# Automated Reverse Engineering of UML Sequence Diagrams for Dynamic Web Applications

Manar H. Alalfi James R. Cordy Thomas R. Dean

School of Computing, Queen's University, Kingston, Canada
{alalfi, cordy, dean}@cs.queensu.ca

## Abstract

*This paper presents an approach and tool to automatically instrument dynamic web applications using source transformation technology, and to reverse engineer a UML 2.1 sequence diagram from the execution traces generated by the resulting instrumentation. The result can be directly imported and visualized in a UML toolset such as Rational Software Architect. Our approach dynamically filters traces to reduce redundant information that may complicate program understanding. While our current implementation works on PHP-based applications, the framework is easily extended to other scripting languages in plug-and-play fashion. In addition to supporting web application understanding, our tool is being used to recover traces from dynamic web applications in support of web application security analysis and testing. We demonstrate our method on the analysis of the popular internet bulletin board system PhpBB 2.0.*

## 1 Introduction

Program comprehension, analysis and evolution is often based on reverse engineering of the structure and behavior of software to visual models such as UML diagrams, and much recent research has been focussed on recovering and presenting the structure of programs as UML class diagrams. However, the recovery of dynamic behavior, and particularly interaction behavior, to models such as sequence diagrams presents many challenges that have yet to be addressed.

The problem of recovering execution traces to sequence diagrams for object oriented systems, written in languages such as C++ or Java, has already been extensively studied. Hamou-Lhadj and Lethbridge [11], and Briand et al. [4] provide surveys of tools that have been applied in the domain of object oriented languages that deal with interaction behaviors, and Merdes and Dorsch [17] have presented the major challenges in building a scalable and efficient tool to understand the interaction behavior of such software. These surveys raise four main issues: First, how do methods model the execution traces? Second, how do they solve the execution trace explosion problem? Third, is the method able to represent technical details such as loops and conditions in the sequence diagrams? And fourth, how do the methods represent the final diagram for visualization purposes?

Reverse engineering of sequence diagrams from web applications implemented using scripting languages such as PHP faces all of these problems, and presents a number of additional challenges that are not addressed by these object-oriented analysis methods:

- Identification of the interaction elements. In general, web applications are not built based on object oriented concepts, so it can be difficult to identify the application entities in the source code.
- Identification of loops and conditions. Web applications often have multiple entry points, and exhibit behavior that is difficult to detect until run time. This behavior usually depends on user inputs that can not be inferred by static analysis.
- Recognition of similar execution trace patterns from static or run time information.
- Representation of the complete set of behavioral changes from state to state in sequence diagram terminology. Web applications often have several components that may be affected by a single page execution, such as the database and session and cookie variables.
- Analysis of multilingual documents.

In this work we present an approach and a tool that faces these additional challenges, automatically generating interaction sequence diagrams from dynamic web applications. Our method is not a perfect solution for all of these issues, but is an improvement to the extent that it delivers accurate results and supports the process of web application comprehension, analysis and evolution.

978-0-7695-3671-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICSTW.2009.8

### 1.1 Web Application Testing

PHP2XMI is an essential part of a framework aimed at testing the conformance of dynamic web applications with role-based access control security policies. A role-based access control (RBAC) security model is recovered from the dynamic web application using a combination of static and dynamic analysis techniques. The static analysis is used to recover application entities and the relations between them as a UML-based ER model [1]. The dynamic analysis, implemented using PHP2XMI, recovers the permissions associated with each user role and expresses them as a behavioral model expressed as a UML sequence diagram. Visualizing execution traces as a sequence diagram facilitates the process of understanding the interaction behavior of the system, and helps us deduce the permissions for each user role. The XMI 2.1 textual representation of the sequence model is analyzed and combined with the XMI 2.1 representation of the ER model to construct a UML-based RBAC model, which can be converted into a formal model to be checked for access control vulnerabilities using a standard model checker.

This paper explains how PHP2XMI is used to recover role permissions at the level of page access, and we are currently also evaluating it at the entity level. The behavioral model recovery technique implemented in PHP2XMI can be used to test for other web application security vulnerabilities, such as SQL injection, by tracking SQL sources and their relation to user inputs captured as Http variables. For security analysis we want to preserve all the possible paths the user may follow to reach target pages. Hence, the model behind the execution traces is a complete graph. A filtering process is used to ensure that each observed path is stored just once in the database. While our current version of the tool does not model loops explicitly, our filtering does not prevent the handling of cycles and they can be easily detected by analyzing the database and representing them using the appropriate UML 2.1 meta-model elements.

The rest of this paper is structured as follows. Section 2 presents the details of our approach, and Section 3 presents an example that demonstrates our method on a real system. Section 4 relates our efforts to previous work. Finally, Section 5 outlines our conclusions and plans for future work.

## 2 PHP2XMI

PHP2XMI is a new reverse engineering tool aimed at recovering UML 2.1 sequence diagrams from PHP-based dynamic web applications. The approach used in PHP2XMI involves three steps, as shown in Figure 1:

1. *Parsing and Dynamic Instrumentation*: The core of our method, which automatically inserts probes into the source code to collect dynamic information such as page URLs, http variables, sessions and cookies.

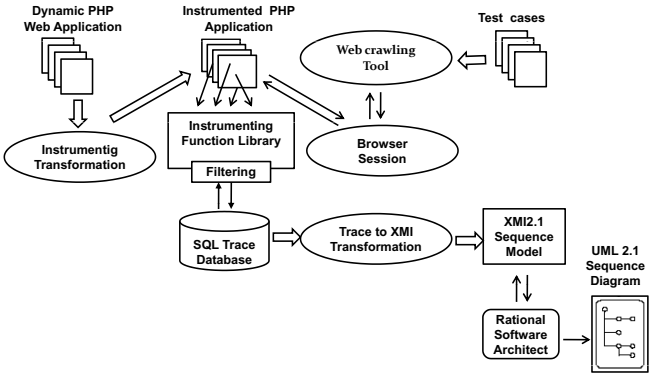


**Figure 1.** PHP2XMI tool Architecture

2. *Filtering and Storing*: During interactive browser sessions, execution traces generated by the probes are filtered to ignore redundant information and stored in an SQL database for further analysis.
3. *Database Analysis and Model Generation*: The execution traces stored in the database are transformed into UML2.1 sequence meta-model elements.

In the following subsections we present the details of each of these main steps.

### 2.1 Parsing and Dynamic Instrumentation

Static analysis alone is not sufficient for architectural recovery of heterogeneous and highly dynamic software such as web applications, and therefore it must be complemented by dynamic analysis [21]. Instrumentation is one of the techniques used to observe and extract dynamic information from systems during execution [13]. Instrumentation does not modify the system structure and behavior. It may add new variables, insert new code, invoke the original program methods, or replace part of the code by an invocation of a new method that substitutes for the omitted code while performing additional tasks related to the instrumentation process. However, the functionality of the original program should not be affected by instrumentation, and the instrumented program must deliver the same results as the original uninstrumented version. The only side effect caused by instrumentation is the additional overhead of recompiling the source code, executing the instrumentation code and generating the execution traces. In our method we use a source transformation technique to add source code instrumentation to dynamic web applications. For this purpose we use TXL [6], a programming language for manipulating and experimenting with programming language notations and features. TXL is a powerful source transformation system that has been used in industrial applications involving millions of lines of source code. The TXL transformation process consists of three parts: a context-free ”base” gram-

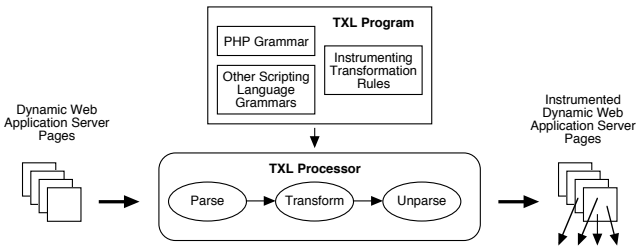


**Figure 2.** TXL transformation technique in PHP2XMI

mar for the language to be manipulated, a set of context-free grammatical "overrides" (extensions or modifications) to the base grammar, and a rooted set of source transformation rules to implement transformation of the extensions to the base language. The TXL processer parses the source program into a parse tree, then recursively applies the set of transformation rules, beginning with a main rule, until there are no remaining matches in the parse tree. The transformation is completed by unparsing the transformed tree to the new target source program.

The source transformation approach brings two benefits to the instrumentation process. First, the process can be adapted in a plug and play fashion to deal with any scripting language as a source for instrumentation. This can be done by writing a set of context-free grammatical overrides to the base grammar (in our case at present PHP versions 3,4,5) to add the grammars of the additional languages, along with additional transformation rules to take into account the code of the newly added scripting languages.

Second, source transformation easily adapts to documents that include a mixture of languages and technologies, usually by applying island grammars [23], for example in the approach used by Synytskyy et al. [22] to handle mixed-language web pages. Island grammars divide the input into interesting input forms, called "islands", and uninteresting sequences of other input items, called "water". In our case, the islands are the PHP script statements that we want to instrument, and the water is the surrounding static HTML code and text. The main benefit of island grammars is that interesting parts can be identified without performing a detailed parse of the entire input [18]. In addition, no preprocessing is needed to unify the source code of the web application as in the approach used by WANDA [3].

The instrumentation process begins with the main rule, shown in Figure 3, by parsing each web application server page into a parse tree based on the context-free grammar definitions in the grammar file `php.grm`.

TXL begins by applying the main rule to this tree, and then recursively applies the transformation rules until the entire set of pages is instrumented. The main transformation rules we used for instrumentation are: `instrumentPage`, which inserts a call to a PHP function that is responsible for tracing page URLs along and

```
include "php.grm"

function main
  replace [program]
        P [program]
   by
        P [instrumentPage]
          [instrumentcookie]
          [instrumentHttpVar]
 end function
```

**Figure 3.** The `main` TXL transformation rule

*The main rule simply matches the entire input PHP server page source document and applies the transformation subrules* `instrumentPage`, `instrumentCookie` *and* `instrumentHttpVar` *globally to it.*

```
rule instrumentHttpVar
    replace [Expr]
        E [Expr]
    construct NewE [Expr]
        E [Conv_func_GET]
          [Conv_func_POST]
          [Conv_func_COOKIE]
          [Conv_func_SESSION]
    where not
        NewE [= E]
    by
        NewE
end rule
```

**Figure 4.** The `instrumentHttpVar` transform. rule

*The* `instrumentHttpVar` *rule finds every PHP expression (*`[Expr]`*) and applies four transformation subrules to recognize and instrument instances of HTTP variables for GET, POST, cookies and sessions respectively.*

their parameters, filtering them and inserting them into an SQL database, `instrumentcookie`, which inserts a call to a PHP function that captures `cookie` information and inserts it into the database, and `instrumentHttpVar` (Figure 4), which inserts a call to a PHP function that captures information about HTTP variables and inserts it into the SQL database. As an example to demonstrate the transformation process, we discuss here the details involved in instrumenting HTTP variables.

The rule `instrumentHttpVar` (Figure 4) begins by searching for HTTP variables in its pattern by finding all instances of the grammatical type *[Expr]* (expression) from the PHP grammar, and applying a set of transformation subrules, each of which matches a specific HTTP variable. Expressions that are not references to HTTP variables are simply left unchanged since no subrule matches them.

For example, the `Conv_func_GET` subrule (Figure 5) matches any expression of the form *$HTTP_GET_VARS [list of parameters]* and replaces it with a call to the instrumenting function *HttpVar_track (Param, $HTTP_GET_VARS [Param], 'GET')* . At run time the *HttpVar_track ()* function inserts into the database the parameter names and values and identifies them as GET parameters, returning the value of the HTTP variable to the caller as originally expected without instrumentation. The transformation is supported by a small library of such

```
rule Conv_func_GET
    replace  [Expr]
        E [ReferenceVariable]
    deconstruct E
        '$HTTP_GET_VARS
            '[ Param [Expr] ']
    by
        HttpVar_track(Param,
          $HTTP_GET_VARS'[Param'],'GET')
end rule
```

**Figure 5.** The `Conv_func_GET` TXL transform. rule

*The* `Conv_func_GET` *rule transforms each* `HTTP_GET_VARS` *expression to a call to the instrumenting function* `HttpVar_track` *which tracks the GET parameters in the database and returns the original result.*

instrumenting functions that interact with the database. The three other subrules referenced in Figure 4 do similar transformations on references to HTTP variables associated with POST, cookies, and sessions.

Figure 2.2 shows the result of transforming the main PHP server page of the PhpBB 2.0 web application. Sections shown in boldface are instrumentation function calls automatically added by our source transformation.

## 2.2 Filtering and Storing

Once the dynamic web application has been instrumented, execution traces are collected as the application is executed in a web browser. The instrumentation function calls inserted by the source transformations dynamically populate a database with the collected trace information. In our security work, we are primarily interested in collecting unique traces based on user roles. Thus at present we are automatically collecting traces and analyzing them one role at a time. Web crawling tools that mimic user interactions with web applications, such as clicking links, filling in forms and pressing buttons [9, 24] are used to automate collecting traces, while the application roles themselves are recovered manually by studying the software documentation. Roles can be identified from the HTTP session variable and by recovering the way the web application classifies users into roles. (Complete automation of this part is currently a work in progress.)

A user in a specific role can visit a web page more than once, following either the same path or different paths. Capturing each visit and storing it in the database leads to a huge amount of redundant information that can complicate the analysis process. To address this issue, we dynamically optimize the recovered traces by filtering such that we store only unique traces. We recognize unique traces as those that lead to the generation of a new client page, or that generate a previously visited client page using a different path.

New and previously visited client pages are recognized by tracking server pages executed due to user visits. Each server page can generate one or more client pages depending on the parameters passed to the page. We consider that a same client page is regenerated if the server page is re-executed without any parameters, or with the same parameters. In such cases we do not insert the new page into the database unless a different path is followed in its generation. Our database is constructed to reject any insertion that violates these conditions. This approach also detects loops in traces, including revisits to pages from themselves.

```
<?php
{
    ob_start ();
    include_once ('sensfunc.php');
}
define ('IN_PHPBB', true);
$phpbb_root_path = './';
include ($phpbb_root_path . 'extension.inc');
include ($phpbb_root_path . 'common.' . $phpEx);
$userdata = session_pagestart ($user_ip, PAGE_INDEX);
init_userprefs($userdata);
$viewcat = (! empty ($HTTP_GET_VARS [POST_CAT_URL])) ?
   HttpVar_track (POST_CAT_URL, $HTTP_GET_VARS [POST_CAT_URL],GET) : -1;
if (isset ($HTTP_GET_VARS ['mark']) || isset ($HTTP_POST_VARS ['mark']))
{
   $mark_read = (isset ($HTTP_POST_VARS ['mark'])) ?
       HttpVar_track ('mark', $HTTP_POST_VARS ['mark'], POST) :
       HttpVar_track ('mark', $HTTP_GET_VARS ['mark'], GET);
}
else
{
    $mark_read = '';
}
if ($mark_read == 'forums')
{
 if ($userdata ['session_logged_in'])
  {
   setcookie($board_config ['cookie_name'] . '_f_all', time (), 0,
       $board_config ['cookie_path'], $board_config['cookie_domain'],
       $board_config['cookie_secure']);
   cookie_track($board_config['cookie_name'].'_f_all', time (), 0,
       $board_config ['cookie_path'], $board_config['cookie_domain'],
       $board_config['cookie_secure']);
  }
}
  . . .

$template -> pparse ('body');
include ($phpbb_root_path . 'includes/page_tail.' . $phpEx);
ob_flush ();
?>
```

**Figure 6.** Result of instrumenting the main `index.php` server page of PhpBB 2.0

*Sections in boldface have been added by our instrumenting transformation.*

## 2.3 Database Analysis and Model Generation

In this phase a sequence diagram is built as a UML 2.1 sequence model based on the execution traces stored in the database. We implement a PHP program to transform the execution traces in the database to the sequence diagram elements shown in Figure 7, in the form specified by the Object Management Group (OMG) [19].

Many web applications, including the example presented in this paper, are not built with object-oriented concepts in mind. Therefore, the interaction elements in our method are the browser session, the generated pages, and the page transitions including their parameters. The instrumentation process generates a record of fine-grained information including such details as http variables, cookies, and sessions. This information is too large to be included in its entirety in the generated sequence diagram, so we have chosen to include only those parameters that are passed in page transitions and shown in the URL address bar.

Execution trace elements, which constitute a database row for a user in a specific role, represent a page ID, a

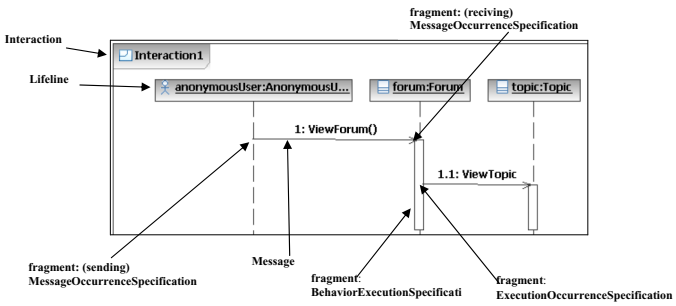


**Figure 7.** UML sequence diagram meta-model elements

page URL, page parameters, and a page access time. User roles and page URLs for a specific page ID are mapped into sequence diagram *lifelines*. The transitions between pages and the set of parameters that accompany these transitions are mapped into sequence diagram *messages*. Page access times in the database are used to determine the order of page transmissions and in the sequence diagram appear as two sets of `MessageOccurrenceSpecification` events, one for sending the message and the other for receiving it. The message receipt event begins the `BehaviorExecutionSpecification` fragment (the rectangle bar in the figure), and the message sending event, `ExecutionOccurrenceSpecification`, ends the same `BehaviorExecutionSpecification` fragment in that lifeline.

## 3 An Example Application

We have applied PHP2XMI to the analysis of the popular internet bulletin board system PhpBB 2.0. Using the method described in the previous section, PHP2XMI was able to automatically recover sequence diagrams from user interaction with this application. The system under test was first uploaded to a test server, where it was automatically instrumented by PHP2XMI and executed in a controlled environment. Test scripts were used to drive the web browser in a manner similar to a user of the application. The test scripts were implemented using *Watir* [24], a library that interfaces Ruby to Microsoft Internet Explorer. Although our framework collects coverage information as part of its instrumentation, in this example we did not attempt to cover all possible execution traces for any specific user role, as that would generate a result much too large to fit in this paper. Instead we collected a representative set of 15 separate execution records, which generated the database shown in Figure 9. From this database PHP2XMI automatically generated a UML sequence diagram model that we imported and visualized using Rational Software Architect (RSA) [7] (Figure 10).

Figure 8 shows a sample of the execution traces collected by PHP2XMI for a browser session with PhpBB 2.0. The sample represents the scenario of an anonymous user visiting a forum main page and exploring one of the active forums, then trying to do a reply on one of its topics, which requires registered user permission. The user then logs in as an Administrator, replies to the post and switches to the Administrator panel.

Each server page is represented by one or more unique `Page IDs` based on whether the page receives parameters when executed. For example, the server page `posting.php` has four entries in the table, each with a different `Page ID`. This is done to reflect the fact that the server page `posting.php`, in this particular example, generates four different client pages based on the parameters it receives. As another example, the server page `login.php` has four entries. Two of them receive different parameters, and thus PHP2XMI gives them different `Page IDs`. On the other hand, the other two entries do not receive any parameters, but are from different paths. PHP2XMI inserts both these visits, but gives them the same `page ID`, in order preserve all the paths that may lead to a specific page. In all cases, filtering insures that the Pages table does not include any duplicate rows with the same combination of `Page Name`, `Page Parameter` and `Prev. Page ID`. Figure 9 shows a view of the join of three tables of the collected database, illustrating the fine-grained information collected by PHP2XMI, including page information, HTTP variables, cookies, and sessions.

In order to map page transitions from rows of Figure 8 to sequence diagram elements of Figure 7, a lifeline is assigned to the interactive browser session and to each server page. For instance, the `Page Name` of the second row in Figure 8 is mapped into an UML 2.1 element of type `uml:Lifeline`. The `covered by` property of this lifeline lists the `xmi:id's` of the set of `MessageOccurrenceSpecification` and `BehaviorExecutionSpecification` events that this lifeline is engaged in, and that to represent the event of sending and receiving the first message between the two lifelines, `index.php` and `viewforum.php`. This message is represented using an UML 2.1 element of type `uml:message`, and its name is the composition of the server page name and the parameter it receives. Finally, UML 2.1 elements of type `uml:class` represent each lifeline and the set of messages it receives as a class with a set of operations.

Figure 10 shows a visualization for a part of the generated model using Rational Software Architect. Once imported into RSA, the diagram can be explored and connected to other UML models to develop a unified understanding of the entire web application.

## 4 Related Work

In CPP2XMI [14], Korshunova et al. describe a reverse engineering tool to extract class, sequence, and activity diagrams from C++ source to XMI 1.1 format. The authors

| Page ID | Page Name | Page parameters | Prev. Page ID | Page Type | Page Acc_TS |
|---|---|---|---|---|---|
| 1 | http:///phpBB2/index.php | | 0 | PHP | 1211134854 |
| 2 | http:///phpBB2/viewforum.php | ?f=1&sid=668ea9c6f9d3530aa85152da6fc3d7c6 | 1 | PHP | 1211134870 |
| 3 | http:///phpBB2/viewtopic.php | ?t=5 | 2 | PHP | 1211134894 |
| 4 | http:///phpBB2/posting.php | ?mode=reply&t=5 | 3 | PHP | 1211134902 |
| 5 | http:///phpBB2/login.php | ?redirect=posting.php&mode=reply&t=5 | 4 | PHP | 1211134903 |
| 6 | http:///phpBB2/login.php | | 5 | PHP | 1211134918 |
| 7 | http:///phpBB2/posting.php | ?mode=reply&t=5&sid=668ea9c6f9d3530aa85152da6fc3d7c6 | 6 | PHP | 1211134918 |
| 8 | http:///phpBB2/posting.php | ?mode=topicreview&t=5 | 6 | PHP | 1211134919 |
| 9 | http:///phpBB2/posting.php | | 8 | PHP | 1211134957 |
| 10 | http:///phpBB2/viewtopic.php | ?p=11 | 9 | PHP | 1211134961 |
| 11 | http:///phpBB2/admin/index.php | ?sid=668ea9c6f9d3530aa85152da6fc3d7c6 | 10 | PHP | 1211134989 |
| 12 | http:///phpBB2/login.php | ?redirect=admin/index.php&admin=1&sid=668ea9c6f9d3530aa85152da6fc3d7c6 | 11 | PHP | 1211134990 |
| 6 | http:///phpBB2/login.php | | 12 | PHP | 1211135018 |
| 13 | http:///phpBB2/admin/index.php | ?admin=1&sid=668ea9c6f9d3530aa85152da6fc3d7c6 | 6 | PHP | 1211135018 |
| 14 | http:///phpBB2/admin/index.php | ?pane=right&sid=668ea9c6f9d3530aa85152da6fc3d7c6 | 13 | PHP | 1211135018 |
| 15 | http:///phpBB2/admin/index.php | ?pane=left&sid=668ea9c6f9d3530aa85152da6fc3d7c6 | 13 | PHP | 1211135018 |

**Figure 8.** Sample of a database view of generated execution traces

| Page ID | Page Name | Page Param. | PageAccess TimeStamp | Prev PageID | HttpVar Name | HttpVar Value | HttpVar Type | HttpVar Time_Stamp | CookiesName | Cookies value | Cookies ExpireTime |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | http:///phpBB2/index.php | | 1211134854 | 0 | NULL | NULL | NULL | NULL | phpbb2mysql_data | a:2:{s:11:"autologinid";s:0:"" ;s:6:"userid";i:.1;} | 1242670855 |
| 2 | http:///phpBB2/viewforum.php | ?f=1&sid=668ea9c6f9d35 30aa85152da6fc3d7c | 61211134870 | 1 | f | 1 | GET | 1211134871 | NULL | NULL | NULL |
| 3 | http:///phpBB2/viewtopic.php | ?t=5 | 1211134894 | 2 | t | 5 | GET | 1211134895 | NULL | NULL | NULL |
| 4 | http:///phpBB2/posting.php | ?mode=reply&t=5 | 1211134902 | 3 | mode | reply | GET | 1211134903 | NULL | NULL | NULL |
| 4 | http:///phpBB2/posting.php | ?mode=reply&t=5 | 1211134902 | 3 | t | 5 | GET | 1211134903 | NULL | NULL | NULL |
| 5 | http:///phpBB2/login.php | ?redirect=posting.php&mo de=reply&t=5 | 1211134903 | 4 | NULL | NULL | NULL | NULL | NULL | NULL | NULL |
| 6 | http:///phpBB2/login.php | | 1211134918 | 5 | username | admin | POST | 1211134918 | phpbb2mysql_data | a:2:{s:11:"autologinid";s:0:"" ;s:6:"userid";i:.1;} | 1242670918 |

**Figure 9.** Fine grained information collected by PHP2XMI

use Columbus/CAN [10] as a fact extractor, which parses the C++ code and generates output in XMI. The authors then analyze the XMI file to extract the information needed for each diagram, using Dot [15] to visualize the output.

Briand et al. [5] propose a method to reverse engineer sequence diagrams from C++ applications, using Perl to implement the automatic instrumentation and Java to transform traces into sequence diagrams. They propose two meta-models, one to represent the recovered traces and the other to represent sequence diagrams, transforming one to the other based on OCL constraints. While they recover the technical details such as conditions and loops, they do not address visualization of the resulting sequence diagrams.

Jiang et al. [12] propose a method to reverse engineer a sequence diagram in UML 2.0 format from the runtime communication between sample applications and an API. Their method works by monitoring API usage in the sample applications, then filtering the generated execution traces and merging them into a state machine. The analysis of the state machine leads to the recognition of common, optional and alternative parts. A combined sequence diagram is then built and illustrated as a UML 2.0 sequence diagram.

Our approach differs from the above methods in its ability to handle applications with multilingual source code documents, such as web applications. Unlike the Korshunova et al. and Briand et al. methods, PHP2XMI automatically generates XMI 2.1 sequence diagram files which can be visualized directly in any UML 2.1 toolset. While both methods use filtration, ours is focussed on minimizing the database to optimize extraction of sequence diagrams, whereas CPP2XMI gathers a much richer initial XMI representation and then filters to extract sequence elements.

Many methods have been proposed to extract behavioral models from web applications for the purpose of testing. Ricca and Tonella [20], for example, construct a UML object model of a web application's pages, frames, forms and the links between them from the dynamic HTML output of the application. While their object model is aimed at generating test cases, our approach uses automated test cases implemented using *Watir* [24] and a coverage metric to recover a model for the purpose of testing security properties of dynamic web applications. A major difference between our recovered behavioral models, the Ricca and Tonella custom models, and almost all other previous work in this field, is that ours are the only recovered behavioral models represented using standard UML2.1 sequence diagrams. This is an important advantage from a practical point of view, since it allows import, analysis and manipulation using standard UML tools such as Rational Software Architect. A detailed analysis of the state of the art in the field of web applications modeling for analysis and testing can be found in our survey[2].

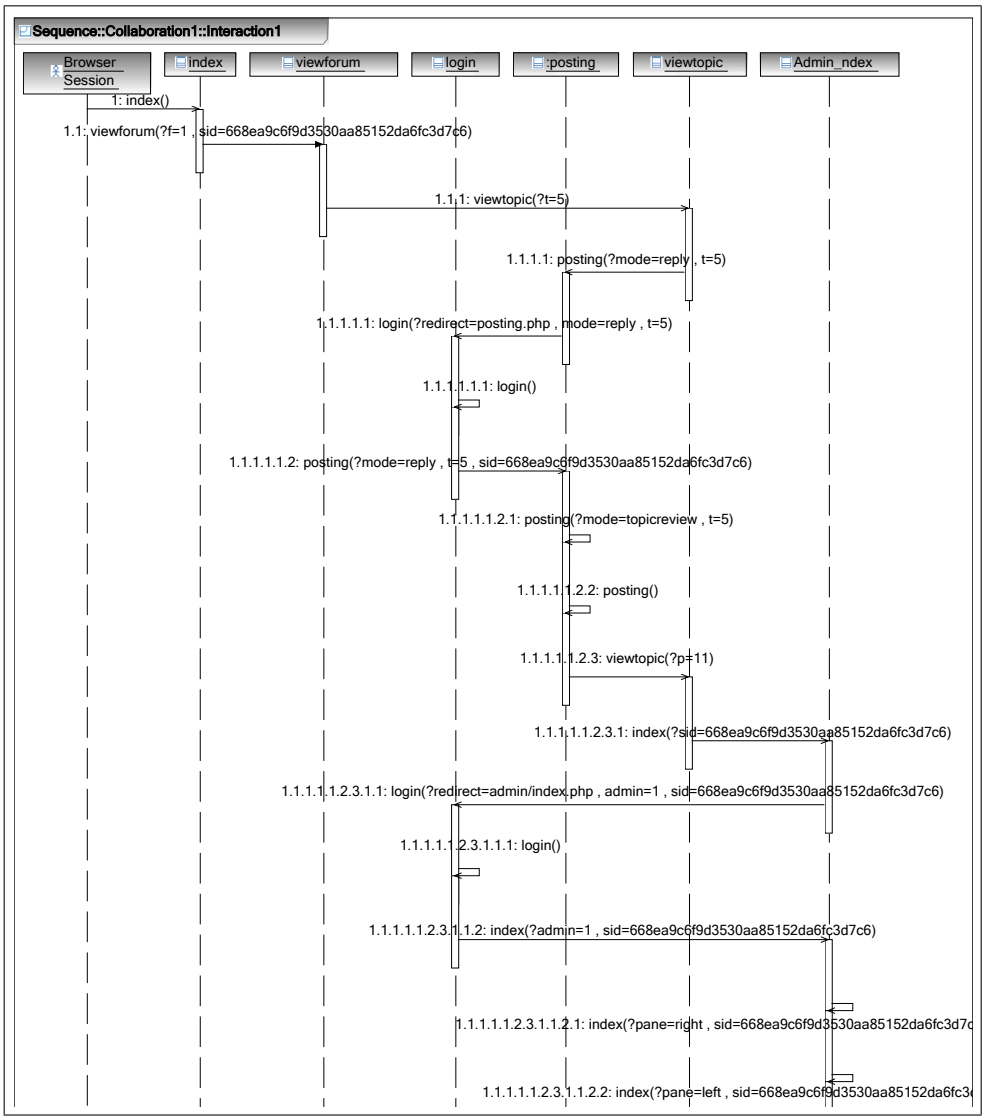


**Figure 10.** An example of a generated sequence diagram

Although it is not designed for web applications testing, the work most similar to our approach is WANDA (Web ApplicatioNs Dynamic Analyzer) [3], which collects execution traces in a similar way. In WANDA execution traces are not filtered, which leads to a huge amount of redundant information stored in the database, and consequently tends to yield cluttered sequence diagrams that are difficult to comprehend. Di Lucca and Di Penta [8] have proposed an approach to filter the execution traces generated by WANDA by using the WARE tool [16] to identify groups of equivalent Built Client Pages (i.e., client pages dynamically built by server pages which share common features). A filtration of the execution traces collected by WANDA is then performed based on page clustering. This method has been evaluated on a PHP web application.

PHP2XMI differs from WANDA in using source transformation technology for the parsing and the instrumentation phases, which aids in eliminating the overhead caused by any preprocessing needed to deal with multilingual documents. The filtration proposed by Di Lucca and Di Penta is based on the static analysis provided by WARE, which may identify the web application pages, but does not preserve all the possible paths that the application may allow its users. For this reason their method is not sufficient for security testing purposes, whereas PHP2XMI's filtering is tailored to the task. WANDA's unfiltered model can also have scalability problems, whereas PHP2XMI's dynamic filtering bounds the number of lifelines and messages in the model to the number of server pages and paths between them. The output format generated by PHP2XMI is in the

XMI 2.1 standard for model interchange between the UML tools, whereas WANDA uses a custom local format.

## 5 Conclusion and Future Work

We have presented an automated approach and practical tool to instrument dynamic web applications using source transformation technology to recover a dynamic behavior model from observed interaction. The tool is able to automatically reverse engineer UML 2.1 sequence diagrams from PHP-based web applications. The result can be imported and visualized in any UML 2.1 toolset. The approach we use filters execution traces directly on insertion into the database, automatically eliminating redundant information that may complicate the understanding process.

Currently the interaction elements in the resulting sequence diagram are the user and the dynamic pages of a browser session, represented as lifelines, and the dynamic transitions between the pages along with their parameters, represented as messages. In future work we plan to raise the sequence diagram to the entity level from the page level. While our current implementation ignores any redundant traces, loops and conditions in web applications can be easily detected and explicitly modeled.

At present we don't try to enumerate all the possible executions for each role, but we intend to do so in future using an instrumentation coverage technique to ensure the accuracy and completeness of the generated sequence diagram. We are also planning the integration of sequence diagrams from different sessions to generate a one complete sequence diagram for the entire web application. Finally, we are working on generalizing PHP2XMI for use in testing other web application vulnerabilities.

## Acknowledgments

This work is supported by the Natural Sciences and Engineering Research Council of Canada.

## References

[1] M. H. Alalfi, J. R. Cordy, and T. R. Dean. SQL2XMI: Reverse Engineering of UML-ER Diagrams from Relational Database Schemas. In *WCRE 2008, Antwerp, Belgium, October 15-18*, pages 187–191.

[2] M. H. Alalfi, J. R. Cordy, and T. R. Dean. Modeling methods for web application verification and testing: State of the art. *Softw. Test., Verif. Reliab.*, 2009 (to appear).

[3] G. Antoniol, M. Di Penta, and M. Zazzara. Understanding Web Applications through Dynamic Analysis. In *IWPC*, pages 120–131, 2004.

[4] L. C. Briand, Y. Labiche, and J. Leduc. Toward the Reverse Engineering of UML Sequence Diagrams for Distributed Java Software. *IEEE Trans. Software Eng.*, 32(9):642–663, 2006.

[5] L. C. Briand, Y. Labiche, and Y. Miao. Towards the Reverse Engineering of UML Sequence Diagrams. In *WCRE*, pages 57–66, 2003.

[6] J. R. Cordy. The TXL source transformation language. *Sci. Comput. Program.*, 61(3):190–210, 2006.

[7] I. Corporation. Rational Software Architect Version 7.0, *http://www-306.ibm.com/software/awdtools/architect/swarchitect/*.

[8] G. A. Di Lucca and M. Di Penta. Integrating Static and Dynamic Analysis to improve the Comprehension of Existing Web Applications. In *WSE*, pages 87–94, 2005.

[9] C. Engineering. Canoo WebTest, *http://webtest.canoo.com*.

[10] FrontEndART Software Ltd. Columbus/CAN 3.5, *http://www.frontendart. com/products_col.php*.

[11] A. Hamou-Lhadj and T. C. Lethbridge. A survey of trace exploration tools and techniques. In *CASCON*, pages 42–55, 2004.

[12] J. Jiang, J. Koskinen, A. Ruokonen, and T. Systä. Constructing Usage Scenarios for API Redocumentation. In *ICPC*, pages 259–264, 2007.

[13] R. Kazman, L. O'Brien, and C. Verhoef. Architecture reconstruction guidelines. Technical Report CMU/SEI-2002-TR-034, Carnegie Mellon University, 2003.

[14] E. Korshunova, M. Petkovic, M. G. J. van den Brand, and M. R. Mousavi. CPP2XMI: Reverse Engineering of UML Class, Sequence, and Activity Diagrams from C++ Source Code. In *WCRE*, pages 297–298, 2006.

[15] E. Koutsofios and S. North. Drawing graphs with dot. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA, September 1991.

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

[17] M. Merdes and D. Dorsch. Experiences with the development of a reverse engineering tool for UML sequence diagrams: a case study in modern Java development. In *PPPJ*, pages 125–134. ACM, 2006.

[18] L. Moonen. Lightweight Impact Analysis using Island Grammars. In *IWPC*, pages 219–228, 2002.

[19] Object Management Group (OMG). OMG Unified Modeling Language (OMG UML), Superstructure, V2.1.2, *http://www.omg.org/docs/formal/07-11-01.pdf*. Technical report, 2007.

[20] F. Ricca and P. Tonella. Analysis and testing of web applications. In *ICSE '01: Proceedings of the 23rd International Conference on Software Engineering*, pages 25–34, 2001.

[21] A. Seesing and A. Orso. InsECTJ: a generic instrumentation framework for collecting dynamic information within Eclipse. In *ETX*, pages 45–49, 2005.

[22] N. Synytskyy, J. R. Cordy, and T. R. Dean. Robust multilingual parsing using island grammars. In *CASCON*, pages 266–278, 2003.

[23] A. van Deursen and T. Kuipers. Building Documentation Generators. In *ICSM*, pages 40–49, 1999.

[24] WatirCraft. WATIR, *http://wtr.rubyforge.org*, accessed 2 March 2009.