How to interconnect operational and behavioral views of web applications*

Ruben Fonseca, Daniela da Cruz and Pedro Rangel Henriques
University of Minho
Department of Computer Science
Campus de Gualtar, 4715-057, Braga, Portugal
{rubenfonseca,danieladacruz,prh}@di.uminho.pt

Maria João Varanda Pereira
Polytechnic Institute of Bragança
Campus de Sta. Apolónia
Apartado 134 - 5301-857, Bragança, Portugal
mjoao@ipb.pt

Abstract

In the context of our research project, we are looking for program comprehension tools that are able to interconnect operational and behavioral views, aiming at aiding the programmer to relate problem and program domains to reach a full understanding of software systems. In particular we have been studying the adaptation of that principle to web applications. In this context, we had designed and implemented a tool called WebAppViewer. *In this paper, we emphasize the development of the module (BORS) that is responsible for providing interconnection functionalities and we propose a tool demonstration.*
A dedicated web server is included in the system to allow the execution of the piece of code selected by the user. This feature is used to relate the source text (html, php, etc) executed by the server, with the web page received by the client. Code instrumentation is used to collect dynamic information.

1. Introduction

PCVIA[1], Program Comprehension by Visual Inspection and Animation, is a research project looking for techniques and tools to help the software engineer in the analysis and comprehension of (traditional or web-oriented) computer applications in order to maintain, reuse, and re-engineer software systems.

*This work was developed in the context of PCVIA, a Portuguese research project sponsored by FCT under the contract nº POSI/EIA/57662/2004.

[1]Homepage at (`http://wiki.di.uminho.pt/twiki/bin/view/Research/PCVIA`)

To build up a Program Comprehension (PC) environment we need tools to cope with a complete set of programming units (the application), and their purpose is to extract and display static or dynamic data about the application. This kind of information will help the analyst to understand the architecture of the application, the different kind of files involved in the system, the structure of each program and the application behavior. The basic ideas and approaches supporting the development of such an environment can be found in [14, 15, 13, 4].

WebAppViewer was developed in the context of PCVIA project but it is specifically devoted to the comprehension of Web Applications. Web applications have their own characteristics and therefore we need a specialized tool to be able to comprehend them.

Since the internet is not anymore just a set of static pages with some links to navigate between them, it is possible to find complex web applications that deal with dynamic information, services and access to databases. This kind of applications usually have a large set of different kinds of files and we have to deal with different languages and formats. So, we use the following definition: a web application is a set of programs that implements an Information System using a client/server model, a http protocol and an interface in html to be read by a browser.

Many IDE's (Eclipse, Visual Studio, NetBeans, etc) provide functionalities — like syntax highlighting, diagrams relating classes and files and debugging — aiming at aiding the software development process similar to those that are necessary for PC tools. However, this is not our goal. The main idea is to understand a complete software system (multi-technology or multi-language) after its deployment.

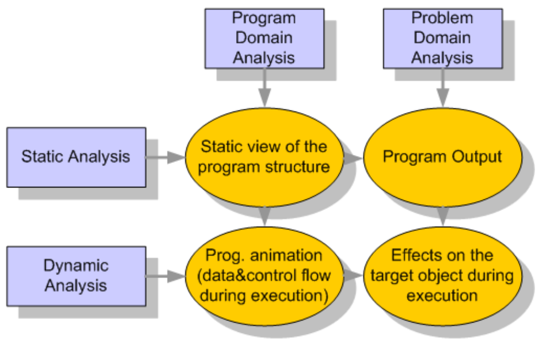
There are some tools that can be really used for web application understanding. Almost them show structural information and just a few ones explain the behavior of the application. Some examples are: PBS[2] [5] and WARE [10, 9]. However, to our knowledge no one of the existing tools found during our recent search satisfy completely all the requirements that we setup for an effective web-application comprehension tool—this statement was precisely the motivation for the development of WebAppViewer. The approach followed by WebAppViewer also allows to extract static and dynamic information, but it is also possible to interconnect the operational and behavioral views in order to relate the program domain with the problem domain. This strategy, called BORS, was born in PCVIA project; it is based on the cognitive models theory of Brooks [3] and it was applied in all PCVIA tools. So, the main focus of this paper is to present WebAppViewer and to discuss how BORS was applied in the context of web applications understanding.

This paper is organized as follows: Section 2 presents BORS in detail; in Section 3 will be emphasized how *WebAppViewer* interconnects the operational with the behavioral view, discussing the static and dynamic analysis process (being illustrated by some of the screens/views produced by the tool); and, in Section 4, we conclude the paper remarking some future work.

2. Architecture of BORS

BORS (Behavioral-Operational Relation Strategy) [2] is a strategy to interconnect the program operation with the problem solving. This task is carried out in two steps: i) showing information about code execution and ii) describing the effects produced by the execution of that code.

So, in order to understand a web application we must carry out static and dynamic analysis and match the outcomes of one to the outcomes of the other, as can be seen in Figure 1.



Program Domain Analysis
Problem Domain Analysis
Static Analysis
Static view of the program structure
Program Output
Dynamic Analysis
Prog. animation (data&control flow during execution)
Effects on the target object during execution

Figure 1. Architecture of BORS

[2]Homepage at `http://www.swag.uwaterloo.ca/pbs/`.

We argue that such analysis and views are relevant. However, the program comprehension is achieved when the user maps the operational view onto the behavioral effects on the target object (notice that we call *target object* to the set of effects produced by the program under analysis during execution).

The program domain includes the knowledge about the programming language and the technologies involved in the code implementation. A static analysis, at this stage, will identify the files, file types, code structure and the relation between files. A dynamic analysis will describe the control and data flow during execution time.

Using a static analysis at the problem domain we can verify the results of the application and check if it really solves the initial problem. A dynamic analysis, at this stage, can be seen as a behavioral analysis. We can verify the application behavior at runtime. It is necessary to map each functionality of the application onto the related code file; BORS is one of the possible approaches to implement this map, that we consider crucial to attain a complete understanding of the program.

3. WebAppViewer

Oliveira et al proposed in [11] the characteristics of a effective tool to help in comprehending a WA. Based on that work, we decided to implement the *WebAppViewer*. In this section, we describe the *WebAppViewer*'s main features and their implementation.

3.1. Static Analysis

The static analysis is accomplished by applying compiler techniques. Compilers are often responsible for the processing, analysis and translation of a set of sentences from a specific programming language. In our case, we are particularly interested in the analysis phase, where the grammatical structure of a program is recognized. In this phase, the compiler analyzes the symbols, the structure and the semantics of each sentence.

In fact, in *WebAppViewer*, we extensively use parsers to gather these results and store them in the data repository. We are particularly interested in:

File type: We can use a parser to detect the type of each text file in the WA. If we search for some *magic tokens* on the text file, we can infer its type.

File contents: We can use a parser — more precisely the underlying lexical analyzer — to highlight the syntax of the file contents. *Syntax highlighting* is well known for aiding the human comprehension of a source text.

Dependencies: During the parsing, we are interested in inferring the dependencies of WA's files. So, everytime a dependency is detected, it is stored into the data repository.

Structure: We can divide each sentence of the source file into some blocks (corresponding to the language structure), more or less detailed. For instance, in a programming language file, we are interested in identifying conditional blocks or function declaration blocks. The parser should store in the data repository the information that will help latter the user to have a better overview (a generic understanding) of a particular WA file.

3.2. Dynamic Analysis

In order to achieve BORS, we should be able to discover exactly which parts of the WA contributes to answer a user request. At the same time, we should be able to associate the parts of the WA executed with the generated output. We argue that this strategy will contribute to understand how a specific part of a WA works.

To achieve this objective, *WebAppViewer* uses well known code annotation and code instrumentation techniques [2]. By introducing inspector functions in the WA's source code, we can collect behavior data in runtime.

So, when the user wants to use *WebAppViewer* to do dynamic analysis, our tool rewrites all the WA programming language files, introducing these inspectors in strategic places. Then, it starts a built-in web server and notifies the user that he can interact with the WA on his regular browser on a specified address.

For each new request the user makes to this web server, we execute the programming files necessary to fulfill the request, gather the extra information produced by the code inspectors, and present different views of this extra data that interconnect the inputs, WA behavior and outputs produced.

The user is free to continue navigating the WA in the usual way, trying different sets of inputs and comparing the results of the dynamic analysis in *WebAppViewer*.

3.3. Implementation

Besides the Java language, the following specific technologies were used to implement *WebAppViewer*:

- Prefuse Toolkit [6]: for graph drawing;
- Apache Batik [8] and Mozilla Rhino[3]: for block diagrams visualization;
- Colorer Library[4]: for syntax highlighting;
- Eclipse SWT [7]: for Graphical User Interface implementation;
- Mozilla Gecko [1]: for the browser.

Moreover, ANTLR [12] was the compiler generator chosen for the parser construction.

3.4. WebAppViewer Visualizations

If the user can not visualize the knowledge extracted, the previous analysis are not useful. To achieve this objective, *WebAppViewer* uses modern visualization techniques to present different views about the WA in study, including:

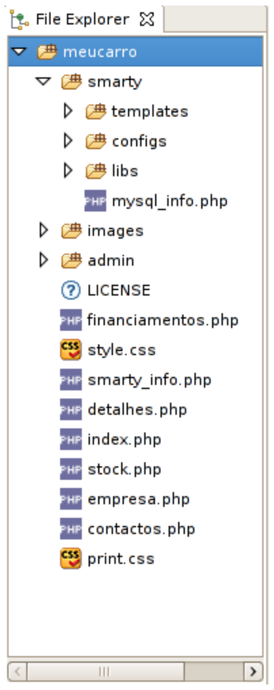


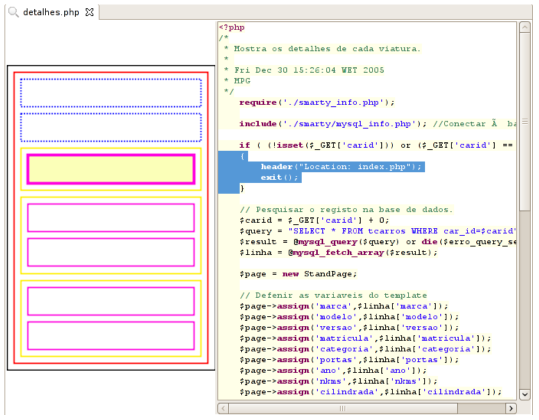
Figure 2. View of the WA's file hierarchy

File hierarchy : We use trees to represent hierarchical information through the entire *WebAppViewer*, as shown in Figure 2. Specific icons are used all over the tree, in the left hand side of each filename, denoting the file type.

File type information : For those types supported by *WebAppViewer*, the tool has a browser widget that can be used to display detailed information about the files of that type (the information shown is extracted from Wikipedia or a similar knowledge repository).

File content : To improve the view that shows the content of a particular WA's file, we use syntax highlighting techniques. Representing the different symbols

[3]Homepage at `http://www.mozilla.org/rhino/`
[4]Homepage at `http://colorer.sourceforge.net/`



```
<?php
/*
 * Mostra os detalhes de cada viatura.
 *
 * Fri Dec 30 15:26:04 WET 2005
 * MPG
 */
    require('./smarty_info.php');

    include('./smarty/mysql_info.php'); //Conectar Ã  ba

    if ( (!isset($_GET['carid'])) or ($_GET['carid'] ==
    {
        header("Location: index.php");
        exit();
    }

    // Pesquisar o registo na base de dados.
    $carid = $_GET['carid'] + 0;
    $query = "SELECT * FROM tcarros WHERE car_id=$carid'
    $result = @mysql_query($query) or die($erro_query_se
    $linha = @mysql_fetch_array($result);

    $page = new StandPage;

    // Defenir as variaveis do template
    $page->assign('marca',$linha['marca']);
    $page->assign('modelo',$linha['modelo']);
    $page->assign('versao',$linha['versao']);
    $page->assign('matricula',$linha['matricula']);
    $page->assign('categoria',$linha['categoria']);
    $page->assign('portas',$linha['portas']);
    $page->assign('ano',$linha['ano']);
    $page->assign('nkms',$linha['nkms']);
    $page->assign('cilindrada',$linha['cilindrada']);
```

Figure 3. View of a WA's programming language file and its Block Structure

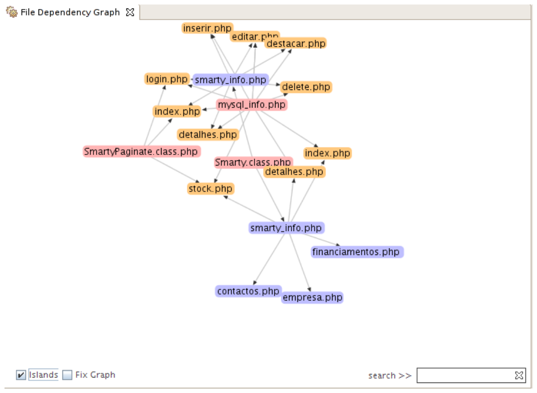


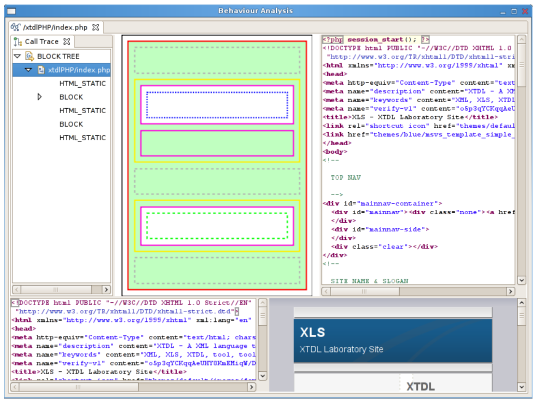
Figure 4. View of the WA's file dependencies

and patterns of a programming language file using different colors is a common way of making easier the code comprehension. *WebAppViewer* has support for dozens of file types used in the current WA.

Programming language files : Figure 3 shows a structured view of a WA's code file. We use SVG to dynamically generate and display a block diagram that represents the different blocks of a particular programming language file. We use different colors and shapes to represent different types of block (e.g., we represent the function declaration and conditional blocks using different colors). The diagrams are fully interactive and the user can pan, zoom and click on a block, highlighting the correspondent piece of code in the source file.

File dependencies : Figure 4 shows a view of the dependencies of a specific WA file. *WebAppViewer* uses graph technology to represent them. The graphs are fully animated and interactive, with edges repelling each other and arcs acting as springs. Moreover, the user can always stop the force mechanics and do incremental search for a particular edge in the graph.

To carry out BORS, we use all these techniques in *WebAppViewer*. For instance, Figure 5 illustrates one of the views produced by *WebAppViewer*, interconnecting the outcomes of dynamic analysis with the reaction of the program to a given specific request.



```
<?php session_start(); ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-stri
<html xmlns="http://www.w3.org/1999/xhtml" xm
<head>
<meta http-equiv="Content-Type" content="text
<meta name="description" content="XTDL - A XM
<meta name="keywords" content="XML, XLS, XTDL
<meta name="verify-v1" content="o5p3qYCKqqAeU
<title>XLS - XTDL Laboratory Site</title>
<link rel="shortcut icon" href="themes/defaul
<link href="themes/blue/msvs_template_simple_
</head>
<body>
<!--

  TOP NAV

  -->
<div id="mainnav-container">
  <div id="mainnav"><div class="none"><a href
  </div>
  <div id="mainnav-side">
  </div>
  <div class="clear"></div>
</div>
<!--

  SITE NAME & SLOGAN
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
 "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
<head>
<meta http-equiv="Content-Type" content="text/html; chars
<meta name="description" content="XTDL - A XML language t
<meta name="keywords" content="XML, XLS, XTDL, tool, tool
<meta name="verify-v1" content="o5p3qYCKqqAeUHY8KmEMiqW/D
<title>XLS - XTDL Laboratory Site</title>
```

Figure 5. ***WebAppViewer*** **dynamic analysis visualization**

On this view, we use all the animated graphs to represent the information gathered by the analyzer:

- The Flow of Execution, in the form of a tree, appears on the top left corner.
- The Block Diagram of the file selected on the tree, with the effectively executed blocks marked with a green background, appears on the middle top.
- The Content of the selected file appears, with the syntax highlighted, on the top right corner.
- The exact HTML produced for the block selected, appears on the bottom left corner.
- The Visualization of this HTML fragment appears, as shown by any browser, on bottom right corner.

We argue that these techniques effectively contribute to a better WA comprehension by exploring the advantages of BORS technique.

4. Conclusion

In this paper, we reinforce the idea of systematically construct a bridge between *operational domain* and *behavior domain*, allowing to map internal/computational operations into external effects, as the key for a successful program comprehension activity. This idea appeared in the context of classic program comprehension tools; here we extend the idea to cope with web application, introducing a new system, called WebAppViewer (a PCVIA project deliverable) specially devoted to web application understanding. As it was told, WebAppViewer provides a set of distinct views and it keeps alive the interconnections between them. In particular, it is possible to have a look into the code that is executed for each action performed in the web page – this functionality realizes the BORS idea exposed above.

To test the tool performance, a real case study was worked out. WebAppViewer was applied to *XTS, XTDL Laboratory Site* (see `http://epl.di.uminho.pt/~gepl/XTDL/`), a web front-end that gives access to the set of tools available to process XTDL, the XML Tool Definition Language. The *XTS* system is made up from 50 files of different types (PHP, HTML, inc, CSS, XSL, GIF, JPEG, TeX) and the PHP files have a size around 7Kb. The static/dynamic analysis was executed successfully, and all the views were available as foreseen; more than that, we could actually test the real impact of observing simultaneously the PHP code under execution and the HTML page shown by the client browser.

As future work, we will follow in two directions: on one hand, we will adapt the system in order to cope with other programming languages, namely we want to support ASP.Net related languages; and, on the other hand, we will assess its effectiveness. For that purpose, it is our intention to follow the traditional application assessment policies to measure the acceptance and usability; we will prepare a questionnaire, as complete as possible, and distribute it to a set of programmers and software analysts using our tool and other tools, in laboratory and real environments, in order to understand the time and effort spent to comprehend the applications selected for testing, as well as the understanding level reached.

References

[1] The design and implementation of the Gecko NFS Web proxy. *Softw. Pract. Exper.*, 31(7):637–665, 2001.

[2] M. Berón, P. R. Henriques, M. J. Varanda, and R. Uzal. Program inspection to inerconnect behavioral and operational view for program comprehension. In *York Doctoral Symposium, 2007*. University of York, UK, Oct 2007.

[3] R. Brooks. Using a behavioral theory of program comprehension in software engineering. In *ICSE '78: Proceedings of the 3rd international conference on Software engineering*, pages 196–201, Piscataway, NJ, USA, 1978. IEEE Press.

[4] T. Eisenbarth, R. Koschke, and D. Simon. Aiding program comprehension by static and dynamic feature analysis. In *IEEE International Conference on Software Maintenance*, Florence, Italy, November 2001.

[5] P. Finnigan, R. C. Holt, I. Kallas, S. Kerr, K. Kontogiannis, H. A. Muller, J. Mylopoulos, S. G. Perelgut, M. Stanley, and K. Wong. The software bookshelf. pages 295–339, 2002.

[6] J. Heer, S. K. Card, and J. A. Landay. Prefuse: a toolkit for interactive information visualization. In *CHI '05: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 421–430, New York, NY, USA, 2005. ACM.

[7] S. Holzner. *Eclipse: A Java Developer's Guide*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[8] A. Kolesnikov. *Java Drawing with Apache Batik: A Tutorial*. BrainySoftware.com, 2007.

[9] G. A. D. Lucca, A. R. Fasolino, F. Pace, P. Tramontana, and U. D. Carlini. WARE: a Tool for the Reverse Engineering of Web Applications. In *CSMR'02: Proceedings of the 6th European Conference on Software Maintenance and Reengineering*. IEEE Computer Society, 2002.

[10] G. A. D. Lucca, A. R. Fasolino, and P. Tramontana. Reverse engineering web applications: the WARE approach. *Journal Software Maintenance Evolution*, 16(1-2):71–101, 2004.

[11] E. Oliveira, M. J. Varanda, and P. R. Henriques. Compreensão de aplicações web: O processo e as ferramentas. In *CAPSI05 - Conferência da Associação Portuguesa de Sistemas de Informação, ESTiG Bragança*, October 2005.

[12] T. J. Parr and R. W. Quong. ANTLR: a predicated-LL(k) parser generator. *Softw. Pract. Exper.*, 25(7):789–810, 1995.

[13] J. Sajaniemi. Program comprehension through multiple simultaneous views: A session with VinEd. In *IEEE journal*. IEEE, 2000.

[14] M.-A. Storey. Theories, methods and tools in program comprehension: Past, present and future. In *13th International Workshop on Program Comprehension (IPWC'05)*, 2005.

[15] A. Walenstein. Theory-based analysis of cognitive support in software comprehension tools. In *10th International Workshop on Program Comprehension (IWPC'02)*, Paris, France, June 2002.