databases we store data in many tables and these data are almost somehow logically separated. In XML database we can store whole XML tree, which is in our case input and output. Thus we may not need to transform data from one form to another as they are in the same format
- In relational DB basic unit of information is row in a table, in XML DB basic unit is XML document
- In XML DB equivalent to tables are collections

In PHP we work with XML database similarly to relational databases. We connect, create and send query and then retrieve result. The main difference is that the result we get single XML document and we do not need to iterate through many lines of result like in case of relational database.

4.4.1 XQuery

XQuery is functional language used in XML databases. XQuery allow us to retrieve, insert, replace or delete some nodes from documents. It also allow us to make some processing over retrieved or inserted data. XQuery has support for functions. This feature can be used for modularization of complicated queries. Here are some examples of XQueries:

Listing 4.4: Retrieve XQuery

```
collection('weblog')/user[@ID='u74']/info
```

Listing 4.5: Insert XQuery

```
update insert
  <post ID="p16084" accessCount="0"
     lang="sk" status="draft" private="no">
    <title>example</title>
    <content> example </content>
    <tags><tag>example</tag></tags>
  </post>
into collection("weblog")/user[@ID="u190"]/blog
```

Listing 4.6: Replace XQuery

```
update replace $i in
  collection("weblog")/user[@ID="u74"]/blog/post
    [@ID="p16016"]/@accessCount
with attribute accessCount {$i+1}
```

Listing 4.7: Delete XQuery

```
update delete
  collection("weblog")/user[@ID="u190"]/blog/post[@ID="p16084"]
```

XPath

As a part of XQuery is XPath language which is query language, used to specify which XML node from each document in collection we want retrieve, update or delete. XPath is also used in XSLT. XPath defines what parts of XML document will be used for processing. XPath consist of expression made by slashes, element names, attribute names and its values, XPath functions and angle brackets where are constraints over elements. In examples above it is a part that defines nodes of documents for which the operation have to be performed. In case of Listing 4.4 whole query is just an XPath expression. More complicated example can be found in Listing 6.8. XPath does not provide support for modularization. The only possibility is dynamic generation of XPath by another language.

Chapter 5

Code Analysis of blog.matfyz.sk

In this chapter we will present results of original code analyses. We will reveal and describe most critical problems with reasons why we consider our findings to be the problems. Analysis start with model description, summary of problems we found and exact description of each problem.

5.1 Original Model

Original data model and some classes description can be found in [Koh08]. We consider XML database data model to be good enough and during development of new features we were able extend it without problems with regards to semantic of its structure. Main application used horizontal separation model of modules. Modules `Page`, `XSLT` represents presentation layer. Modules `Controler`, `BlogReader`, `BlogWriter`, `PortalReader`, `User` represents business logic layer. Modules `Sedna`, `System`, `Session` represents resource access layer.

On Figure 5.1 we can see main modules of original web application providing main input processing logic. On Figure 5.2 are modules responsible for generating output of each section of portal.

- **Session** - keeps session information about current visitor. This module was used for retrieving informations about current session, like currently logged user or selected language
- **User** - provide logic for reading and writing data, like password, email, name, about

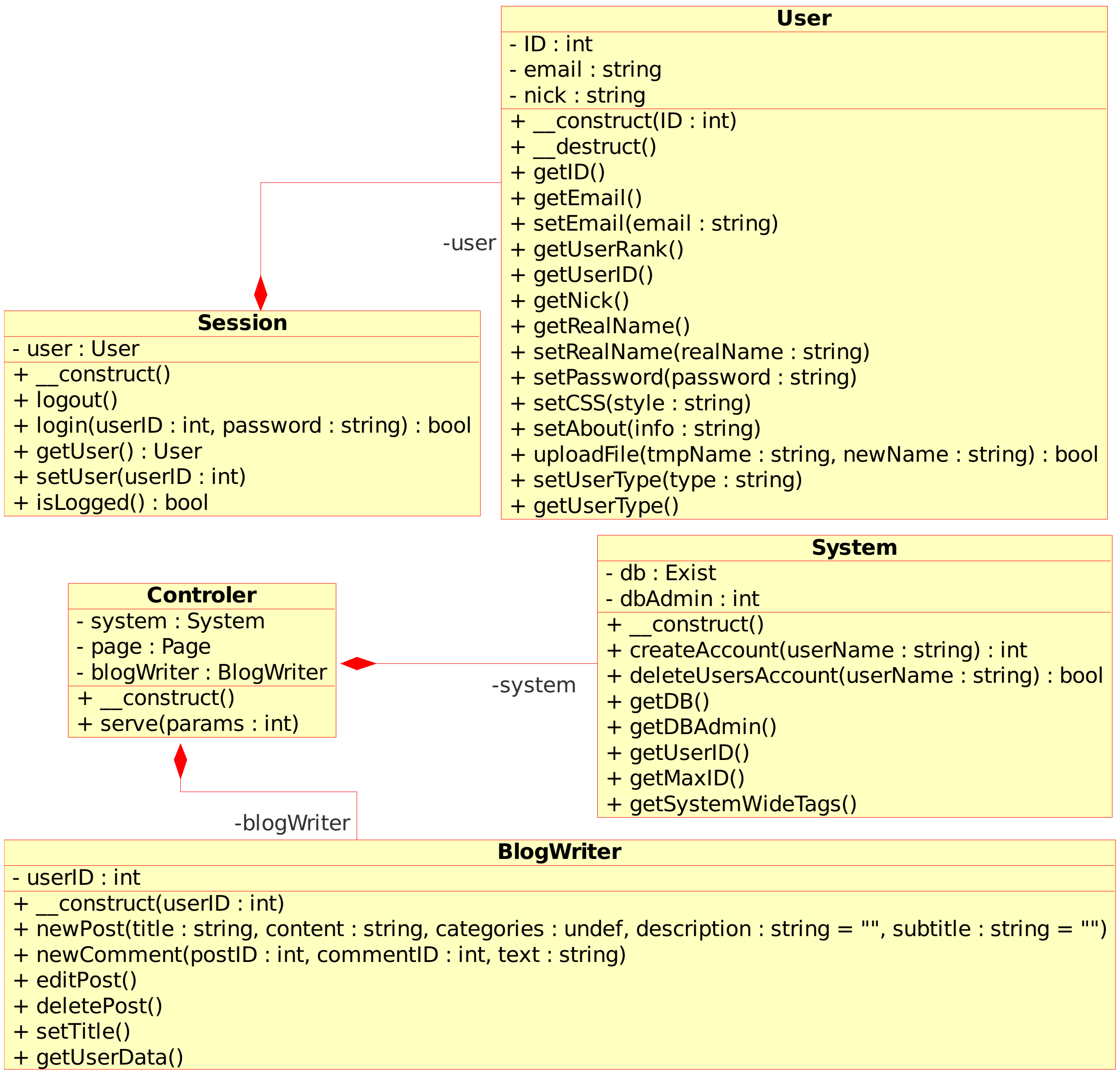


Figure 5.1: Main modules containing almost all logic of portal

particular user to database.

- **System** - provide access to database object (of NXD[1] type) and other actions of various type and semantic
- **BlogWriter** - contains logic for writing data which are related to main blog of each

[1]Native XML Database

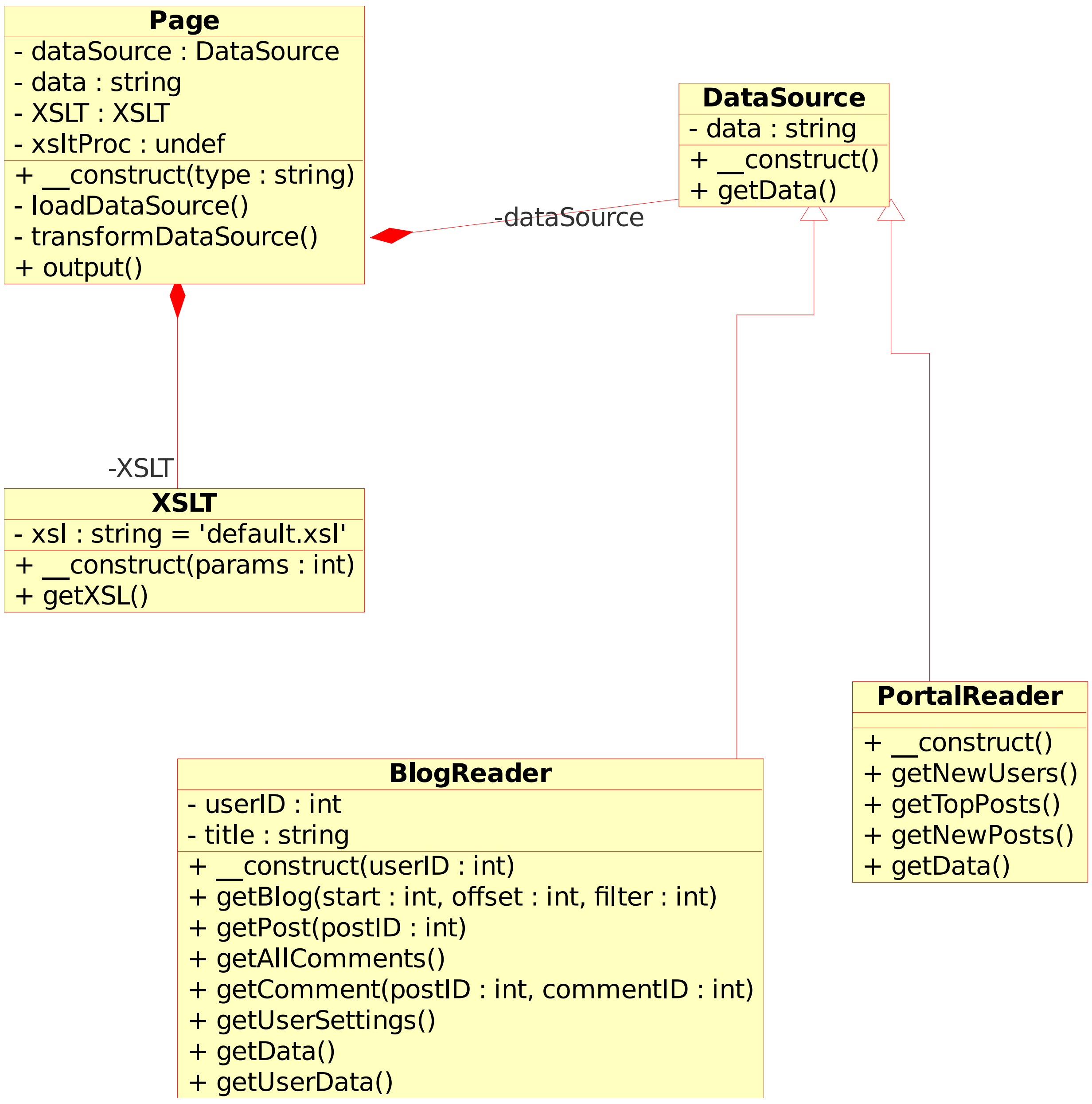


Figure 5.2: Modules providing logic for presentation layer

user. This modules was used for writing data to database related to particular blog. It writes new articles, comments, blog title.

- **Controler** - main class controlling code flow of main application, checks security and permissions

- **Page** - this module was used for generation of application output. It contained main logic of presentation layer. It took one of `BlogReader` or `PortalReader` object type, asked for XML and according to page type choose XSL template, transformed XML, parsed using DokuWiki and sent to output.
- **XSLT** - helper module for simple working with XSL templates. It was used for XSL transformations of XML documents by module `Page`
- **DataSource** - interface defining methods for output modules implemented by `BlogReader` and `PortalReader`
- **BlogReader** - contained everything logic needed for generating of user blog. It was used for generation of XML with informations about particular blog. Informations like articles, comments, blog title were retrieved from database
- **PortalReader** - contained everything logic needed for generating of main portal page. This module generated XML with informations about portal, informations for each portlet placed on main page. Such informations were articles, portal users, top 10 articles

Original application was separated into three layers: presentation, logic and resource access. Inside business layer, there was several modules that contained main logic and can be classified as modules used for reading (`BlogReader`, `PortalReader`) and modules for writing (`BlogWriter`). Moreover we can classify them as abstraction over page type (main page, blog).

Modules such as `Session`, `User`, `Page`, `XSLT` has been used as an abstraction over data objects and we can call them *libraries*. `Controler` was used for decision of what to do (write data or generate page) according to input parameters. Moreover it contained logic for users registration, login, permissions checking, saving user settings and management for file upload.

On Figure 5.3 we can see code flow and interaction between modules and libraries. Flow can be described by following steps:

1. user send request
2. `controler` create necessary objects and checks permissions using session and user library

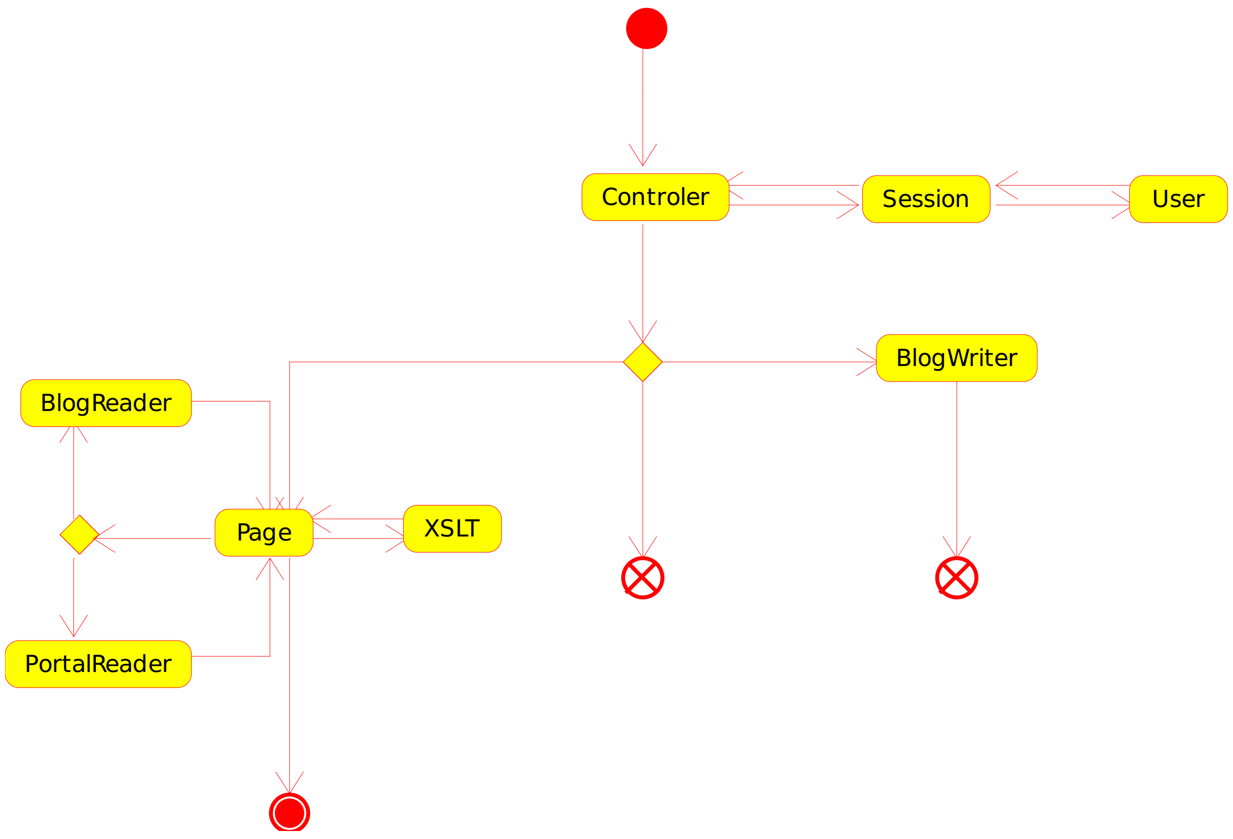


Figure 5.3: Application code flow

3. upon input action, it demands where to redirect flow
4. in case of login, registration, changing user settings and file upload
   (a) ends by redirecting user to blog of logged or new user
5. in case of doing change related to user blog (adding comment, writing new article, ...)
   (a) create new instance of BlogWriter and pass control to it
   (b) BlogWriter checks input and apply required changes
   (c) finally it redirects user to page that belongs to users blog depending on input action
6. in other cases control is passed to `Page` module to generate output for user

(a) upon action `Page` choose one of classes which support `DataSource` interface (`BlogReader` or `PortalReader`)

(b) call `getData()` method from that class

(c) called class generate all XML data needed for output

(d) `Page` use `XSLT` for transformation of previously generated data to HTML format. In `XSLT` is all logic of choosing the right stylesheet.

(e) output of `XSLT` is transformed using DokuWiki parser as we support multiple formats for articles

(f) output is sent to user

5.2 Revealed Problems

In this section we mention most critical problems we have found during analyses. Most of them lead to regressions, code duplication, slow development, ambiguity of functionality and making so called "ugly hacks".

One of hacks was applying *DokuWiki* parser over output from XSLT, where all parts that had to be transformed had been marked with special tags. Most of regressions originated from ambiguity of function of each class, code duplication and missing documentation.

5.2.1 Functional Oriented Classes

In Section 5.1 we mentioned that almost all logic over data is separated in three classes (`BlogReader`, `PortalReader`, `BlogWriter`). If we would like to add some features like private articles, categories, collaborative editing, their functionality should be separated into this three modules depends on what action we want to do (read or write and where). Thus with any new feature, the number of methods in this classes would grow up.

In our team, each developer has responsibility for particular function such as tags, private posts, design. Thus original decomposition lead to state where each developer had to edit multiple files, multiple shared functions. Basically everyone edited everything.

Finally in almost every class, main logic is inside constructor. In original code, user authentication was fully implemented inside class constructor. According to B. Meyer

[Mey97] class constructor should only initialize local variables and no exception should be thrown.

5.2.2 Filtering Articles

Articles in our database has many attributes (language, private, author, title, date ...). During data retrieving we often need to filter articles by various criteria. For this purposes there was a function `getPost`. This function had for each attribute we wanted to filter one parameter. Another two parameters were used for paging, to tell function that we want ten articles and skip first 20 articles.

As we had added tags, private articles, search and so on, we always needed to change the function interface. This function was called from several places and different modules. Sometimes we didn't want to filter for tags or private articles. As a result, during development, there were two forks of this function with only small changes, mainly because of different sorting algorithm inside XQuery that were in this function. As we describe in Section 5.2.3 all this changes had to be applied also inside templates.

As we added more and more filters and properties, we have needed to rewrite all three functions. Adding more and more properties in future would lead to many regressions as it was in the past. Moreover, using different sorting algorithms would leads to more forks. Finally, it is not clear which function we want to call in many cases.

5.2.3 XSLT

Functionality in Templates

XSLT has to be used for transformation of XML to another XML we are using it only for transformation of input data from database to HTML. However, XSLT has a power of Turing machine [RO06]. According to single responsibility principle almost every functionality has to be in PHP code and XSLT should be used only as a template engine for HTML output.

In original implementation of portal there was many computing and filtering functionality in XSLT instead of PHP which has to be used for logic. This lead to duplicity of semantically similar code between many files and languages.

Following logic was in templates:

- additional filtering of articles based on user input parameters
- computing of tag clouds
- many case-of statements demanding upon filter type
- texts translations - we have support for multiple languages on our portal
- opening external XML files used for caching. More in Section 5.2.4
- hardcoded file paths - to CSS, images, translation files
- link modifiers - used as a part of SEO optimizations. For example instead of `http://tbc.blog.matfyz.sk/?postID=p14617` we have `http://tbc.blog.matfyz.sk/p14617-boli-ste-uz-tam`

Moreover, in future we want to add some level of design and functional customizations for user blogs. It would arise complexity of templates.

Code Sharing

There were almost no modularization over templates. As a result, many templates had large parts of code same. There were no single template for header or footer that are same on each page. As a result, once we needed to change a footer, it had to be changed in many files instead of one.

5.2.4 Caching

We are using several algorithms that needs to run offline because makes complex calculations which takes long time. For example: recomputing users karma, article score takes several hours. Computing answers for tag-cloud and ontology over tags takes approximately one minute. Since web application should response as soon as possible to users requests, we can't run such calculations for each response. This is unacceptable long time. Thus we are using cache for storing intermediate results of calculations. Each request that needs results of such calculations provide data from cache.

Original code has two main problems.

Caching Codepath in PHP

Creating XML files with results for cache was done through special codepaths created for this purpose. There were no class that managed caching results. Decision whether to use cached results or compute them was done in constructor of main class instead of function that calculated data itself. Caching has to be transparent from definition. Another problem was, that caching and recalculations were done periodically, using cron task[2], even if source data was not modified since last computation. As a result, caching was not transparent and ineffectively slows database responses.

Use of Cached Results

Because caching was not transparent, there were calls from XSLT templates to open external XML documents with cached results. These results had to be filtered and modified according to user input parameters, because in the time of caching, these parameters were not known. For example: articles had to be filtered by tags, searched words, selected language, and many more. Thus we had duplicate code logic, doing the same in two different languages.

5.2.5 Connection With Courses

Our portal is also used for learning purposes, where each student of course have to create its own XSL template. There was support for this feature. It was implemented across each library and interfere with many methods. In many cases it was not clear which code is related to logic of blog portal and which manage logic of course.

For example: blog portal support different user levels such as *visitor*, *regular user* or *admin*. Code related to courses added new types: *student*, *teacher*, *coursevisitor*. According to this new types portal enabled new features and disabled some common. Students can write posts, but they has special editor which use its own syntax.

If we would like to add a support for new course with its own set of features to original code, we would need to add it to each class, change many existing code to reflect our changes. Code will become too complex and removing it would be hard.

Adding support for courses needs to be more transparent and clearly separated from portal code.

[2]Cron is a service that periodically runs defined tasks

5.2.6 Backward Compatibility

We had problems with students templates, which were created few years ago. This problems are results of changes in input format of XML that was extended with global tags database. We found, that many old templates used `//tags` as XPath expression for iterations over tags in articles.

Backward compatibility of links to articles and some parts of portal is important for SEO optimizations. There was several changes of links format. This leads to breakage of links from other portals. This needs to be defined for future to avoid regressions in templates.

5.2.7 Database Performance

During development, we had several performance issues with our Portal. Sometimes request takes over one minute and many times it was over one second. This is unacceptable for web applications that interact with users. Another problem was that application reached limit of connection number to database at the same time however there were not so many user requests.

We added basic logging. It simply gets system time at request start, get system time just before output is sent and add difference to the end of output.

This first benchmark showed us, that the problem is not in line speed to user, but application itself. As next step, we added time measurement for each XQuery request (Figure 5.4).

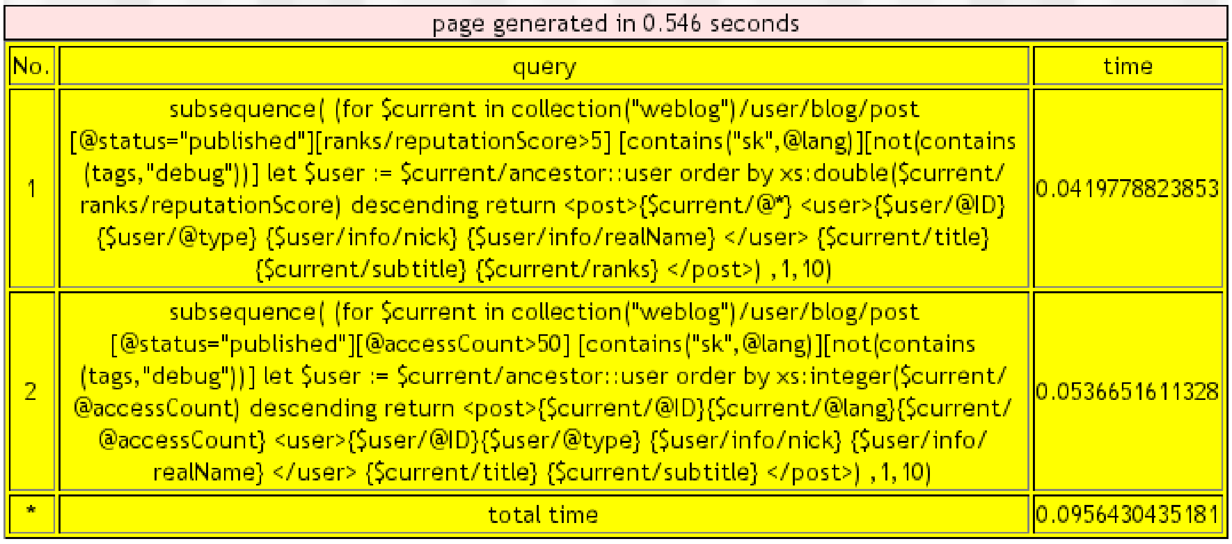
Difference between *total time* and *page generated* time is time used by PHP and XSLT processor.

Our analysis showed that most of the time, our application was waiting on XML database. This vary from request to request. Most of time had take write requests such as in Listing 5.1.

Listing 5.1: XQuery that increase number of accesses to article

```
update replace $i in
collection("weblog")/user[@ID="u77"]/blog/post[@ID="p123"]/@accessCount
with attribute accessCount {$i+1}
```

We've tried to split this request into two. One, that get *accessCount* value and second that write that value, increased by 1. This didn't help so we had to analyze Sednas



page generated in 0.546 seconds		
No.	query	time
1	subsequence((for $current in collection("weblog")/user/blog/post [@status="published"][ranks/reputationScore>5] [contains("sk",@lang)][not(contains (tags,"debug"))] let $user := $current/ancestor::user order by xs:double($current/ ranks/reputationScore) descending return <post>{$current/@*} <user>{$user/@ID} {$user/@type} {$user/info/nick} {$user/info/realName} </user> {$current/title} {$current/subtitle} {$current/ranks} </post>) ,1, 10)	0.0419778823853
2	subsequence((for $current in collection("weblog")/user/blog/post [@status="published"][@accessCount>50] [contains("sk",@lang)][not(contains (tags,"debug"))] let $user := $current/ancestor::user order by xs:integer($current/ @accessCount) descending return <post>{$current/@ID}{$current/@lang}{$current/ @accessCount} <user>{$user/@ID}{$user/@type} {$user/info/nick} {$user/info/ realName} </user> {$current/title} {$current/subtitle} </post>) ,1, 10)	0.0536651611328
*	total time	0.0956430435181

Figure 5.4: Table with debug time measurement

- **Page generated** time - total time from request to final output, including requests on DB
- **Total time** - sum of time that takes each query to database

behaviour itself. We used program `top`. This program can show overall *CPU usage*, *CPU usage per process*, *memory usage*, *I/O wait*, *CPU idle* and many more. We monitored Sedna during requests. After while, problem appeared to be I/O wait, while CPU was in almost idle state.

According to [fSPR10] Sedna is using temporary file `database.setmp` for intermediate results and for caching. This was causing heavy I/O load and was root cause of slow responses.

After we improved performance (description in Section 6.4), we had still sometimes page response over 1 second. Further analyses showed, that there must be problem in Sednas locking mechanism, because we had made several concurrent requests on database and there was no O/I wait, but CPU was almost at 60% idle. This was reported. At this time, we are waiting for confirmation from Sednas team.

5.2.8 Missing documentation

During development we had only documentation of class relations and data model of database. There was missing:

- description of format of input XML for templates created by students
- list of input parameters and their values from user requests
- description of code flow
- methods and their description implemented in each class (especially `BlogReader`, `PortalReader` and `BlogWriter`)

Chapter 6

Proposed Solution

In previous chapter we analysed problems of our portal. In this chapter we will describe proposed solution which would solve these problems using code modularization. As not every language, we are using, provide same or enough level of support for modularization, we need to consider it in our propositions and solve it using different techniques. During analysis we found also problems with performance and time generation of pages. We identified that source of this problems is XML DB and implementation of its several parts. We propose preliminary solution, however complete solution is to fix Sedna itself. Web application code was fast enough. Finally we proposed library which allow us to extend any part of application in future using hooks.

6.1 Improved Model

We need approach how to separate existing code into many semantically different portlets, but still can easily provide at least same level of functionality and increase overview over code from developers point of view. Moreover we want to group semantically same functionality together. For example: almost everything that is related to tags should be in one portlet. In Chapter 2 we identified basic objects and actions applied on them. This will be used as a key in our propositions. Almost everything related to graphic design in another with minimum overlap. Once we had overlap, we strictly define how to solve it.

Web application basically consists of two tasks. First is to process input data from request. Second is to provide output based upon user request. Our model will use vertical separation of web application to portlets and one resource access layer common for each portlet. Each portlet will be separated horizontally to two layers. Business and presentation layer. Each layer of each portlet will be called from application core.

We make an assumption that web application consists of three main steps:

1. main initialization
2. process input data from user
3. generate output

In first step we will load and initialize each portlet responsible for one feature. In second step core let every portlet process input data and in third step each portlet have to provide data for output.

6.2 Core

Core of our application is responsible for main application work flow. The only assumption it has is that we works with XML as input data for presentation layer and XSLT as final output generator.

On Figure 6.1 is code flow of proposed application core. In our case, taking into account used technologies, core consists of following steps:

1. **Initialize libraries** - main initialization and creation instants of libraries which represents abstract object (database, user, session, ...)
2. **Create portlets** - create instants of portlets
3. **Process input data** - process input data from user (check, modify, write to database)
4. **Generate XML** - generate input for templates (XML and parameters)
5. **Apply XSL** - apply XSLT templates on generated XML and send result to output

In first step we will create libraries responsible for communication with database, manipulation with session, user data and some libraries which help us to manipulate with XML and XSL. In second step we load and initialize each portlet. In third step, every

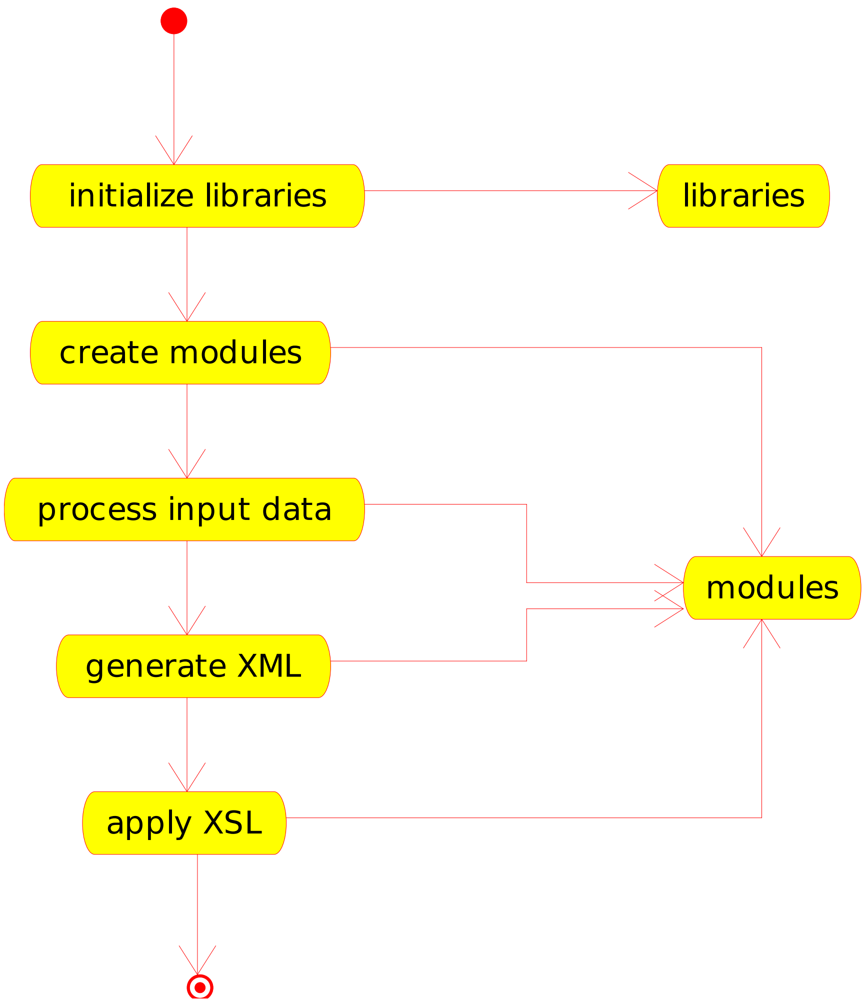


Figure 6.1: Main application code flow

portlet has to process input data from request. In fourth step core let every portlet generate XML data for requested page. In last step each portlet has to provide XSL stylesheets used for generation of final output.

Each portlet has to work only with data which are related to its feature which implements. Steps 4. and 5. are responsible for output and represents presentation layer. From above assumption, we design new core and portlets called by core.

In our model libraries are used for manipulation with data and logical objects and are available to each portlet. Portlets provide methods for processing input data, generates XML and processing instructions for templates. Portlets initialization is done in constructor in step 2.

After step 4. we get XML data from each portlet for future processing by XSL templates. There are two ways how we could manage output XML data for stylesheets.

- Save output and then use it as an input for templates from the same portlet.
- Join portlet generated XML parts together into single one, mark each chunk by portlet name, identifier or use namespaces, which produced it and process final XML

We decided to use second option. However it is less cleaner in terms of modular programming. It has following advantages:

- it allow portlets to share common informations through XML (for output, however it is not recommended)
- allow us to make modular layout much more simple. Otherwise we would need to merge processed XML (for final output) in another special step.
- save multiple calls for processing XML by XSLT and process it in a single pass
- makes application core simple

6.2.1 Main XSL Template

There are three ways how to merge templates from portlets by core:

- **static merging template** - is one file on disk importing static templates from files returned by portlets. List of files for import have to be passed in input XML. This method require static templates from portlets.
- **dynamic template creation with imports** - template is dynamically created during runtime and its code contains paths to templates returned by portlets. This method require static templates from portlets.
- **merging templates using PHP** - each template is merged into one template using PHP in similar way XML was. This method may accept both types of templates and in case of static templates uses `xsl:import`.

Using *static merging template* would require adding instruction for imports XML inside XSL template. Dynamic templates uses advantage of XSLT processor `xsl:import` feature and list of imported templates can be generated during main template generation. In Listing 6.1 is example of dynamically created template which imports static templates provided by portlets.

Listing 6.1: Example of template generated by *dynamic template creation with imports* method in core

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:import href="articles.xsl" />
  <xsl:import href="tagcloud.xsl" />
  <xsl:import href="userlist.xsl" />
</xsl:stylesheet>
```

6.3 Portlets

We are using portlets in similar way to other blog portals but our portlets have simple interface, does contain only high level logic and could implement only functional part without presentation part. Moreover every functional part of our page is a portlet. Other blog systems use portlets only as a simple side-showed functional panels which provides several simple actions or display content specific informations. We extends idea of portlets to almost everything. Every part of page responsible for some functionality. Thus portlets from Figure 2.1 will be language menu, main menu, search field, login/logout field. We use one special portlet for managing blog layout. Full description is in Section 6.3.4.

We use portlets as a tool for grouping same functionality into one place. Thus we uses almost combination of vertical and horizontal separation principle together described in Section 3. Our portlet has to:

1. process input data from users request
2. generate XML
3. provide template for processing XML returned from previous step to output

For each step, portlet has to implement method. All steps are called consecutively from core. In first step, portlet has to process all input data from users request. In step 2. portlet has to take all needed data for requested page from database and other resource access modules, process it and return. In step 3. portlet has to return XSL templates needed for proper transformation of data to output, returned from step 2.

Portlet can be used twice at a time. Once showing extending informations in main content area and second in side panel. However, our design allow one portlet to generate output for both places showed at the same time, we propose to separate both parts into two portlets for better code clarity.

We propose following interface for portlets:

- **init($params)** - handles user input and performs all necessary writes to database. It should return true if everything related to its function performed successfully otherwise false
- **getXML($params)** - returns generated XML inside string for further XSLT processing, according to requested page type, available in `$params['type']`
- **getXSLlist($params)** - return XSL templates in array
- **getXSLParams($params)** - returns associated array of parameters for XSLT processor. Every portlet that use `xsl:param` inside template have to return required parameters through this function

Variable `$params` stores all input parameters from users request. Every portlet should react only to **known** requested type. In other case for unknown types added in future it will produce undesirable output.

Introducing portlets would lead to many small modules with single responsibility and almost constant number of public functions. In future, adding new feature would lead to new portlet instead of new function inside existing module as it was in original application.

6.3.1 Generating XML

During phase of XML generation, portlet should collect all informations required for proper generation of output. Collected informations should be returned as well-formed XML. As we will need to match correct data during XSL processing phase, we need to uniquely sign generated data. We have three choices:

- using unique XML namespace
- wrap data inside element and with unique identifier for each portlet
- combination of both

In Listing 6.2 we are using combination of both methods. Identifier `article` is namespace prefix and attribute `id` has unique value for each portlet.

Listing 6.2: Example of proposed XML generated chunk

```
<article:module id="jkg645jkg">
  <!-- module data -->
</article:module>
```

If we would like to allow multiple use of same portlet at one time, we have to use unique identifiers for each module instance for proper identification of XML data during XSL transform process. This identifiers should be generated by core and assigned to each module during runtime for proper data generation and match.

For backward compatibility with students templates from courses and future support of courses there should be module which generates XML in original format.

6.3.2 Modular XQuery

XQuery is used by portlets mostly in data processing and XML generation phase for querying database. As we intend to use several sorting algorithms inside XQuery we would like to modularize it. Since XQuery doesn't support import of additional files we need to create full query each time at runtime of PHP. Some level of modularization of XQuery is possible since XQuery support functions. Example of functions used in extended XQuery is in Listing 6.8. Inside each XQuery original code had implemented several algorithms related to calculations of tags strength, sorting algorithms and filters for nodes with specified attributes. We propose modularization of XQueries for sorting algorithms and for node filters described in Section 6.3.3.

This modularization had to be done by dynamic generation of different parts of XQuery. Each part is generated by one PHP class which implements interface in Listing 6.3.

Listing 6.3: Interface for XQuery sorting algorithms

```
interface SortQuery {
  public function getQuery($params);
}
```

Function **getQuery** should return string where is part of XQuery with algorithm returning the value of key used by compare function. Example of such function is in Listing 6.4. Inside each class we can implement different sorting algorithm based on different criteria. Once we will need to change one algorithm for another it would be easy to replace one implementation for another just by changing parameter in function call.

Listing 6.4: Example of sort function for XQuery

```
class ReputationSortQuery implements SortQuery {
  public function getQuery($params){
    return '
      order by (
        if ($i/ranks/reputationScore) then xs:double($i/ranks/reputationScore)
        else xs:double(0.0)
      )
      descending';
  }
}
```

This construction should solve problem with many forks of function `getPost` mentioned in Section 5.2.2. It also provide great interface for future implementation of new sorting algorithms based on different criteria.

6.3.3 Modular XPath

As we dynamically create XQuery we may dynamically create also parts of XPath which identify nodes in database. We will use it in construction of XPath which selects articles according to its attributes. Conditions inside XPath are in conjunctive normal form, thus we can transform each previously created XPath conditions to it with same functionality.

For this purpose we propose `Filter` interface listed in Listing 6.5. If we would like to add new property to articles, according we may want to filter, we just needs to implement class with algorithm which generates needed condition in XPath. During call of function which creates XQuery we would pass new filter as a parameter. Such function has to call `getQuery` function and result use in XPath. In Listing 6.6 we implements basic filter for articles which are tagged by *tag* from user request.

Listing 6.5: Filter interface for XPath conditions

```
interface Filter {
  public function getQuery($params);
}
```

Listing 6.6: Example of XPath filter for tags

```
class TagFilter implements Filter {
  public function getQuery($params){
    if (isset($params['tag']))
      return '[tags/tag="'.formatText($params['tag'], false).'"]';
    return "";
  }
}
```

Functions that generates XQuery with XPath may include support for multiple filters. These functions need only to add array as additional parameter to their interface where will be stored classes implementing interface Filter. Each of these classes would produce its part of XPath condition.

Modularization of XPath using PHP solves problem with changing interface of function `getPost` each time we would like to add new property to article (or any other element) inside database.

6.3.4 Portlet XSL Template

According to our model described in Section 6.1 each portlet generate templates used for generation of proper output from generated XML in previous step. There exists two ways how portlet can generate templates:

- **staticaly** - templates are static files on disk and `getXSLlist` function returns paths to templates
- **dynamically** - templates are generated at runtime and returned from `getXSLlist` function

Templates generated by portlet has to match its parts of generated XML in case of single merged XML without namespaces. As we proposed in previous Section 6.3.1 portlet generated parts should have unique attribute value to be properly matched by template. In that case, core has prepared single XML file that consists of many portlets generated XML parts. Once dynamic XSL is generated, it contain each template returned by portlets. There have to be one which will match `/` root element and call other templates in proper order.

In Listing 6.7 is example of template for portlet `Lang`. Core generates main XSL template which import templates from all portlets including one from example. One of template have to contain match for root element. This template is executed and calls `xsl:apply-templates` for each element module in proper order according to settings passed during XML generation phase. Each template match its part of XML according to unique identifier *name* and generates proper HTML output.

Listing 6.7: Proposed form of template generated by portlet

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

<xsl:template match="module[@name='Lang']">
    <!-- HTML output for portlet -->
</xsl:template>

</xsl:stylesheet>
```

Example in Listing 6.7 template will match XML part generated by portlet `Lang`. This portlet should return template in such form. There have to be exactly one portlet that generate template which match `/` root element and will be used as a main layout defining position of each portlet.

6.4 Improving Database Performance

In this Section, we describe solution for problem with database performance described in Section 5.2.7.

As we noted in Section 5.2.7, the main bottleneck was I/O wait. We had created ramdisk and then we moved temporary file to ramdisk. This caused almost zero seek time and immediate data access. We have tested several XQueries and we compared their execution time. There were speed up from 0 do 10 times for single non-concurrent XQuery that were just reading data. We have tested XQueries that we are using in our code. Most notable difference was on XQuery from Listing 6.8.

Listing 6.8: example XQuery

```
declare function local:getAllTags($lang as xs:string) as element(tag)*{
  for $posts in collection("weblog")/user/blog/post
    [@status="published"][contains($lang,@lang)],
  $tags in $posts/tags,
  $tag in distinct-values($tags/tag),
  $tagCanonical in collection("weblog")/tags/tag[alt=$tag]
    [normalize-space(data(@canonical))!=""]
    [normalize-space(data(@canonical))!="Test tag"]
  return
    <tag id="{$tagCanonical/@id}" post="{$posts/@ID}">
    {normalize-space(data($tagCanonical/@canonical))}
    </tag>
};

declare function local:getUserTags($userID as xs:string, $lang as xs:string)
  as element(tag)*{

  for $posts in collection("weblog")/user[@ID=$userID]/blog/post
    [@status="published"][contains($lang,@lang)],
  $tags in $posts/tags,
  $tag in distinct-values($tags/tag),
  $tagCanonical in collection("weblog")/tags/tag[alt=$tag]
  return
    <tag post="{$posts/@ID}">
    {normalize-space(data($tagCanonical/@canonical))}
    </tag>
};

declare function local:filterRelatedTags($tags as element(tag)*, $tag as xs:string)
  as element(tag)*{

  for $selected in collection("weblog")/tags/tag[@canonical=$tag]
  return
    $tags[@post=$tags[text()=$selected/alt/text()]/data(@post)]
};

declare function local:filterUserTags($tags as element(tag)*, $user as xs:string)
  as element(tag)*{

  for $userPosts in collection("weblog")/user[info/nick/text()=$user]/blog/post/@ID
  return
    $tags[@post=$userPosts]
};

declare function local:countTags($tags as element(tag)*) as element(tag)*{
  for $tag in distinct-values($tags)
  order by count($tags[.=$tag]) descending
  return
    <tag id="{subsequence($tags[.=$tag]/@id,1,1)}" count="{count($tags[.=$tag])}">
      {$tag}
    </tag>
};

declare function local:log($n as xs:double) as xs:double{
  if ($n>1) then local:log($n div 2)+1 else 1
};

let $tags:=subsequence(local:countTags(local:getUserTags("u9","sk")),1,50)
  (: let $tags := for $tag in $tags
    return <tag count="{local:log($tag/@count div 100) * 100}">{$tag/text()}</tag> :)
  let $tags := for $tag in $tags return <tag count="{$tag/@count}">{$tag/text()}</tag>
  let $maxTagCount := max($tags/@count) let $minTagCount := min($tags/@count)
  let $tagsResult:= for $tag in $tags order by lower-case($tag/text()) ascending
    return
      <tag percent="{(($tag/@count - $minTagCount)div ($maxTagCount - $minTagCount+1))}">
        {$tag/text()}
      </tag>
  return $tagsResult
```

In case of concurrent reading XQueries, we had speedup from 2 to 100 times. Write non-concurrent queries was slower from 0 to 3 times. Write queries depends on locating xml part, that are going to be modified and then are written on disk. We can also speed up this part, using same trick (with ramdisk), but we can lost database when server crash. Temporary file can be deleted without any penalty. We don't know about any other workaround at this time. The only definitive solution would be fix in Sedna itself.

For problem with reaching limit number of connections to database at the same time, we propose to use only one instance of class `Sedna` during application lifetime.

6.5 Introducing Hooks

We would need mechanism of how to extend different parts of application without altering existing code. We may use it for other courses which would like to use portal engine in future. Alternatively we may want to use portlet architecture but we may want to bind two or more portlets together without code separation. This can be done using hooks.

Proposed model of portlets allow good separation of code, but we may want to call function for processing one feature in portlet for another feature in middle of processing input.

We propose `HookManager` with following interface and behaviour:

Listing 6.9: Interface for hook managment

```
class HookManager {
  public function registerHook($action, $function, $class=null);
  public function callHook($action, $params);
}
```

- **registerHook** - call function `$function` from class `$class` at desired place in program
- **callHook** - call all functions registered for `$action` with parameters inside array `$params`

For example, we got request with new article. We want to process tags related task right after checking validness of input article but before data are written to database in portlet `Article`. Without hooks, we have to add some explicit functionality to portlet `Article` which is related to tasks. With hooks, we will call every function which will register with this purpose at certain point. In this case tags will register its function for key *articleCheckDone*. In portlet `Article` we call all functions registered for key *articleCheckDone* at certain point. In Listing 6.10 is part of portlet for managing tag related features and in Listing 6.11 is a part of portlet for managing articles.

Listing 6.10: Example of program extending main one using hooks

```
registerHook('articleCheckDone', prepareTags);
...
function prepareTags($params){
  //do something
}
```

Listing 6.11: Example of main program for hooks

```
//do something
...
callHook('articleCheckDone',$article);
...
//do something
```

New model should improve developers overview over system. For backward compatibility, analysed in Section 5.2.6, with links from other pages we remain in identifying page type by one `GET` parameter. Every portlet should demand what to do according to this parameter. Since modules are grouped by same functionality (ex. everything related to article (reading, writing,...)) this should solve problem with classes functions analysed in Section 5.2.1). Also connection, analysed in Section 5.2.5, with courses would be cleaner since there exists option to inherit new module from old one and extend it with specific functionality.

Chapter 7

Implementation

In this chapter we will describe our implementation. We implemented almost everything from proposed solutions. In cases where we had several options we choose one and we will give reason why. On Figure 7.1 is class diagram of main application.

7.1 Core

Main application core consists of class `Controler`. Implements function `Serve` which runs entire process described on Figure 6.1 and function `registerPortlet`. Main core is fully implemented in PHP. We are using method of dynamically merging XML generated by portlets into one single XML for faster processing and to avoid merging of output which would be even more complicated.

We decided to use *dynamic template creation with imports* because we don't need dynamically generated stylesheets. We don't use static merging template because generation is straight forward and we won't alter input XML with additional data needed for imports of static templates.

Main flow consist of these steps:

1. Inside constructor initialize main libraries (Database, User, Session, XSLT, HookManager, ...) and making them available through array `$GLOBALS`

2. Include list of files defined in settings. Included files should contain code of portlets.

3. Create an instance of each registered portlet

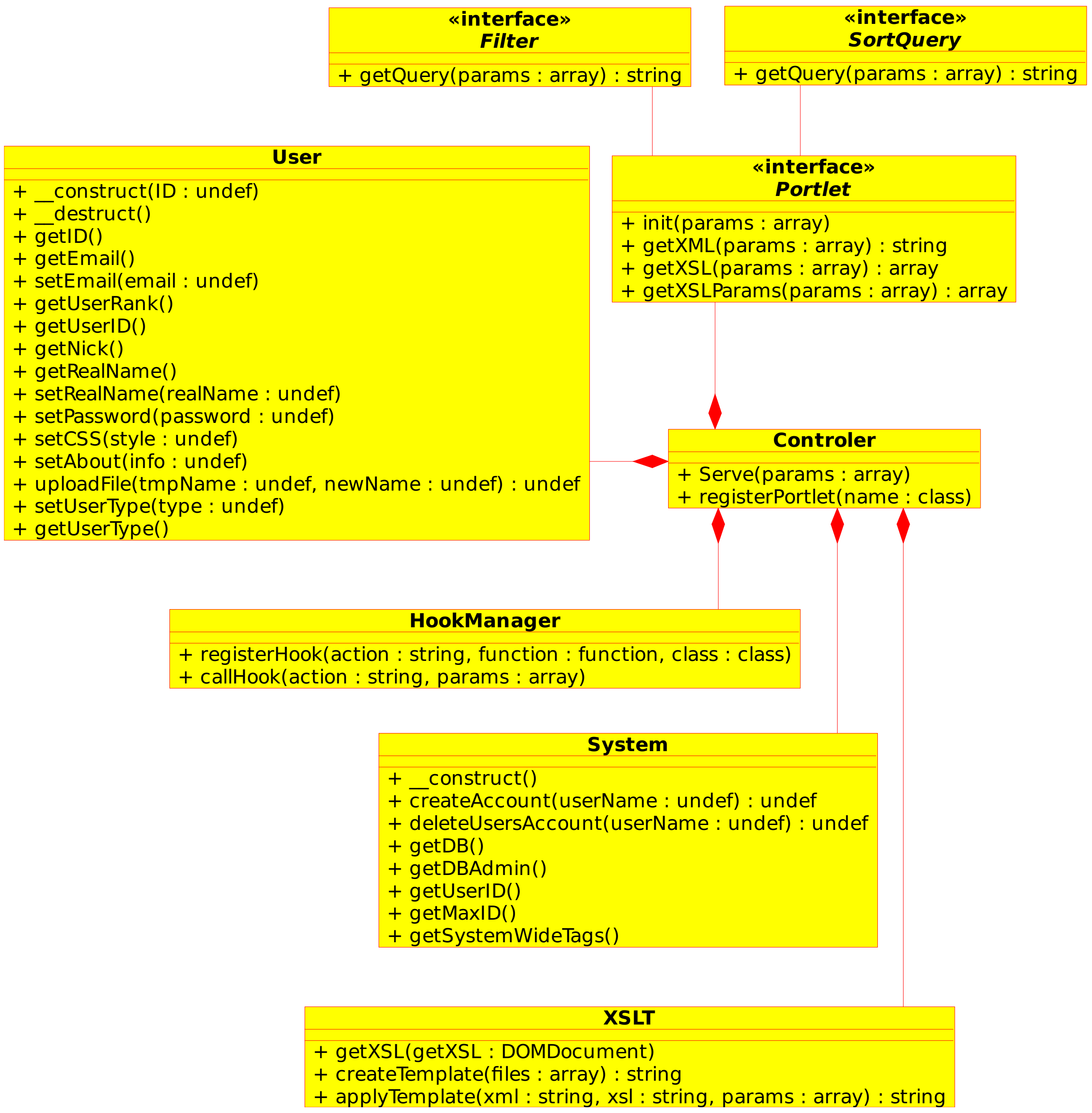

«interface»
Filter
+ getQuery(params : array) : string
«interface»
SortQuery
+ getQuery(params : array) : string
User
+ __construct(ID : undef)
+ __destruct()
+ getID()
+ getEmail()
+ setEmail(email : undef)
+ getUserRank()
+ getUserID()
+ getNick()
+ getRealName()
+ setRealName(realName : undef)
+ setPassword(password : undef)
+ setCSS(style : undef)
+ setAbout(info : undef)
+ uploadFile(tmpName : undef, newName : undef) : undef
+ setUserType(type : undef)
+ getUserType()
«interface»
Portlet
+ init(params : array)
+ getXML(params : array) : string
+ getXSL(params : array) : array
+ getXSLParams(params : array) : array
Controler
+ Serve(params : array)
+ registerPortlet(name : class)
HookManager
+ registerHook(action : string, function : function, class : class)
+ callHook(action : string, params : array)
System
+ __construct()
+ createAccount(userName : undef) : undef
+ deleteUsersAccount(userName : undef) : undef
+ getDB()
+ getDBAdmin()
+ getUserID()
+ getMaxID()
+ getSystemWideTags()
XSLT
+ getXSL(getXSL : DOMDocument)
+ createTemplate(files : array) : string
+ applyTemplate(xml : string, xsl : string, params : array) : string

Figure 7.1: Class diagram of our application

4. Call init function of each portlet (where input data has to be processed)

5. Call all hooks registered to name of requested page type

6. Get XML from each portlet and merge them together

7. Get list of XSLs from each portlet and merge them using using *dynamic template creation with imports* method to main template
8. Get input parameters from each portlet and pass them to XSLT processor
9. Process merged XML with main template using XSLT library and send result to output

It is almost implementation of model from proposed solution. We have added two steps. Step 5. which allow portlets to create specific functions for each page type (instead of creating decision tree inside `init` function). Step 8. was added to allow portlets to pass parameters for stylesheet not through XML but in semantically correct way using function of XSLT processor.

7.2 Libraries

Libraries `Sedna`, `Session`, `User`, `System` were left untouched. Library `XSLT` has been extended by function which generates templates with imports.

We are also using following libraries and interfaces:

- **User** - implements abstract methods over users data
- **System** - provide access to database and implements several functions related to database
- **Session** - provide methods for manipulation with informations related to current session
- **HookManager** - used for controlling hooks
- **XSLT** - library for merging stylesheets and processing XML with merged stylesheet
- **Sedna** - implement methods for working database
- **Portlet** - interface for portlets described in Section 6.3
- **Filter** - interface for XPath conditions used inside XQueries described in Section 6.3.3
- **SortQuery** - interface for sort algorithms inside XQuery described in Section 6.3.2

7.3 Portlets

We reimplemented everything from original code into portlets. Portlets are separated according to their functionality described in Section 6.1. For layout generation we are using special portlet layout. It is the only one which has template matching `/` (root) element. Its description is in Section 7.3.4.

We have implemented following portlets:

- **Article** - reading, writing, voting, listing of articles
- **BestRated** - portlet showing top 10 best rated articles
- **Lang** - changing portal language and showing language menu
- **Layout** - management of portal layout according to page type
- **Login** - manage login and logout function and provide login form
- **Menu** - displays menu according to user type (visitor, logged user, student)
- **MostRead** - portlet with top 10 most read articles
- **Search** - search field
- **TagCloud** - dynamic tag cloud with top 50 tags on our portal

7.3.1 PHP Part

Every portlet has its PHP part and implements interface described in Section 6.3. Each has access to libraries mentioned above through array `$GLOBALS`. Since we decided for static templates, function `getXSLlist` returns array of strings. Each string represents path to `.xsl` file with template. Currently we defined this page types (names were preserved from original code to maintain compatibility) that should be recognized inside portlets to make desired action:

- **blogView** - blog of user specified in `userID`
- **commentForm** - show comment form replaying to comment with `commentID` or post with `postID`

- **loginForm** - show page with login form
- **login** - login action
- **logout** - logout action
- **getTagBrowserSVG** - used by tag browser for generation of graph with tags relationship in SVG format
- **portalView** - main page
- **postForm** - form for creating or editing existing (`postID`) post
- **postView** - show article with `postID`
- **registration** - show registration form
- **rss** - RSS version of portal
- **search** - search mode for and article with specified attributes
- **settingsForm** - page with specific user (`userID`) options
- **showStudentXSL** - used by courses portlet for rendering users blog using its own template
- **uploadForm** - page for user (`userID`) file management
- **userPosts** - page with management of users (`userID`) posts
- **vote** - user vote for an opened article or comment (deprecated)
- **ajax** - special type for ajax requests

Original code was using ajax[1] for preview of article during writing. Since ajax returns only small portion of output and not full page, portlets that uses ajax has to handle this type and also use parameter `portlet` which will specify which portlet has to render output. No portlet has to generate output for unknown type.

[1]Asynchronous JavaScript and XML

7.3.2 XML Part

Each portlet has to return string with **well formed** XML. We don't use namespaces to make implementation simplier. Each data part has to be inside element *module* with attribute *name*. Its name is used as a value. Example of such XML is in Listing 7.1. As we don't use namespaces in PHP, portlet name is unique in our application.

Listing 7.1: XML output format for portlets

```
<module name="article">
  <!-- module data -->
</module>
```

7.3.3 XSLT Part

There should be no complex calculations in templates. Everything what can be calculated in PHP should be done that way. XSL templates have to be used only for formatting of output. Each portlet have to return list of `.xsl` files using `getXSLlist` function if it wants to produce output. Each template have to match XML data generated by same portlet. This should be done using `xsl:template` construction. Example is in Listing 7.2.

Listing 7.2: Interface for hook managment

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

<xsl:template match="module[@name='Article']">
<!--
transformation for (HTML) output
-->
</xsl:template>
</xsl:stylesheet>
```

7.3.4 Portlet Layout

Portlet layout is used for placing final content of each portlet into right place in right order. It also implements support for portlet order customization available for each user on its own blog. During phase of XML generation it generate description of position of each portlet, such as listed in Listing 7.3. Then it generates XSL template and calls `xsl:apply-templates` in right order over XML data. Example of dynamically generated main template is in Listing 7.4.

Listing 7.3: Example of XML generated by layout

```
<module name='Layout'>
  <portlet name='TagBrowser' col='1' row='1'/>
  <portlet name='Articles' col='1' row='2'/>
  <portlet name='BestRated' col='2' row='1'/>
  <portlet name='MostRead' col='2' row='2'/>
</module>
```

Listing 7.4: Example of XSL with two column layout

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:func="http://exslt.org/functions">

<xsl:template match="/">
<div class="colA">
<xsl:for-each select="output/module[@name='Layout']/portlet[@col='1']">
        <xsl:sort select="@row"/>
        <xsl:variable name="name" select="@name"/>
        <xsl:apply-templates select="/output/module[@name=$name]"/>
</xsl:for-each>
</div>

<div class="colB">
<xsl:for-each select="output/module[@name='Layout']/portlet[@col='2']">
        <xsl:sort select="@row"/>
        <xsl:variable name="name" select="@name"/>
        <xsl:apply-templates select="/output/module[@name=$name]"/>
</xsl:for-each>
</div>
</xsl:template>
</xsl:stylesheet>
```

`xsl:template` has one additional attribute - `mode`. It could be used as a key for calling right template, but it does not match correct data.

We have three types of layout on our blog. Three column (Figure 7.2) for title page, two column (Figure 7.3) for user blog and single column (Figure 7.4) for user settings, posts and other management tasks.

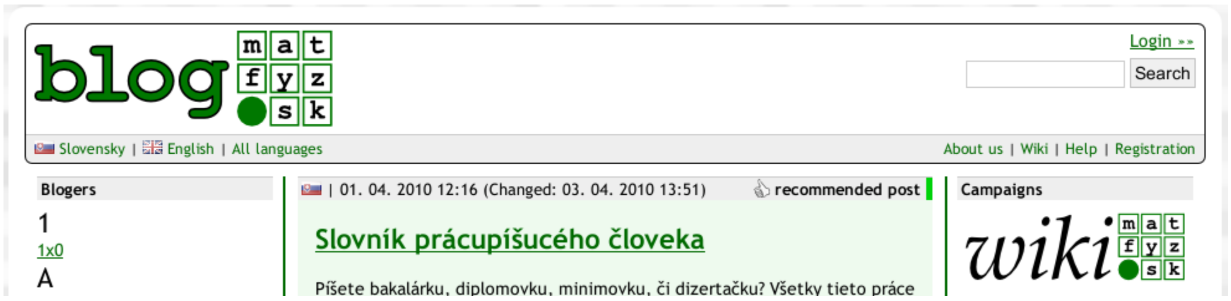
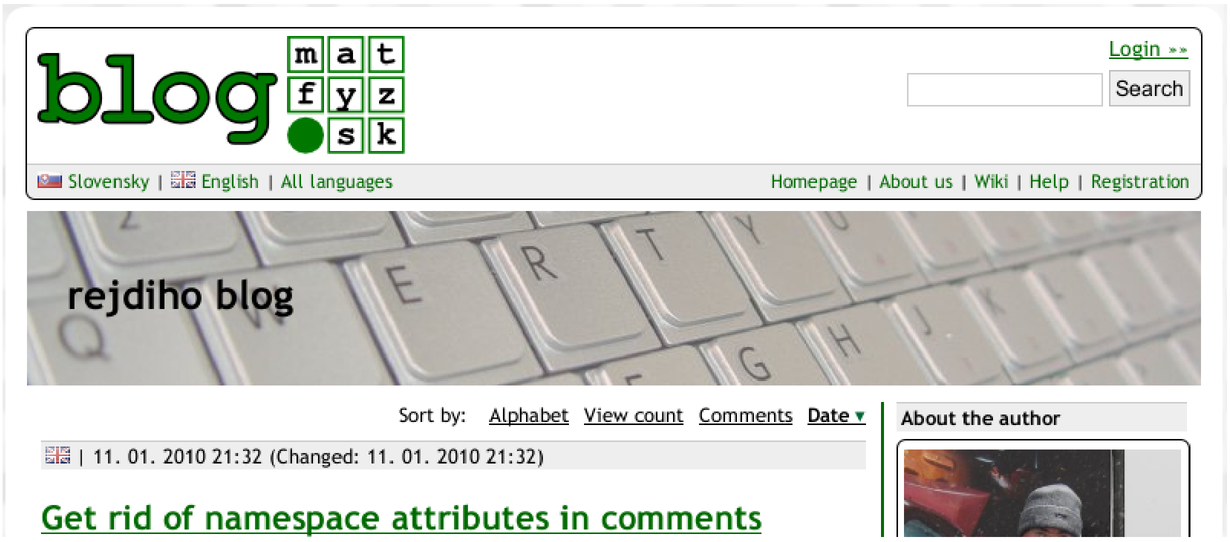

Login »
Search
Slovensky | English | All languages
About us | Wiki | Help | Registration
Blogers
1
1x0
A
| 01. 04. 2010 12:16 (Changed: 03. 04. 2010 13:51)
recommended post
Slovník prácupíšucého človeka
Píšete bakalárku, diplomovku, minimovku, či dizertačku? Všetky tieto práce
Campaigns

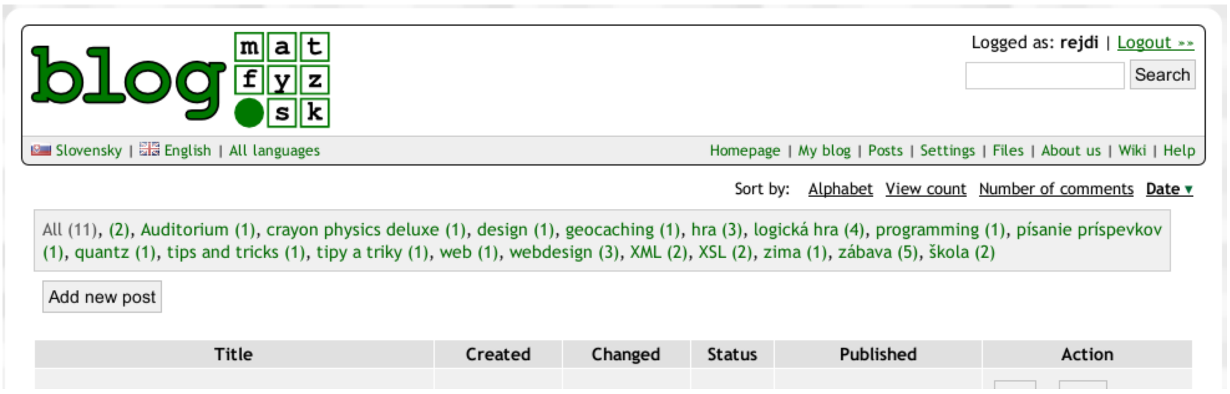
Figure 7.2: Three column layout


blog
m a t
f y z
s k
Login »»
Search
Slovensky | English | All languages
Homepage | About us | Wiki | Help | Registration
rejdiho blog
Sort by: Alphabet View count Comments Date ▾
About the author

| 11. 01. 2010 21:32 (Changed: 11. 01. 2010 21:32)

Get rid of namespace attributes in comments

Figure 7.3: Two column layout


blog
m a t
f y z
s k
Logged as: rejdi | Logout »»
Search
Slovensky | English | All languages
Homepage | My blog | Posts | Settings | Files | About us | Wiki | Help
Sort by: Alphabet View count Number of comments Date ▾
Add new post

All (11), (2), Auditorium (1), crayon physics deluxe (1), design (1), geocaching (1), hra (3), logická hra (4), programming (1), písanie príspevkov (1), quantz (1), tips and tricks (1), tipy a triky (1), web (1), webdesign (3), XML (2), XSL (2), zima (1), zábava (5), škola (2)

Title	Created	Changed	Status	Published	Action

Figure 7.4: One column layout

Chapter 8

Conclusion

Our work consists of three main parts. In first we analysed original portal and pointed out to main problems we had found. In second part we proposed new architecture usable for our blog and in third part we describe our implementation a proposed solution.

During analysis we revealed several problems of original code. We found that original code was using horizontally oriented architecture which lead to increasing number of functions in classes with every new feature we would like to add. Filtering of articles were parametrized through function interface which was changed each time we added new filter parameter. Changing function interface lead to another regressions. Moreover filter function used sorting algorithm inside XQuery. This algorithm had to be changed according page type. Original code contained three copies of filter function for articles, each with different algorithm.

Original XSL templates contained calculations and many duplicate logic which were also in main PHP. Moreover templates directly opened files used for cached results which is not transparent. Transparency is important in case we added additional modification of calculated data in PHP we had to add it also to XSLT. With transparent caching modification of templates won't be needed. Portal has been also used for several courses. For each course there where modifications included directly in portal code without any separation. This lead to higher code complexity. Finally we had problems with database performance.

In second part we proposed new model of application architecture by using modular programming and several other techniques, such as combination of vertical and horizontal separation or code generation at run-time. New model introduces portlets. Portlets are basic independent modules providing small features available on our blogs. Our portlets uses vertical separation architecture. Each portlet is separated horizontally to business logic and presentation logic. We also proposed solution for modular filtering of XPath queries using PHP run-time code generation. We proposed similar method for modularization of XQuery which we parametrized by custom sort function. We also proposed preliminary solution for improving database performance without changing database itself. Finally we proposed method of hooks as additional modular method for extending portal functionality in future.

We reimplemented original portal to new architecture. In this work we also focus on several implementation details where we describe why we choose several options when we had more choices. As a consequence of problems mentioned in analysis, during development in past we remarked many regressions and development of new features was very slow. New architecture would speed up development, reduce number of regressions and allow making changes more easier without side effects. During reimplementation we found out that new core is easily understandable, reimplementation was quite fast and new model separate each feature we have, enough. Moreover new model should easily allow us to implement planned features.

Chapter 9

Related Work

9.1 Teamwork Organization

During research on our portal I became a team leader. This portal was used by other students for their diploma and bachelor thesis. Namely:

- **Juraj Frank** - worked on Tags and their semantic with ontology in background
- **Juraj Ďuďák** - worked on many graphical representations of tags
- **Marek Sivčo** - portal layout with respect to accessibility and usability
- **Martin Králik** - portal security
- **Marek Kováčik and Daniel Adam** - connection between blog engine and course *Modern approaches to web design*
- **Tomáš Jurík** - Advanced XML editor for students of *Modern approaches to web design*

We had almost regular meetings where we discussed our next steps, we planned every release cycle and solved other common problems. I also did code reviews that result into suggestions how to make better integration with our code architecture.

From beginning we used SVN versioning system for code repository, that helped us with coordination, code conflicts resolution and reverting changes when needed. I took a lesson and for future projects that will contain many different functionality developed in parallel I will use GIT, since it is more friendly to parallel development of branched code. However its price is in more work during code merge process, it would easily allow us to defer some functionality for next release cycle.

9.2 Database Issues

For main data we used Sedna XML database as this was part of previous research. Unfortunately this database is still in heavy development. However, authors of Sedna declared that it has ACID properties, we had several crashes that leads to dataloss.

I created script that made regular backups of DB. During research we helped Sedna team to identify several bugs that resulted in

- corrupted database
- suddenly stopped (crashed) Sedna
- DB governor was in indefinite loop (bug in Sednas deadlock detection algorithm)

This bug were reported.

Sedna is logging every XQuery into text file with specific format for bug reports. Once we've needed replay queries from this log files. Unfortunately there were no utility available which was able to parse those log files and extract only write / delete queries. So I had to write one using *flex* and made those queries replayed over DB from last backup.

Listing 9.1: Code for Flex

```
%x query
%option noyywrap
%%
---[ ]+update[^\r\n]+[\r\n]+---[ ]+<ranks>          /* ignore */
"---    update"           { BEGIN(query); printf("update"); }
"---    CREATE DOCUMENT"  { BEGIN(query); printf("CREATE DOCUMENT"); }

<query>^---       /* ignore */
<query>^[^-]      BEGIN(INITIAL);
<query>.          printf("%c", yytext[0]);
a|[^a]            /* ignore */
%%
int main ( int argc, char** argv )
{
  yylex();
}
```

This code was transformed using `flex -i file.l` and then compiled using gcc. Final program take sedna logs from standard input and put on output only XQueries that make changes in database, except those that change user ranks (that are counted regularly using our scripts and during recover process were just a waste of time).

However, database has still problems, we are still using Sedna XML database, which was choose by Anton Kohutovič [Koh08], because authors are providing feedback and great support.

Bibliography

[Cha02] D. Chamberlin. Xquery: An xml query language. *IBM Syst. J.*, 41(4):597–615, 2002.

[fSPR10] Institute for System Programming RAS. Sedna documentation. `http://modis.ispras.ru/sedna/documentation.html`, 2010.

[Gil04] W. Jason Gilmore. *Velká kniha PHP5 a MySQL.* Zoner Press, 2004.

[Gre08] Derek Greer. The art of separation of concerns. `http://www.aspiringcraftsman.com/2008/01/art-of-separation-of-concerns/`, 2008.

[JC10] editor J. Clark. Xsl transformations (xslt) version 1.0. `http://www.w3.org/TR/xslt`, 2010.

[Koh08] Anton Kohutovič. blog.matfyz.sk community portal. Master thesis, Comenius university, 2008.

[MAea10] editors Mehdi Achour et al. Php manual. `http://www.php.net/manual/en/`, 2010.

[Mar03] Robert Martin. *Agile Software Development.* Twayne Publishers, Boston, 2003.

[Mey97] B Meyer. *Object-oriented software construction.* Prentice Hall, second edition, 1997.

[MFea10] editors M. Fernandez et al. Xquery 1.0 and xpath 2.0 data model. `http://www.w3.org/TR/xpath-datamodel`, 2010.

[MK10] editor M. Kay. Xsl transformations (xslt) version 2.0. `http://www.w3.org/TR/xslt20`, 2010.

[Pec07] Rudolf Pecinovský. *Návrhové vzory.* Computer Press, 2007.

[rB02] Jiří Bráza. *PHP4 učebnice základů jazyka.* Grada Publishing a.s., 2002.

[RO06] Zeki Bayram Ruhsan Onder. *Implementation and Application of Automata*, volume 4094/2006. Springer Berlin, 2006.

[SBea10] editors S. Boag et al. Xquery 1.0: An xml query language. `http://www.w3.org/TR/xquery`, 2010.

[W3C06] W3C. Extensible markup language (xml) 1.0 (fourth edition). `http://www.w3.org/TR/REC-xml`, 2006.

[Wik10] Wikipedia. Portlet — wikipedia, the free encyclopedia. [Online; accessed 28-April-2010], 2010.

Abstrakt

Táto práca sa zaoberá analýzou pôvodného kódu portálu blog.matfyz.sk, návrhom jeho novej architektúry a následne jej implementáciou. Počas návrhu je využité modulárne programovanie a niekoľko princípov z oblasti softwareového inžinierstva. Modulárny návrh je potrebné vytvoriť aj s ohľadom na rôzne typy jazykov a technológií, ktoré sa používajú v tejto webovej aplikácii.

Počas analýzy sa zaoberáme najväčšími problémami portálu ako je zle navrhnuté delenie tried, či rozšírenie o dodatočnú funkcionalitu zapracovanú priamo do portálu zložitým a neprehľadným spôsobom. V návrhu predstavujeme nový model obsahujúci kľúčové prvky z oblasti modulárneho programovania aplikované na jazyky ako XSLT, XQuery či XPath a technológie ako XML, ktoré neumožňujú jednoducho modularizovať dáta alebo kód.

Na záver predstavujeme implementáciu s návrhom tried a spôsobom prepojenia jednotlivých technológií a jazykov.

Kľúčové slová: Weblogy, Modulárny dizajn, XSLT, XML, XML databáza