# Integrating Static and Dynamic Analysis to improve the Comprehension of Existing Web Applications

Giuseppe A. Di Lucca and Massimiliano Di Penta

dilucca@unisannio.it, dipenta@unisannio.it

RCOST - Research Centre on Software Technology, University of Sannio
Palazzo ex Poste, via Traiano, 82100 Benevento, Italy

## Abstract

*Web Applications are today becoming more and more interactive and dynamic: the pages rendered in a browser on the client side can be dynamically built by server pages according to the user inputs or requests. Moreover, even pieces of code (e.g., client-side scripts) can be dynamically generated.*

*The comprehension of such applications, needed to effectively perform maintenance or testing, may be very hard. Indeed, for highly dynamic Web Applications, static analysis is likely to give only an imprecise and approximate picture, thus, also dynamic analysis is required to gain a proper understanding of complex and dynamic application behavior.*

*This paper presents the integration and the enhancement of two existing Web Application reverse engineering tools: one, WARE, performing static analysis, the other, WANDA, dynamic analysis. In particular, the paper shows how the integrated toolkit can be used to identify equivalence classes from groups of dynamically built client pages, with the aim of improving comprehension. To validate the proposed approach, results from a case study are presented and discussed.*

## 1. Introduction

The level of adoption of Web Applications (WAs) has changed a lot in the last 10 years. From very simple, static web pages, WAs now constitute a relevant part of many business-critical software systems. This, however, has caused radical changes in the technologies used to develop WAs. While old web sites were composed of just HTML pages, today's WAs are composed of server pages that dynamically generate the HTML, client-side scripts and any other kind of content accessible from the browser.

The dynamicity constitutes a powerful mechanism to enable the creation of complex applications with a Web interface, however it poses serious challenges in the comprehension, maintenance and testing of WAs. In fact:

- comprehending a WA by just analyzing the generated pages from the client's side does not suffice, since it merely provides a black-box view of the WA. Details such as the WA architecture, database access, use of web services and external components are not visible;
- on the other hand, a pure static analysis of the server-side pages would not suffice to deal with the WA dynamicity. For example, a server page may react to different inputs by generating two completely different client pages (e.g., a page containing a form and an error page), or there may exist a data flow dependency between two pages, due to HTTP or session variables. However, the list of passed variables or session variables is dynamically generated, thus, again, static analysis is not enough.

Furthermore, as it happens for traditional applications, dynamic information is necessary for many reverse engineering tasks, such as feature extraction [11], recovering of sequence diagrams [3][17] and of design patterns [10].

This indicates the need for properly complementing WA static analysis with dynamic analysis. While static analysis is not able to capture the dynamicity of the WA, a pure dynamic analysis may fail to detect features/components that are not exercised by the execution of instrumented WAs.

To this aim, we have decided to integrate two WA analyzers, namely WARE [9] that is, basically, a static analyzer, and WANDA [1], that extracts facts from execution traces of an instrumented WA. Both tools have been successfully adopted in the past to perform several reverse engineering tasks over WAs. This paper describes how the two tools have been integrated to identify group of equivalent Built Client Pages (BCPs). As it will be clearer later, the latter task can be exploited for both program comprehension [8] and for testing purposes.

The paper is organized as follows. After a discussion of the related literature, Section 3 describes the WARE and WANDA architecture, also summarizing the features provided by the two tools; then, it is described how the have been integrated. Section 4 describes the approach adopted to identify BCPs, while Section 5 presents a case study that validates the proposed approach. Finally, Section 6 concludes.

## 2. Related Work

The recent literature contains many approaches for reverse engineering WAs. Ricca and Tonella developed the *ReWeb* tool to analyze web sites [14][15][16]. In particular, they extended to WAs traditional static flow analyses such as reachability, dominance, and data flow analysis. Ricca and Tonella also proposed to enhance the analyses considering dynamic information [13]. However, while *ReWeb* obtains dynamic information from web server logs, we obtain dynamic information by instrumenting the WAs, in order to capture some other data that is not available from server logs, such as data stored into/read from a database or a file. The adoption of the Conallen notation [4] for WA documentation introduced the need for reverse engineering UML documentation relying on that extension. To this aim, Di Lucca et al. [9] proposed an approach and a tool, named WARE, to recover WA's documentation represented by UML diagrams (see Section 3.1). In particular, Di Lucca et al. [6][7] applied the tool to abstract use case diagrams, sequence diagrams and business object models from WAs. The proposed approach relies on static information, which may not suffice for an effective and complete abstraction of UML diagrams, due to the dynamic nature of some WA components.

Antoniol et al. [1] proposed a tool, named WANDA, for WA dynamic analysis. The tool enables a fine-grained level dynamic analysis of WAs under execution (see Section 3.2). The aim of this paper is to integrate the latter contributions of WARE and WANDA to achieve a unique toolkit able to benefit from the advantages of both tools.

Boldyreff and Kewish proposed a methodology for the reverse engineering of WAs, with the purpose of identifying duplications and improving maintainability [2]. Vanderdonckt et al. proposed a tool, named *Vaquista* [18], for reverse engineering of WA presentation model. Hassan and Holt [12] proposed a tool for WA architecture recovery. However, while their analysis is merely static, we emphasize here the fact that static analysis needs to be complemented with dynamic information. Finally, Di Lucca et al. [8] presented an approach where static and dynamic information were used to extract use case diagrams.

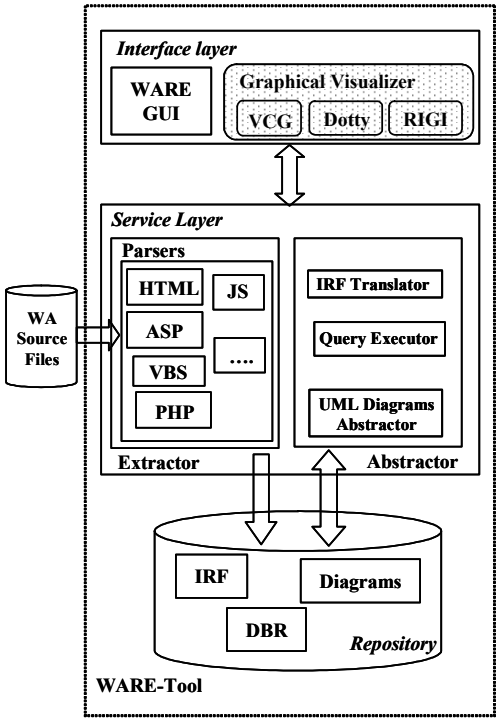



**Figure 1. The WARE architecture [6][9]**

In that case the information obtained from WARE and WANDA was manually integrated, since no unique environment was available. This, once again, indicates the usefulness of a unified environment.

## 3. WARE & WANDA integration

After briefly summarizing the characteristics of the two tools WARE and WANDA, this section describes how the two tools have been integrated.

### 3.1 WARE

The tool WARE (*Web Applications Reverse Engineering*) [6][9] is an integrated environment including several components arranged in the software architecture shown in Figure 1. As the figure illustrates, WARE comprises three layers: the *Interface Layer*, the *Service Layer*, and the *Repository Layer*. The *Interface Layer* implements the user interface providing access to the functions offered by the tool and the visualization of recovered information and documentation both in textual and graphical format. The *Service Layer* implements the tool services, and includes two main components: *Extractors* and *Abstractors*. *Extractors*, by static analysis, directly retrieve relevant information from the source code of an application and store it in intermediate format files, while *Abstractors* are able to abstract further information and documents from the information retrieved by the *Extractors*.

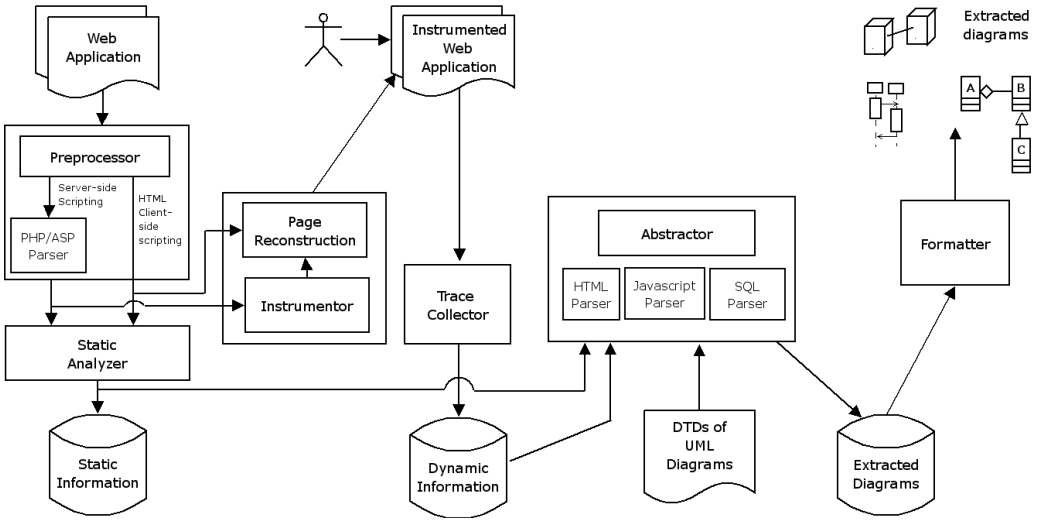



**Figure 2. The WANDA Architecture [1]**

The *Repository* stores facts related to Web applications using an Intermediate Representation Form (IRF) mapped onto a relational database. The main features available to the user include: (1) Web application code parsing, (2) translation of the IRF into the relational database, (3) software comprehension activities, e.g. (3.1) clustering, (3.2) browse of the inventory of the WA entities and source code visualization, and (3.3) reachability analysis. In addition, the user is able to restrict the focus over a subset of WA entities, according to a given partitioning criterion implemented by WARE.

### 3.2 WANDA

WANDA (*Web ApplicatioNs Dynamic Analyzer*) is a tool for the dynamic analysis of WAs. The tool instruments HTML pages and embedded scripting (the current version instruments PHP, although other languages can be added). Then, information collected during WA executions is stored into a database. Dynamic information contained in the database is then used to extract UML diagrams, such as component, deployment, sequence and class diagrams. Component and deployment diagrams highlight the multi-tier architecture of the WA, as well as the interaction with web services, DBMS, files, etc. Sequence diagrams show the interaction between boundary, control and entity classes. Class diagrams represent the WA according to Conallen UML extensions [4]. Since WANDA is mainly a tool for dynamic analysis, the Conallen UML model has been further extended (by defining novel stereotypes and tagged values) to represent the frequency of invocation for each association. This allows the maintainers to visualize the interactions actually taking place during the WA *in-field* usage, for example indicating the number of times a link was followed, the frequency of access to databases and type of operations performed, or the load of a link between two peers.

As shown in Figure 2, the WANDA architecture is organized into three horizontal layers:

1. a *data layer*, responsible for storing the extracted static and dynamic information, as well as the abstracted UML documentation;
2. a *business logic layer*, that contains all the analysis and transformation subsystems; it manipulates the information at various levels of abstraction; and
3. an *interface layer*, to access the original WA, the traces collected by the execution of the instrumented WA; the interface layer is also responsible for the visualization of the results and diagrams.

Viewing the WANDA organization from a tool perspective (i.e., looking at Figure 2 from left to right), we can distinguish tools for i) parsing and instrumentation; ii) information extraction; iii) model abstraction; and iv) presentation of results. Further details about WANDA can be found in the paper [1].

### 3.3 Tools integration and improvement

The WARE and WANDA tools have been integrated and improved to better exploit the results they provide. The integration has been reached by making compatible and consistent the data stored in the databases produced by the two tools. To this aim, tables of the WANDA database were modified to use the same identifiers produced by WARE, as well as some 'correspondence' tables were created as well. In this way, i.e. by modifying the databases, the impact of modifications on the code has been very low.

The main improvements have regarded:

- the static production of a Page Control Flow Graph (PCFG) and the identification of the Linearly Independent Paths (LIPs) in the server pages (see Section 4); and

- the automatic instrumentation of the server pages to record the actual executed paths along user sessions and the LIPs coverage analysis.

The new features of the integrated environment have been implemented as a Java application, and they are accessible by a new user interface added to the pre-existing ones.

The next section describes how the two integrated tools have been used together to identify groups of equivalent built client pages.

## 4. Identifying groups of equivalent built client pages

Built Client Pages (BCP) [4], i.e. the client pages dynamically built by server pages, may make difficult the comprehension and the testing of a WA, when an appropriate documentation does not exist. A full comprehension of a WA implies the knowledge of all the BCPs it can generate. However, this is very difficult to be achieved, especially if the reverse engineering task is limited to static analysis. The BCPs generated from executions of the same server page can differ each other, and the resulting set can be very large. To reduce the comprehension and testing effort, the BCPs can be grouped into equivalence classes, where each equivalence class will include a set of BCPs sharing common features.

A client page, and thus a BCP, can be considered as composed by two main components [7]:

- a *control component*, i.e., the set of items - such as the HTML code and scripts - determining the page layout, business rule processing, and event management; and
- a *data component*, i.e., the set of items - such as text, images, multimedia objects - determining the information to be read/displayed from/to a user.

Pages with the same control component, but different data components, can be considered as equivalent pages, belonging to a same equivalence class. These pages will exhibit the same behavior, thus are likely to have the same semantics.

The set of BCPs generated from a server page has to be analyzed to identify groups of equivalent pages. An equivalence class will be defined for each group and a single equivalent page can be considered to represent each group of equivalent BCPs of a given Server Page.
Thus we can reduce the comprehension effort because we only need to analyze a page for each class. The testing effort is reduced as well: indeed it is sufficient to cover at least a BCP for each equivalent class.
The identification of clusters of equivalent BCPs can be obtained by exploiting the clone detection techniques proposed in the paper [7]. That approach identifies as clones the groups of similar pages according to a Levenshtein distance over structural information. However, this be expensive, since it requires that all the possible kind of BCPs are generated and then clone analysis to be carried out.

In the following a method exploiting the results of static and dynamic analysis of the WARE and WANDA tools is proposed. This method allows reducing the effort for identifying the BCP equivalence classes and permits to know if all the possible BCPs from a server pages have been considered.

The method is made up by the following main steps performed for each server page generating BCPs:

1. represent the page code by a PCFG;
2. identify the LIPs in the PCFG;
3. identify the LIPs along which any BCPs is generated;
4. identify, by analysing the traces of the executed server page paths, the LIPs, or combination of LIPs, covered by the executions generating BCPs; and
5. group in the same equivalence class the BCPs generated by the executed paths covering the same LIPs or combination of LIPs.

The following subsections describe in detail the different steps.

### 4.1 Represent the page code by a Page Control Flow Graph (PCFG)

A server page may be considered as a program that runs on the server. Its code can be modeled by a PCFG (obtained using WARE static analysis capabilities), where a node represents a sequence of statements unconditionally executed, or a predicate of a conditional statement branching the control flow. The statements considered in each node may be script statements, HTML tags including text sentences, and so on. The edges in the PCFG represent the control flow transfer between statements; edges from predicate nodes are labeled with (TRUE, FALSE).

### 4.2 Identify the Linear Independent Paths in the PCFG generating BCPS

In the second and third steps of the proposed approach, the PCFG of each server page is statically analyzed to identify the LIPs it includes. The statements along each LIP are examined to identify the ones that contribute to generate BCPs. The LIPs not including any statement that generates a BCP will not considered in the successive steps. The WARE tool provides to compute the number of LIPs for each server page.

### 4.3 Execution traces analysis

The execution traces of the server pages, registered by WANDA, are analyzed to identify which LIPs have been covered in each execution generating a BCP. Each

BCP is associated to the covered LIP or combination of LIPs whose execution generated it.

### 4.4 Identifying the BCP equivalence classes

The BCPs generated by the execution of the same LIPs, or combination of LIPs, are equivalent BCPs. Indeed they will have the same *control component* or just differ because a subset of the *control component* is present more than once in the page, due to the execution of cycles along the PCFG. Of course, the equivalent BCPs may have a different *data component*. All the BCPs associated to the same LIPs, or LIPs combination, are grouped into an equivalence class. As a simple example, let us consider a server page SP whose PCFG is showed in Figure 3. In this PCFG we can identify the following three LIPs (each LIP is represented by the node sequence it includes):

1. i, 1, 2, 4, 6, 7, f;
2. i, 1, 2, 4, 5, 4, 6, 7, f;
3. i, 1, 2, 3, 7, f;

Let us now suppose that some BCPs are generated by SP along the paths *1* and *2*, while no BCP is generated along path *3*. Let us assume that the following set of Execution Paths (EP) has been recorded for the page SP:

EP(SP) = {

a. (i, 1, 2, 4, 5, 4, 6, 7, f),
b. (i, 1, 2, 4, 6, 7, f)
c. (i, 1, 2, 4, 5, 4, 5, 4, 5, 4, 6, 7, f)
d. (i, 1, 2, 3, 7, f)
e. (i, 1, 2, 4, 6, 7, f)
f. (i, 1, 2, 3, 7, f)
g. (i, 1, 2, 4, 5, 4, 5, 4, 6, 7, f)
h. (i, 1, 2, 3, 7, f)

}

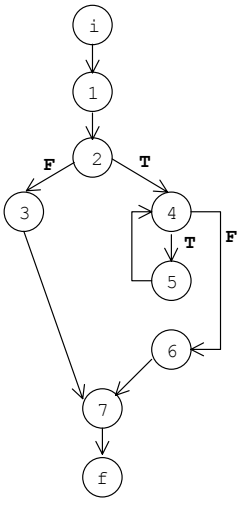



**Figure 3. An example of PCFG**

While defining the BCP equivalence classes we do not consider the traces *d*, *f*, *h*, since they correspond to LIPs where no BCP is generated. We define an equivalence class $EC_1$ = (*b*, *e*) grouping the BCPs generated along the traces *b*, and *e* and an equivalence class $EC_2$ = (a, c, g) grouping the BCPs generated along the traces *a*, *c*, and *g*.

The BCPs in $EC_1$ are pages having the same control component, being them generated by executing the same sequence of statements in SP. The BCPs grouped in $EC_2$ are generated along paths that differs each other because of the different number of repeated executions of nodes *4* and *5*. Nevertheless, we consider that the control component of these BCPs is the same because they, eventually, differ for some repeated elements (such as a different number of rows in a table) without changing the page behavior.

## 5. Case study

To validate the proposed approach, a case study aiming at assessing the effectiveness of the approach to identify classes of equivalent BCPs in existing WAs has been carried out. We considered a small application providing information for tourist itineraries in the Sannio countryside[1], and allowing to make reservations for bus trips. It was analyzed on May 2005. The application server pages are coded using the PHP script language; we counted 26 PHP server pages and 12 HTML static client pages. First, these pages were statically analyzed by the WARE tool. A class diagram, using the Conallen UML extension and the model proposed in [5][9], was built to represent the application pages and the relationships among them. The WARE static analysis revealed that 23 server pages dynamically generate at least a BCPs, but it was not possible to identify haw many different pages can generate a server page. WARE computed the LIPs for each server page too. By counting the number of LIPs and analyzing the code of the corresponding pages, we noted that most of the server pages generate more than 2 BCPs. The PCFGs of the 23 server pages were drawn and the LIPs of each of them identified. Then, the tool WANDA was used to dynamically analyze the application, and to identify the groups of equivalent BCPs from each server page. The application was instrumented by WANDA and 209 execution traces (where each execution trace corresponded to a user session) were recorded, also storing the paths exercised from each server page.

The analysis of the execution traces revealed a large number of BCPs, thus the resulting UML class diagram was not readable and useless for comprehension purposes. The dynamic analysis allowed to identify links between BCPs, and between BCPs and the static pages. The traces corresponding to each server page executions were selected and grouped together. Each group of path traces was analyzed with respect to the LIPs of corresponding executed server pages, to identify which LIPs each path trace covered. The path traces generating

[1] *http://www.pietrelcinaservice.it*

**Table 1. Path Traces of the page carrello.php**

| # | Path Traces |
|---|---|
| 1 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - w94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - end |
| 2 | i - IF76 - IF80 - IF84 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 3 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - end |
| 4 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 – W133 - IF175 - 177 - end |
| 5 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 6 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - end |
| 7 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - W133 - IF175 - 177 - end |
| 8 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 9 | i - IF76 - IF80 - IF84 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 10 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end |
| 11 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 12 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end |
| 13 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - W133 - IF175 - 177 - end |
| 14 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 – 105 – IF115 – IF 119 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 15 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 – 105 – IF115 – IF119 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end |
| 16 | i - IF76 - IF80 - IF84 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 17 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end |
| 18 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end |
| 19 | i - IF76 - IF80 - IF84 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 20 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end |

a BCPs and covering the same LIP, or combination of LIPs, were identified and the BCPs generated along them were associated to a same equivalence class.

We report the analysis made for the server page `carrello.php` (shopping cart). This page, whose PCFG is shown in Figure 4, has 12 LIPs[2] (it is the server page with the highest number of LIPs in the application). 20 executions of this page were considered, and Table 1 shows the path traces recorded by WANDA along such executions. In the table each path is identified by a number, in the first column, while each path is described by the nodes identifiers of the PCFG in Figure 4.

A BCP was generated along each of the executed paths. After this analysis, the executed paths were grouped into seven equivalence classes, according to the LIP they covered, where each class corresponds to a BCP equivalence class. Table 2 reports the identified equivalence classes. In this table the column 'EC Composition' reports the path trace identifiers, (the same of the Table 1), grouped into each equivalence class, while the 'Covered LIP' column indicates which of the LIPs in the PCFG in Figure 4 is covered.

The paths grouped in each class just differ from the number of times some elements (e.g., table rows) appear in the page. Usually that number is determined from user inputs. The pages in Figure 5 (a) and (b) are just an example of equivalent BCPs grouped in the class 2, while the one Figure 5 (c) is a page grouped in the class 4. Note that not all the LIPs were covered by the considered executed paths, thus other BCPs equivalence classes may be defined for the uncovered LIPs.

A manual validation of the BCP Equivalence Classes was carried out. Every equivalence class actually included equivalent BCPs. We noted that some equivalence classes could be merged together because they included groups of pages similar; this suggests us that the method can be improved by better distinguishing between *'similar LIPs'*, i.e. different LIPs that do not produce radical changes in the control component of the equivalent BCPs.

The clustering of BCP into equivalence classes allowed, in program comprehension tasks (that we do not report here, because of the lack of space and since they are out of scope of this paper) to produce UML class diagrams (where equivalence classes are represented) by far better understandable than the original ones (containing all BCPs).

## 6. Conclusions and future work

The need for defining novel approaches allowing a complete comprehension of an existing Web Application (WA) is well known. Static analysis has been already used in a number of approaches proposed to this aim,

[2] We do not report the list of the LIPs for sake of brevity.

while dynamic analysis has been used by a few approaches. In this paper, an approach based on both WA static and dynamic analysis has been proposed to identify

**Table 2. The BCPs Equivalence Classes (EC) for the page carrello .php and the corresponding covered LIPs**

| # | EC Composition | Covered LIP |
|---|---|---|
| 1 | 1-3-6 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - end |
| 2 | 2-9-16-19 | i - IF76 - IF80 - IF84 - W94 – 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 3 | 4-7-13 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - IF123 - 146 - IF175 - 177 - end |
| 4 | 5-8-11 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 5 | 10-12-17-18-20 | i - IF76 - IF80 - IF84 - W94 - IF95 - 96 - W94 - 105 - IF115 - IF119 - 120 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end |
| 6 | 14 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 – IF115 – IF119 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - end |
| 7 | 15 | i - IF76 - IF80 - 81 - IF84 - W94 - IF95 - 96 - W94 - 105 – IF115 – IF119 - IF123 - W133 - IF134 - 135 - W133 - 143 - IF175 - 177 - end |

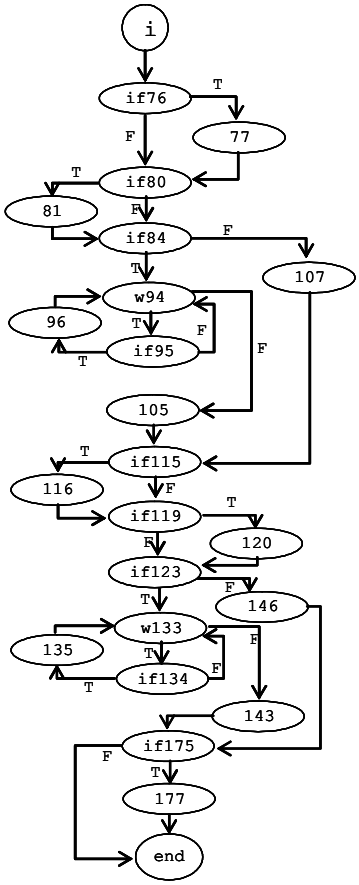
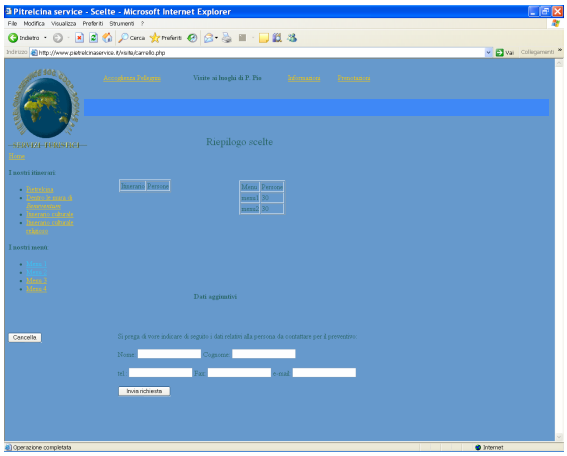
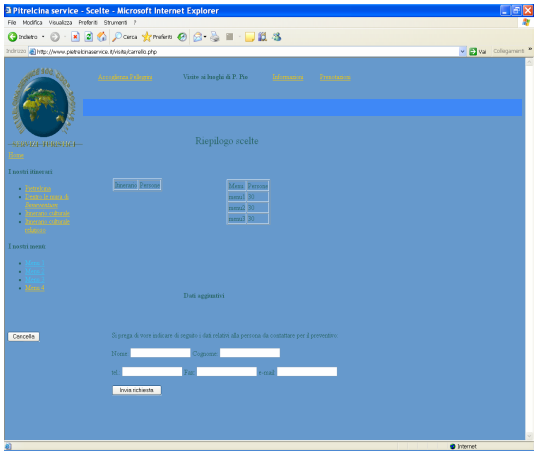


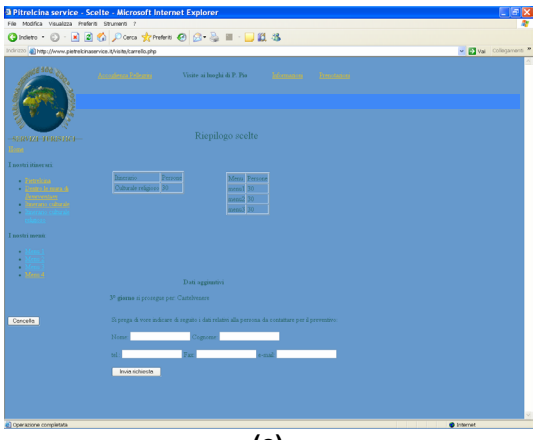

**Figure 4.The PCFG of the server page carrello.php**




**(a)**




**(b)**




**(c)**

**Figure 5. An example of equivalent BCPs in the case study application**

the client pages dynamically generated at run time by server pages and to group together sets of similar built client pages to be considered as a single equivalent page. This approach requires that the WA is first statically analyzed to identify the server pages generating BCPs and the LIPs in such pages.

Then, the WA code is instrumented in order to collect the execution traces corresponding to the WA navigations performed by its users, and the paths actually executed in the server pages. Finally, the execution traces are analyzed to identify the actual BCPs. Equivalent BCPs are identified by grouping together the ones due the execution of the same LIPS in the server pages.

A case study has been carried out to validate the proposed approach. The results showed that dynamic information collected via WA instrumentation allowed the set of BCPs to be correctly identified as well as the groups of equivalent BCPs. The case study also showed that the greater is the number of collected execution traces, the better are the results the approach produces.

Future work will be devoted to identify 'similar LIPs' in the server pages, whose identification will allow reducing the number of BCPs equivalence classes, and thus the comprehension effort.

## 5. Acknowledgments

The authors would thank Gino De Maria for the work done in integrating the WANDA and WARE tools.

## References

[1]. G. Antoniol, M. Di Penta and M. Zazzara "Understanding Web Applications through Dynamic Analysis", in *Proceedings of the 12th International Workshop on Program Comprehension*, IWPC 2004, pp. 120-129.

[2]. C. Boldyreff and R. Kewish, "Reverse engineering to achieve maintainable WWW sites," in *Proceedings of IEEE Working Conference on Reverse Engineering*, (Stuttgart, Germany), pp. 249-258, Oct 2001.

[3]. L. Briand, Y. Labiche, and Y. Miao, "Towards the reverse engineering of UML sequence diagrams," in *Proceedings of IEEE Working Conference on Reverse Engineering*, (Victoria, BC, Canada), pp. 57-66, Nov 2003.

[4]. J. Conallen, "Building Web Applications with UML", Addison-Wesley Publishing Company, Reading, MA, 1999.

[5]. G. A. Di Lucca, M. Di Penta, G. Antoniol, G. Casazza, "An approach for reverse engineering of web-based applications". In *Proceedings of 8th Working Conference on Reverse Engineering* - 2001, IEEE Computer Society Press, Los Alamitos, CA.

[6]. Di Lucca G. A., Fasolino A.R., De Carlini U., Pace F., Tramontana P. "WARE: a tool for the Reverse Engineering of Web Applications". In *Proceedings of 6th European Conference on Software Maintenance and Reengineering* - 2002, IEEE Computer Society Press, Los Alamitos, CA.

[7]. G. A. Di Lucca, M. Di Penta, A. R. Fasolino, "An Approach to Identify Duplicated Web Pages", in *Proceedings of 26th IEEE Annual International Computer Software and Applications Conference (COMPSAC 2002)*, Oxford, England, 2002, IEEE Comp. Soc. Press, Los Alamitos, CA.

[8]. G. A. Di Lucca, M. Di Penta, A. R. Fasolino, P. Tramontana, "Supporting Web Application Evolution by Dynamic Analysis". To appear in *Proceedings of the International Workshop on Principles of Software Evolution*, September 2005, Lisbon, Portugal.

[9]. G.A. Di Lucca, A.R. Fasolino, P. Tramontana, "Reverse Engineering Web Application: the WARE approach", *Journal of Software Maintenance and Evolution: Research and Practice*, Volume 16, Issue 1-2, Date: January - April 2004.

[10]. D. Heuzeroth, T. Holl, G. Högström, and W. Löwe, "Automatic design pattern detection," in *Proceedings of the IEEE International Workshop on Program Comprehension*, (Portland, OR, USA), pp. 94-103, May 2003.

[11]. T. Eisenbarth, R. Koschke, and D. Simon, "Locating features in source code," *IEEE Transactions on Software Engineering*, vol. 29, pp. 210-224, Mar 2003.

[12]. A. Hassan and R. Holt, "Architecture recovery of web applications," in *Proceedings of the International Conference on Software Engineering*, (Orlando, FL, USA), pp. 349-359, May 2002.

[13]. P. Tonella and F. Ricca, "Dynamic model extraction and statistical analysis of web applications," *in Proceedings of International Workshop on Web Site Evolution*, (Montréal, QC, Canada), pp. 43-52, Oct 2002.

[14]. F. Ricca and P. Tonella, "Web site analysis: Structure and evolution," in *Proceedings of IEEE International Conference on Software Maintenance*, (San Jose, CA, USA), pp. 76-85, Oct 2000.

[15]. F. Ricca and P. Tonella, "Analysis and testing of web applications," in *Proceedings of the International Conference on Software Engineering*, (Toronto, ON, Canada), pp. 25-34, IEEE Computer Society Press, May 2001.

[16]. F. Ricca and P. Tonella, "Understanding and restructuring web sites with ReWeb," *IEEE Multimedia*, vol. 8, pp. 40-51, Apr-Jun 2001.

[17]. T. Systä, "On the relationships between static and dynamic models in reverse engineering Java software," in *Proceedings of IEEE Working Conference on Reverse Engineering*, (Atlanta, GA, USA), Oct 1999.

[18]. J. Vanderdonckt, L. Bouillon, and N. Souchon, "Flexible reverse engineering of web pages with Vaquista," in *Proceedings of IEEE Working Conference on Reverse Engineering*, (Stuttgart, Germany), pp. 241-248, Oct 20, 2001.