University of Zurich[UZH]

FACHARBEIT – Communication Systems Group, Prof. Dr. Burkhard Stiller

Implementation of a Publication System

Michel Hunziker
Zürich, Switzerland
Student ID: 10-721-108

Supervisor: Christos Tsiaras, Dr. Thomas Bocek
Date of Submission: May 31, 2013

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Facharbeit
Communication Systems Group (CSG)
Department of Informatics (IFI)
University of Zurich
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland
URL: http://www.csg.uzh.ch/

Abstract

Die Communication System Group (CSG) hat ein online Publikationssystem, bei welchem Publikationen angeschaut, hinzugefügt, geändert und gelöscht werden können. Zusätzlich gibt es Merlin – ein zweites Publikationssystem der wirtschaftswissenschaftlichen Fakultät der Universität Zürich. Beide Systeme waren nicht voll kompatibel zueinander, da die Publikationstypen und -attribute nicht übereinstimmten. Ziel dieser Arbeit war es, das Publikationssystem der CSG mit Merlin abzugleichen. Da die alte Applikation strukturelle und sicherheitsrelevante Mängel hatte, wurde eine komplett neue Implementation entwickelt. Das neue Publikationssystem folgt einer klassenbasierten Model-View-Controller-Architektur mit einer klaren Trennung der Verantwortlichkeiten. Um die Kompatibilität mit Merlin zu gewährleisten wurden 16 neue Attribute hinzugefügt und die verschiedenen Typen mit den in Merlin verfügbaren, sowie benötigten Attributen abgeglichen. Die Datenbank blieb dagegen strukturell identisch, weshalb sie in zukünftigen Arbeiten in mehrere Tabellen aufgeteilt und normalisiert werden sollte.

The Communication System Group (CSG) has an online publication system, where publications can be viewed, added, modified, and deleted. There is also Merlin, a second publication system of the Faculty of Economics, Business Administration and Information Technology of the University of Zurich. Both systems were not fully compatible to each other, as the publication types and attributes did not match. The goal of this thesis was to make the CSG implementation compatible with Merlin. As the old application had structural and security issues, a completely new implementation was developed. The new publication system follows a class based Model-View-Controller architecture with a focus on separation of concerns. To ensure the compatibility with Merlin 16 new attributes were added and the various types were synchronized with the available and required attributes of Merlin. However, the database structure remained the same, which is why it should be separated to multiple tables and normalized in future work.

Acknowledgments

I would like to thank Prof. Burkhard Stiller for giving me the opportunity to complete this "Facharbeit" for the Communication System Group, even if it was originally announced as a "Vertiefungsarbeit". In addition, he provided a lot of examples for the expected data. I would also like to thank Christos Tsiaras and Dr. Thomas Bocek for their great and always friendly support.

Contents

Chapter 1

Introduction

The Communication Systems Group (CSG) at the Computer Science Department of the University of Zurich has an online publication system [1], which allows the user to view, search, and export the existing publications. In addition, the web application provides functionality for the CSG staff members to create, update, and delete a publication entry. Besides the publication system of the Communication Systems Group exists Merlin [2], a second publication system, used by all the members of the Faculty of Economics, Business Administration and Information Technology of the University of Zurich. However, the publication system of the Communication System Group was not fully compatible with Merlin.

1.1 Motivation

The publication system was limited in several ways. First, the implementation of the CSG included less attributes than Merlin. Second, the required attributes of Merlin types did not always correspond with the CSG-implementation for the various publications – there were cases where an attribute was optional in the publication system of the Communication System Group, but required in Merlin. Third, the CSG publication system did not provide information to the staff about the expected format of the attributes and only performed basic data validation. Overall, all three points made it difficult to export an entry from the CSG publication system to Merlin. Furthermore, to synchronize the two systems an entry has to be exported from the CSG publication system as BibTeX and then imported in Merlin. However, the old implementation did not create valid BibTeX entries and, therefore, the BibTeX importer of Merlin could not import all provided attributes correctly. Thus, the person responsible for the synchronization had an increased overhead in the form of correcting the missing or invalid attributes.

1.2 Description of Work

The contribution of this thesis was to fix the described incompatibility between the CSG implementation and Merlin with the goal to allow a more sophisticated synchronization of the two different publication systems. The objectives were as follows:

- Every attribute which Merlin requires and is currently missing in the CSG implementation should be added.
- Mandatory and optional attributes should be the same in both implementations.
- The publication system should provide examples for the expected format of the attribute to achieve uniform data.
- All data should be filtered and validated to achieve a high data quality.
- The BibTeX export should create valid BibTeX entries to ease the synchronization between the two publication systems.

1.3 Thesis Outline

In the next chapter the old implementation will be analyzed, the resulting requirements are given, as well as the new implementation discussed. In the third chapter the new CSG publication system is evaluated and compared with the old implementation. Additionally, the resulting benefits are presented as well as some limitations of the new application discussed. In the last chapter the work is summarized. To close the thesis some ideas for future work are highlighted.

Chapter 2

Implementation

As presented in the last chapter, the Communication System Group already had a publication system based on PHP. The initial plan was to extend the existing code base. However, after reviewing the source code it became clear that this may not be a good idea. A big problem of the old code was that everything was mixed together. Most of the pages used HTML, PHP, CSS, and JavaScript in the same file, which lead to repetition, as various pieces had to be redefined on every page. This approach was particularly adverse for pages existing both as a public and a similar internal version (for example the home page where the internal version additionally includes hyperlinks to directly edit a publication). The PHP implementation itself only relied on global functions containing a lot of repeated code with no separation of concerns. For example, a single function was used to grab the query parameters of the HTTP request, get the publications from the database, format the values, and output the formatted string. In addition, the implementation had security issues. As no values were escaped, the system was vulnerable to cross side scripting, allowing everyone to inject arbitrary code into the page by using query parameters (see Figure 2.1 for an example). Using the same technique, it was possible to execute custom SQL queries. Moreover, if an invalid query was executed, the error was directly presented to the user exposing the database schema to the user.

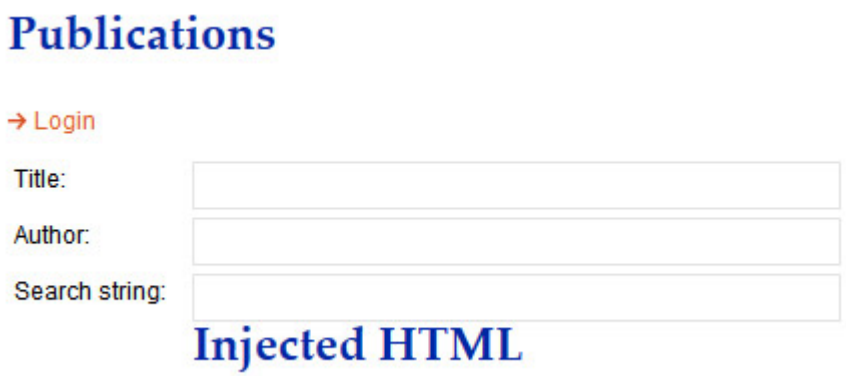

Figure 2.1: Example of injected HTML code in the old publication system

2.1 General Design

Since the old implementation was very hard to test, extend, and maintain, it was decided to redesign the web application. The focus of the new implementation was to build a secure application with separated concerns and to eliminate repetitive elements. For this reason a class based implementation was pursued where each class only has one responsibility. The classes follow the principle of dependency injection. This allows to set every relevant dependency to other objects from the outside making the code base testable and open for further extensions, as the classes can easily be exchanged with new ones. To enforce a clean separation a test driven development approach using PHPUnit [3] was followed where the API with the expected outcome is defined first and the concrete implementation follows afterwards. Such an approach was especially useful to ensure that the new CSG publication system still included every functionality of the old implementation. Therefore, the expected result was well defined, making it ideal for unit tests.

The new implementation follows a Model-View-Controller architecture to further address the named focuses. The model contains the business logic and is the biggest part of the application, whereas the controller only glues the various pieces together. The view part renders the data it received from the controller and does not contain any logic. In the following sections the model and the view are discussed in depth.

2.2 Model

As mentioned in the previous section, the Model is the biggest part in the application. At the core of the application is the `Entry` class. It acts as a holder for various constants (e. g. the different publication types), which are used by different parts of the application. This central organization of the constants makes it, for example, very easy to add new options, as they simply have to be added to the corresponding array. By doing so, the new option is immediately available in the user interface and is recognized as a valid value for the database. Besides the constants, this class also provides metadata for the attributes, which is used in the view (e. g. the label or the examples for each attribute).

2.2.1 Publication types and attributes

Each publication is represented by an instance of the `Entry` class. Therefore, it contains all possible attributes the publication can have. Since the new publication system had to be backward compatible with the old publication system of the Communication System Group, the existing attributes were used as a base. The attributes required by Merlin were then used to extend the base. Additionally, all optional attributes which are also official BibTeX attributes were added. The other optional attributes of Merlin were not added, as they cannot be imported using the BibTeX importer. This process resulted in a total of 32 attributes (16 new) for the new CSG publication system.

Another class (called `TypeMap`) was used to define which attributes should be available and which attributes are required for each publication type. This class provides a static method returning for every publication type an array with the attribute names and a flag for each attribute indicating whether the attribute is required (value is true) or optional (value is false). Therefore, it is very easy to change the requirement (by changing the flag for the attribute name) or to add or delete an attribute (by adding or removing the attribute name).

2.2.2 Filtering and validating input

The `TypeMap` class also has the mapping information between the attributes and types of the CSG implementation and BibTeX. This is used in the `BibtexParser` class to parse the type and attributes of a given BibTeX string, convert the attributes and values to the CSG format, and to return an array with all the parsed data allowing the user to import a publication using the standard BibTeX format as well as adding publications manually.

Either way, all incoming data is filtered and validated to achieve a high data quality. As this is a very common task in web applications, many frameworks exist for handling user input. One of the most popular is the Zend Framework – an open source framework built with a use-at-will design, which means the architecture of the framework is loosely coupled allowing to only use some selected components [4]. For the publication system the `Zend\InputFilter`, `Zend\Filter` and `Zend\Validator` were used. The `InputFilter` filters the provided data with the attached filters and uses the validators to validate the data. The `InputFilter` provides a method to determine whether the data is valid and returns error messages if not to inform the user about the invalid input. In the `InputFilter` class of the publication system several filters and validators of the Zend Framework were used as well as custom ones. For example, the pages attribute has a default filter to trim whitespace and a custom filter to replace the typographically correct en dash (both as character and LaTeX markup) with a hyphen as well as a custom validator to make sure the beginning page number is not greater than the ending page number (see Figure 2.2). For the digital object identifier and the ISSN custom validators were created as well.

```
'pages' => array(
    'filters' => array(
        $trimFilter,
        // replace double hyphens and the en dash with a single hyphen
        new Filter\PregReplace(
            array(
                'pattern' => '/\s*(-|\-{1,2})\s*/',
                'replacement' => '-',
            )
        ),
        $nullFilter,
    ),
    'validators' => array(new PageRange()),
),
```

Figure 2.2: InputFilter specification of the pages attribute

In addition, the `InputFilter` class contains advanced cross-validation to make sure some attributes are not set at the same time under certain conditions. It also accesses the `TypeMap` to determine which attributes should be considered for filtering and validation as well as which attributes should not be empty. If the provided data passes the strict validation process, the filtered values can be grabbed from the class. For some publication types these values are mixed with additional and predefined default values.

2.2.3 Database

Once the publication is filtered and valid, it is saved to the database. The `Database` class is based on PDO, a database abstraction layer included in the standard PHP distribution. Using this approach it is possible to use different underlying database systems without the need to change any code. The PDO instance is configured in a separate file and injected into the `Database` class on creation, which makes it easy to change the database details. Prepared statements are used in all cases to eliminate any SQL injection vulnerabilities. If a query is created dynamically using the HTML query, then every parameter name is matched against a whitelist and discarded if the name does not exist in the predefined list.

2.3 View

The separation of concerns was also implemented in the view part of the application. All inline JavaScript and CSS were extracted into a separate file. Both the JavaScript and the styles are minified to increase loading time. Additionally, all images were put together into a single image (sprite sheet) to reduce the number of calls to the webserver to a minimum. In order to prevent repetition a layout file was used, which defines the basic HTML structure (e. g. the doctype and the header). The various pages then only have to define the actual content of the page. If a page exists both as a public version and an internal version, the same view file is used. A `Url` class, which maps a given page name to the specific URL, is used to ensure valid links in such cases. If the page is viewed internally, the internal URL is automatically returned instead of the public URL.

Several formatters were created to format the various publications: `Text`, `Html`, `Bibtex`, and `Csv`. The `Text` formatter uses a predefined list of attributes. If an attribute in this list is available in the publication entry, it is added to the output. Each attribute also has a string, which will be added before the attribute, as well as a string, which will be appended (e. g. the value “doi:” is prepended to the attribute for the digital object identifier). Using this approach the formatter can handle the different attribute combinations of the publication types while formatting the values in the same style and order. Furthermore, the formatter will continue to work if the entries have missing data or the required attributes change. The `Html` formatter extends the `Text` formatter, but adds additional HTML markup (e. g. adding a hyperlink to the DOI). The `Csv` formatter prints all attributes separated with a semicolon and the `Bibtex` formatter returns valid BibTeX entries by using the `TypeMap` class to convert the attributes, types, and values to the BibTeX specific format.

Chapter 3

Evaluation

Since the application was built from the ground up, it was essential that every functionality of the old publication system is included in the new implementation. As a test driven development was pursued, this could be checked very easily. In the new publication system of the Communication System Group all 261 tests are passing. Every class has a code coverage of 100 % except the `Controller` class where only 11 of the 17 methods are covered with unit tests. However, based on the tests it can generally be assumed that every functionality of the old publication system is implemented in the new application.

3.1 Comparison with the old implementation

In contrast to the old implementation, the new publication system has several advantages. First, the code is now much better organized, separated, and extendable. This advantage is also visible in the static code analysis. PHP Mess Detector [5] was used to localize any potential problems. The source files were checked against all code size rules (e. g. excessive class complexity or length), controversial rules (e. g. accessing superglobals), design rules (e. g. using `eval`), naming rules, and unused code rules. The tool reported nine notices (mainly a too large `Entry` class), one warning (many dependencies in the `InputFilter` class, since the various filters and validators are directly instantiated), and no errors. The second advantage is the elimination of all code duplicates – PHP Copy/Paste Detector [6] does not report any copied code in the source and test files. Third, the code style (e. g. placement of the braces) is uniform across the whole application and follows the PSR-2 standard (checked by PHP CodeSniffer [7], which shows no errors or warnings).

Besides these more technical improvements, the new CSG publication system also includes enhancements for the user. All objectives outlined in the introduction were implemented. The new application includes the 16 missing required attributes of Merlin and has now the same attribute requirements as Merlin. In addition, several publication types were added for a better matching with Merlin. Furthermore, the various types are now grouped, which helps structuring the data. Moreover, several examples are now displayed to the user and all inputs are specifically filtered and validated to achieve a high data quality. On the

page to add and edit a publication the possible attributes are now dynamically displayed based on the publication type. In contrast to Merlin, it is even possible to change the publication type at any time. For the public user a new details page is available, which shows all values in a clear table. If the page is viewed internally, a direct export to Merlin is now possible as well (see Figure 3.1). For the export the new BibTeX formatter is used, which now produces a valid BibTeX string. The other output has improved as well, as every output is now escaped.

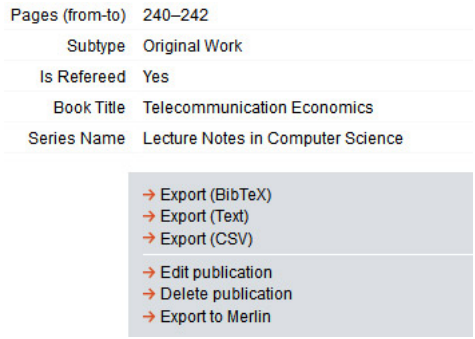

Figure 3.1: Extract of the new details page with the exports and internal options

3.2 Limitations

Nevertheless, the new CSG publication system still has some limitations. First of all, only unit tests were created and no integration tests. The various injected objects are always mock objects and no real instances are used. Second – as noted in the previous section – PHP Mess Detector reported a couple of notices about a too large `Entry` class, as the file is 1164 lines long and the class includes 37 fields. This is problematic in the database as well, as most of the attributes are left blank resulting in a lot of `null` values. The database is also limited, as it is not normalized. For example, if the author name of one publication is changed, all the other publications with the same author name will stay the same. This makes it also impossible to provide a high-performance auto-completion of the names to the user. The second problem PHP Mess Detector reported was the many dependencies in the `InputFilter` class. Therefore, it is difficult to change the filters and validators without changing the code in the class itself. The Zend Framework would provide a `ServiceManager` component to avoid this problem. However, as it is unlikely that a filter or validator has to be changed from outside the class, it was decided against using this component and keep it lightweight and simple. Another limitation is the handling of the pages in the public folder. Every page has a simple PHP file, which loads the bootstrap file and calls the correct action of the controller. To avoid repetition, it may be a better idea to just use one file for all pages to which all HTML requests are rewritten.

Chapter 4

Summary and Conclusions

In this thesis a new publication system for the Communication System Group was built from the ground up. The new implementation is now fully compatible with Merlin, as all attributes of the CSG publication system were updated to match the attributes of Merlin. Custom filters and validators were added to satisfy the high data quality of Merlin. The security vulnerabilities of the old implementation were removed by using prepared statements, whitelists, and escaping. The BibTeX export now creates entries in the correct format, which eases the synchronization between the two publication systems. In addition, the implementation was built with a focus on the separation of concerns. As this makes the code easy to test, the application has a good code coverage with unit tests. The separation of the different responsibilities has also the advantage that the code is easy to modify and extend. Thus, the current implementation provides a solid foundation for future extensions.

Further work should primarily optimize the database part. It would be a good idea to split the `Entry` class into multiple classes for the different publication types and use this approach in the database as well by creating multiple tables. Additionally, the database could be normalized in order to avoid repeated values and make the database more compact. Moreover, the normalized database would allow to introduce auto-completion in the user interface, i. e. where the author names could directly be selected from a list after typing the first couple of characters. Another area of work could be an even better synchronization between the CSG implementation and Merlin. Since the current export uses BibTeX to import a publication in Merlin, some attributes have to be added by hand, as they do not have a matching BibTeX attribute. In addition, at the time of writing, the BibTeX importer of Merlin did not work properly (sometimes the authors had to be removed from the BibTeX entry in order to make the import work, as Merlin probably had problems matching the string to an author ID). If Merlin provided an API to add a new publication, it would make the synchronization much more effective.

Bibliography

[1] Communication System Group. (2013, May 31). Publication system, [Online]. Available: http://www.csg.uzh.ch/publications.html.

[2] Software Evolution and Architecture Lab. (2013, May 31). Merlin, [Online]. Available: https://www.merlin.uzh.ch.

[3] S. Bergmann. (2013, May 31). PHPUnit, [Online]. Available: http://www.phpunit.de.

[4] Zend Technologies Ltd. (2013, May 31). Zend Framework, [Online]. Available: http://framework.zend.com/.

[5] M. Pichler. (2013, May 31). PHP Mess Detector, [Online]. Available: http://phpmd.org/.

[6] S. Bergmann. (2013, May 31). PHP Copy/Paste Detector, [Online]. Available: https://github.com/sebastianbergmann/phpcpd.

[7] G. Sherwood. (2013, May 31). PHP CodeSniffer, [Online]. Available: http://pear.php.net/package/PHP_CodeSniffer.

List of Figures

Appendix A

Installation Guidelines

- Upload all files in the `sources` folder to the server (the `build` folder is not necessary and is only used for development).
- Make sure the `public` folder is the web root, either with the server configuration or the included `.htaccess` file.
- Rename the `pdo_config.php.dist` file to `pdo_config.php` and update the database credentials in this file.
- To update the database run the `migrate.sql` file. The existing publications with the `thesis` type have to be changed to one of the new thesis type.
- To run the unit tests execute the following command from the `publications` folder: `vendor/bin/phpunit test`.

Optional:

Update third party libraries Dependencies are managed by Composer. Get the `composer.phar` file with `curl -sS https://getcomposer.org/installer | php` or update the existing PHP archive by executing `php composer.phar self-update`. Then run the update command of Composer: `php composer.phar update`.

Run Phing build file The included build file can be run with Phing (`http://www.phing.info/`). The default target will run `phplint` to check the syntax, `phpunit` to execute the unit tests with code coverage, `phpcs` to find code standard violations, `phpmd` to find messes and `phpcpd` to find copied code.

Appendix B

Contents of the CD

- Report in source files, including figures
- Report as a PDF
- Excel file with the schema of the new implementation and the mapping to BibTeX and Merlin
- Source of the new implementation, including third party libraries for production and PHPUnit