Improving Web Sites by Automatic Source Code Analysis and Modifications

Krzysztof Ciebiera and Piotr Sankowski

Warsaw University, Warszawa 02-097 ul. Banacha 2, Poland,
{ciebiera,sank}@mimuw.edu.pl
http://www.mimuw.edu.pl

1 Introduction

We propose an automatic tool for analysis and transformation of PHP programs which may be used in many different situations including:

- layer (presentation, logic, database) separation,
- authorization verification,
- security audits,
- performance improvements.

Our tool is based on system of rules which describe possible transformations and calculations, but as opposed to most of existing refactoring tools we use very intuitive method of applying rules based on program flow. In our opinion this approach is more convenient to learn and to use by ordinary developers.

2 System Overview

System works in following steps:

1. A set of PHP files is parsed and the code trees are build.
2. Parser reads code transformation rules from configuration files.
3. Rules are applied to code tree in one or many passes.
4. Resulting HTML is normalized and searched for common parts.
5. Finals results are written to files.

3 Sample Applications

3.1 Layer Separation

In most of Web programs it is possible to separate at least three layers:

- presentation,
- logic,
- database operations.

N. Koch, P. Fraternali, and M. Wirsing (Eds.): ICWE 2004, LNCS 3140, pp. 579–580, 2004.

Transformation is performed in two phases. First we find all functions which are responsible for presentation by checking if they can call `echo` or `print` directly or by calling other function. Those functions are modified to return generated output together with their original results. Such functions have attribute `Function Prints` set. In second phase we only go through main program and split it into appropriate layers. There are different set of rules for statements generating output and not generating it.

An example of simple `if` statement transformation is shown below:

```
singlecontextrule(1
| if ($1) {$*2 exists{tor{name{'Echo'},
  name{'Print'},tisset{'Function Prints'}
  }}}
|template "{if (\$\$3)}", $*2 ,
  template "{/if}"
| $$3 = $1; if ($$3) {$*2}
);
singlecontextrule(9
| if ($1) {$*2}
|
| if ($1) {$*2}
);
```

The first rule fits only those `if` statements which call either `echo` or `print` or any function which has `Function Prints` attribute set. After fitting the rule two `if` statements are outputted, one goes into PHP code and another into Smarty template. Also new variable `$3` is created which will be evaluated only once and then its value will be used in both PHP code and Smarty template. The second rule as it has higher priority will be executed otherwise.