Slicing of

Web Applications using Source Code Analysis

Synopsis report submitted in partial fulfilment
of the requirements for the degree of

Bachelor of Technology
in
Computer Science and Engineering

by

Ankit Kumar and Anubhav Panda
111CS0129 and 111CS0044

under the guidance of

Prof. D.P.Mohapatra

Department of Computer Science & Engineering
National Institute of Technology, Rourkela
Rourkela-769008, Odisha, INDIA
March, 2015

Department of Computer Science and Engineering
National Institute of Technology, Rourkela
Rourkela-769008, Odisha, India

March 9, 2015

Certificate

This is to certify that the work in the thesis entitled *Slicing of Web Applications using Source Code Analysis* by Anubhav Panda and Ankit Kumar is a record of an original research work carried out under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering. Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Durga Prasad Mohapatra
Associate Professor,
Dept of Computer Science and Engineering,
NIT Rourkela

Abstract

Program slicing revealed a useful way to limit the search of software defects during debugging and to better understand the decomposition of the application into computations. The web application is very widely used for spreading business through out the world. To meet the desire of the customers, web applications should have more quality and robustness. Slicing, in the field of web application, helps disclosing relevant information and understanding the internal system structure. This in-terns help in debugging, testing and in improving the program comprehensibility.

The system dependence graph is an appropriate data structure for slice computation, in that it explicitly represents all dependencies that have to be taken into account in slice determination. We have extended the system dependence graph to Web-Application Dependence Graph (WADG). We have developed a partial tool for automatic generation of the WADG and computation of slices. In our literature survey, we found that most of the automatic graph generation tools are byte-code based. But, our tool uses the dependency analysis form the source code of the given program. We have presented three case studies by taking open source web programs and applying our techniques and slicing algorithm. We have found that the slices computed is correct and precise, which will be help full for program debugging and testing.

Construction of the system dependence graph for Web applications is complicated by the presence of dynamic code. In fact, a Web application builds the HTML code to be transmitted to the browser at run time. Knowledge of such code is essential for slicing

Contents

1 Introduction

One of the first systematic studies on Web site analysis is, where the evolution of the entities in Web sites is characterized by means of a set of metrics traced over time. Program slicing is a special type of static analysis, consisting of a decomposition technique for the extraction of program statements relevant to a specified computation. Program slicing has many applications such as debugging, code understanding, program testing, reverse engineering, software maintenance, reuse, software safety and metrics. A Web application consists of a set of Web pages displayed to the user and of a set of server side programs (usually scripts), performing some computation and producing output pages to be displayed.

Java Server Pages (JSP) web applications focuses more on presentation logic. They are easier to maintain than a servlet. As opposite to servlets, JSP adds Java code inside HTML rather than adding HTML code inside Java code but can still achieve everything that a servlet can do.

We will present an approach to Web application slicing. A Web application is a special case of client-server system, in which the Web server plays the role of the server,the Web browser plays the role of the client which establishes communication between the client and the server by using some specific protocols (http, https). Due to the process of modularization most of the current applications use the service of web servers or web APIs (Application Programming Interfaces). With the development of internet the quality of web applications has also increased and is continuing to do so. The result of slicing a Web application is still a Web application (often simpler than the initial one), which exhibits the same behavior as the initial Web application with respect to some information of interest. Web application slicing can be used, as with traditional software systems, to assist programmers and Web designers in tedious and error prone tasks such as understanding, debugging, and regression testing [1][2][3].[4]

In this thesis we have proposed our approach to handle the web pages which consists of JSP ,HTML and JavaScript parts. We have processed the 3 parts differently and generated the WADG (Web Application Dependence Graph) in order to explain the flow of web applications as well as different kind of dependence between them. We have considered page call and function call differently. Also, we have applied our slicing algorithm upon the generated WADG with multiple Slicing criteria and analyzed the resulting slices.We are also working on the dynamic slicing of web application. And we have considered different cases of it.

Web applications comprising dynamically generated pages are more difficult to analyze than staticWeb sites. The server script that is executed when a page is requested builds the HTML structure of the resulting page at run time. The tags inserted into the page can, in general, depend on the state of the program and on the input values or the value from the databases. Moreover, even the names of the HTML variables in the generated page, the FORM, ACTIONS and the hyperlinks can be constructed dynamically, and vary from execution to execution. If the WADG is constructed with no regard to the generated code, important nodes and edges are possibly missed. For example, when tags are produced dynamically, the related nodes are not in the WADG built for the server script. When variable names are generated dynamically, the data dependences associated to them are absent in the WADG. When FORM,ACTIONS are inserted dynamically in a page, the related call dependences are not reported in the WADG. The problem of statically constructing a precise WADG for a dynamicWeb application is in general undecidable. This means that there existWeb applications for which it is not possible to build a precise WADG just by analyzing the source code, because the dynamically generated WADG nodes cannot be approximated statically. However, under reasonable assumptions it is possible to devise static analysis techniques that are able to precisely produce the WADG for a subset of all possibleWeb applications, including most of the real world ones. Another completely different approach to the problem of determining the dynamically constructed WADG portions relies on dynamic slicing. When a specific execution of a Web application is considered, with given input values, the problem of approximating the generated HTML code disappears, since this can be obtained by actually running the application. Thus, slices can be computed directly on the generated code, which is known. However, the resulting slices hold only for the input values used in the execution. Some of the excluded portions of the Web application could be relevant to the selected computation for different inputs. This is why we focus on static slicing.We are currently working on static construction of the WADG for Web applications in presence of dynamic code generation. The code produced at run time is extruded from the statements that generate it. However, code extrusion is not sufficient when the generated code is accumulated into a string variable before being printed out. A flow analysis algorithm, called string-cat propagation, is proposed to handle such cases. Combined with code extrusion, it allows treating several programming patterns commonly used withWeb applications, such as string variables, where the HTML code is temporarily stored, and general purpose functions, that are parameterized at invocation in order to generate the desired HTML fragment.

2 Basic concepts

A program slice is a reduced, executable program obtained from a given program by removing statements, so that it replicates part of the behavior of the initial program.[5][6]
Similar concept has been applied on web applications to generate web application slicing. This can be achieved with the help of system dependence graph.

2.1 Web Slicing

A Web application consists of a set of Web pages displayed to the user and of a set of server side programs (usually scripts), performing some computation and producing output pages to be displayed. As a consequence, slicing a Web application should result in a Web application which exhibits the same behavior as the initial Web application with respect to the information of interest. A Web application slice is obtained from a given set of Web pages and scripts by selecting HTML and script statements, so that part of the behavior of the initial Web application is replicated.[7][8]

2.1.1 Slicing Criterion

The criterion for slice computation is an information item of interest displayed in a given Web page, and the resulting slice reproduces the same information item, if the user performs the same navigation actions and provides the same input in the original and in the sliced Web application. Formally, a slicing criterion (n, x) consists of a statement n, displaying the information item of interest, and a variable x, used for the production of such an information item[7].

2.1.2 Types of Slicing

There are many types of program slicing exists. Slicing can be distinguished by the nature of program's execution or by its approach. We are here presenting some basic types of slicing, but for more detailed study some good review papers [9][10], can be referred.

Static Slicing: A static slice includes all the statements that affect variable v for a set of all possible inputs at the point of interest (i.e., at the statement x). Static slices are computed by finding consecutive sets of indirectly relevant statements, according to data and control dependencies.

Dynamic Slicing: A dynamic slice contains all statements that actually affect the value of a variable at a program point for a particular execution of the program rather than all statements that may have affected the value of a variable at a program point for any arbitrary execution of the program.

Forward Slicing: Forward slice consists of all statements and control predicates dependent on the slicing criterion, a statement being "dependent" on the slicing criterion if the values computed at that statement depend on the values computed at the slicing criterion, or if the values computed at the slicing criterion determine the fact if the statement under consideration is executed or not.

Backward Slicing: A backward slice contains statements of a program which has some effect on the slicing criterion. It helps the developer to locate the parts of the program that contains a bug. Backward slice requires tracing dependencies in the backward direction.

2.2 System Dependence Graph

A System Dependence Graph (SDG) is used to model dependence between statements within a procedure including inter-procedural dependence. The final slicing algorithm will be depend over the SDG. The SDG is a graph whose nodes roughly correspond to program statements and whose edges model dependence in the program.

There are various types of dependence which could explain the different relation between two statements of the input program.

2.2.1 Control Dependence

A control dependence holds between two statements if the former defines a scope which directly includes the latter. It is also known as nested dependence. Control dependence for traditional programs connect the predicate tested at a conditional or loop statement to the instructions the execution of which directly depends on the truth value of the predicate. The same holds for the server side scripts, but Web applications are also characterized by an additional kind of control dependence, which is found in the HTML code between a tag and all directly enclosed statements[7].

2.2.2 Data Dependence

A data dependence holds between two server side statements or between a server side and an HTML statement if the former defines the value of a variable which is used by the latter, and a definition clear (i.e., a path containing no redefinition of the given variable) path exists between the two. Data dependence for traditional programs are associated with statements defining the value of a variable, which is used at another statement after being propagated to it along a definition clear path.[7]

2.2.3 Call Dependence

A call dependence holds between each HTML input statement of type submit and the dynamic page specified in the associated action. When a server side program is invoked from an HTML statement (e.g., via form submission or HREF), control of execution is transferred to the server and never returns back to the Web page issuing the call. In other words, a server program invoked from an HTML statement cannot produce any effect on the variables of the calling page, since the invocation is a no-return invocation. A consequence is that the calling context problem, i.e., the problem of keeping the data flows associated to the different call sites separated, is absent, since call sites are not affected by the invocation[7].

2.2.4 Parameter - In

A parameter-in dependence holds between an HTML input statement and a server side request-form statement if the form containing the input calls the dynamic page containing the request form and the names of input and requested variables are the same[7].

2.2.5 Parameter - Out

A parameter-out dependence holds between any value returned from an invocation and the related variable in the call site.

2.2.6 Summary Edge

In the case of slicing, summary edges represent a transitive data dependence through a particular call site. Summary edges connect actual-in vertices with actual-out vertices and represent a that dependence between the connected vertices may be created in the called procedure/pages or (transitively) through other called procedures/pages.

2.3 Web Application Dependence Graph

A System Dependence Graph is incapable of handling web applications and is only suitable for programs. Hence we introduce a Web Application Dependence Graph(WADG) with following additional dependences:-

2.3.1 Event Loop

Mouse or keyboard events occurring during or after page loading can trigger the execution of a client side procedure, can produce the loading of another page (hyperlinks), or the calling of a JavaScript. The graphical user interface of the browser handles such events by means of a so called event loop.

2.3.2 PageCall Dependence

A PageCall dependence holds between each HTML input statement of type submit and the dynamic page specified in the associated action.

2.3.3 Build Dependence

A build dependence holds between a server program statement and an HTML statement if the former generates the latter. During slice computation, build dependences are traversed backward similarly to the other dependences. In this way, the statements in the original program responsible for generating the HTML statements included in the slice will be also part of the slice.[11]

In the above example (in Figure-1), the first jsp page consists of two subsections of code, i.e, HTML

```
aa.jsp
1 <%
2 z = 'aa';
3 %>
4 <FORM method="POST" action="bb.jsp">
5 <INPUT TYPE="Submit"><BR>
6 <INPUT TYPE="Text" NAME="x" VALUE=z><BR>
7 <INPUT TYPE="Text" NAME="y" VALUE=z><BR>
8 </FORM>
bb.jsp:
9 <%
10 var x', y';
11 x' = request.getParameter("x");
12 y' = request.getParameter("y");
13 %>
```

Figure 1: Sample JSP program

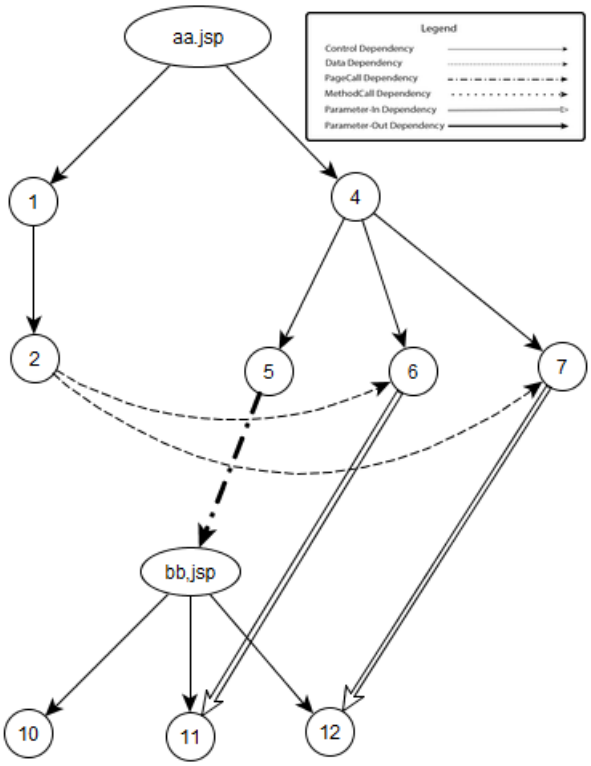

Figure 2: SDG of the program in fig-1

and Java beginning at statements 1 and 4 respectively. Thus, there is a control dependence between the page name and the two statements. Similarly, there is a control dependence between statement 1 and all other Java statements, i.e 1 and also between statement 4 and all other HTML statements, i.e,5,6,7 enclosed within tag represented by statement 4.

Now, statements 6 and 7 get their value of x from statement 2, resulting in data dependence between statement 2 to 6 and 2 to 7.

Also, the page bb.jsp is called when the FORM in aa.jsp is submitted from click event of statement 5, resulting in a call dependence between statement 5 and bb.jsp. Values are passed from statement 6 to 11 and statement 7 to 12 resulting in a parameter-in dependence between those statements. The final SDG for the above example program is shown in Figure-2.

3 Related Works

In this section, we present a brief survey of the existing literatures those are closely related to our work.

3.1 Slicing of Web Applications

A web application consists of, a set of web pages displayed to the user and a set of server side programs (usually scripts). The web application performs some computation and produces output pages to be displayed. A web application slice is obtained from a given set of web pages and scripts, by removing HTML and script statements, so that part of the behavior of the initial web application is replicated.

Ricca et al. [12] have used ReWeb tool for creating the UML diagram for web application. He has developed SDG for web-based programs taking different edges for control dependency, data dependency, call dependency and semantic dependency. He has implemented his slicing technique on a Travel Agency web application, developed using ASP and HTML.

Sahu et al. [13] have proposed an algorithm for slicing of JSP web applications. They have constructed the SDG for the JSP program. The slice is coputed by traversing backward in the graph and marking the edges and nodes.

In a paper by Junhua et al. [14], they have proposed a new method named Program Dependency Hyper Graph (PDHG) to describe the dependency of a web application using hyper graph theory. Also proposed an algorithm for slicing using PDHG.

4 Motivation

1) Web applications are becoming huge day by day.
2) Static web applications are used widely in firms which needs debugging techniques.
3) Most web applications are dynamic in nature, debugging them becomes even more important.
4) Slicing of Web Applications helps a great deal with applications like debugging, code understanding, program testing, reverse engineering, software maintenance, reuse, software safety and metrics.

5 Objective

1) To study the importance of web slicing.
2) To generate SDG for web applications.
3) To propose WADG for web applications.
4) To implement static slicing in java server pages.
5) To implement dynamic slicing in java server pages.

6 Proposed WADG For Web Application

When a JSP page is called from an HTML statement of another JSP page, (e.g. via FORM submission or HREF), control of execution is transferred to the newpage and never returns back to the web page issuing the call. In other words, a server program invoked from an HTML statement cannot produce any effect on the variables of the calling page, since the invocation is a no-return invocation.

Definition- *PageCall Dependence*: A PageCall dependence holds between each HTML input statement of type submit and the dynamic page specified in the associated action.

6.1 WADG Generation

Algorithm-1 is used to generate the WADG of a given JSP program. It basically has 3 sections - JavaScript, HTML and JSP. First it reads file by file from a given folder. Then it processes its JavaScript part. It constructs node, inserts control and data dependence between these nodes. It also stores function names to a HashMap, FuncMap. It also stores variable names to a HashMap, JSmap.

First the algorithm finds whether the code is in Html or not.If it is in Html it calls the algorithm html-parser.In the HTML section it searches for <Form> tag and stores it's action in a HashMap, PageMap. It searches for <input> tags and stores their names in a HashMap, HTMLmap. If <input> tag contains a JavaScript function call, it adds a call dependency between this and matching nodes of FuncMap.ii.if in js function call it finds that html parser contains that parameters that have been passed to js function then it adds a summary edge between this and all entries of HTMLmap.This part is modularized because the Dynamic part also calls it. If that function contains parameter(s) then it adds param-in dependence between nodes from HTMLmap and JSMap. It process the JSP part at last. During this process, it constructs nodes for individual statements and inserts control and data dependencies between nodes. If line contains request. getparameter or any method through which it can access a parameter sent from other page, it adds param-in dependency between calling form node and current node.If it finds that a page returns some parameter from the page from which it was called it Adds param-out dependency between current node and destination form node.In order to handle the dynamic Web pages we have defined three algorithms.Since the Jsp part contains the dynamic code so the algorithm calls three different algorithm to handle that dynamic part.

After processing all the pages from a folder it adds PageCall dependency from each entry of PageMap to the corresponding pages.

We have taken a JSP program , as shown in Figure-3 and Figure-4 . This program takes an input from the user and calculate the factorial of the number and display it to the user. We have applied the Algorithm-1 and generated the WADG for the whole JSP program and shown in Figure-5.

6.2 WADG for dynamic code

The problem of statically determining the HTML code generated dynamically by a Web application is in general undecidable. Consequently, it is in general impossible to build an accurate SDG for a Web application that generates some HTML code at run time. Since it is not possible to determine the HTML code generated by a dynamic scripts in the general case, the typical patterns of code generation are considered and a technique to handle them is presented.

6.2.1 Code Extrusion

A JSP page may contain print statements which prints HTML code. In such cases, the output of such HTML code can only be know at runtime making it impossible to generate the WADG with simple source code analysis. To deal with this, we propose an algorithm of code extrusion, which converts and replaces the print statements with their HTML output in the source code.

6.2.2 String-Cat Propagation

A JSP page may contain variables initialized and concatenated with values which result in those variables containing HTML code. When print statements print such variables the same problem of dynamic code generation occurs as discussed in previous section, but cannot be resolved by code extrusion alone. To deal with this, we propose an algorithm of string-cat propagation, which converts and replaces the print statements with the values of variables, containing HTML code, in the source code.

6.2.3 Flow Information

It might happen that a given JSP variable is associated to more than one string-cat after flow propagation. This occurs, for example, when alternatives are in the code(such as If Else construct). To deal with this, we propose an algorithm of flow information , which converts and replaces the print statements with an <ALT><CASE> HTML construct. In the presence of loops,an <ALT><CASE> construct with k(properly set) iterations is considered.

Algorithm 1 WADG Generation

INPUT: P- Input program, I- Input set for P,
Slicing criterion s

OUTPUT: The Slice S for s.

```
1: for each JSP page do
2:     while !End of File do
3:         Line = readline
4:         if Line = comment then
5:             continue
6:         else if Line = JavaScript then
7:             a) Store function names to a HashMap,
8:                FuncMap
9:             b) Store variable names to a HashMap, JSmap
10:            c) Construct nodes for individual statements
11:            d) Insert control and data dependencies
12:               between nodes
13:        else if Line = HTML then
14:            a) htmlParse(line)
15:        else if Line = JSP then
16:            Construct nodes for individual statements
17:            Insert control/data dependencies between nodes
18:            if Line contains request.getparameter then
19:                Add param-in dependency between calling
20:                    form node and current node
21:            if Line contains response.sendredirect then
22:                Add param-out dependency between
23:                 current node and destination form node
24:            e) Add Summary edge between calling form node and destination form node.
25:            f) codeExtrusion(line)
26:            g) stringcatPropagation(line)
27:            h) flowInformation(line)
28: Add PageCall dependency from each entry of PageMap to
29:  the corresponding pages
```

Algorithm 2 Html Parser

```
1: if < form > tag found then
2:     Store action in a HashMap, PageMap
3: if < input > tag found then
4:     Store names in a HashMap, HTMLmap
5:     if < input > contains jS function call then
6:         Add a call dependency between this and
7:                 matching nodes of FuncMap
8: Add param-in dependency between nodes from
9:          HTMLmap and JSMap
```

```
index.jsp
4  <html>
5  <head>
6  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7  <title>Insert title here</title>
8  <script>
9  function validateForm() {
10     var x = document.forms["form1"]["x"].value;
11   if(parseInt(x)!=x||parseInt(y)!=y || parseInt(q)!=q)
12         {
13                   alert("Enter Integers only as Input");
14                   return false;
15         }
16     else
17         return true;
18 }
19 </script>
20 </head>
21 <body>
22 <form action="factorial.jsp" method="post" name="form1" >
23 <input type="text" name="x" /><br>
24 <input type="submit" name="max" onclick="return validateForm()"/>
25 </form>
26 </br>
27 <%
28 String z;
29 z=request.getParameter("result");
30 if(z!=null)
31         out.println("<input type=\"text\" name=\"result\" value=\""+z+"\"/>");
32 %>
33 </body>
34 </html>
```

Figure 3: Example JSP program

Algorithm 3 CodeExtrusion

1: **If** line contains System.out.println:
2: a. Replace the print statement with an unquoted version of the printed string
3: b. Replace concatenated variables with a print within JSP delimiters (<% %>).
4: c. Call htmlParse(line).
5: d. Add build dependency between print statement and generated HTML statement.

```
factorial.jsp
4  <html>
5  <head>
6  <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
7  <title>Insert title here</title>
8  </head>
9  <body>
10 <%
11 String x,y,q;
12 int z,i;
13 x=request.getParameter("x");
14 z=Integer.parseInt(x);
15 for(i=z-1;i>1;i--)
16         z=z*i;
17 response.sendRedirect("index.jsp?result="+z);
18 %>
19 </body>
20 </html>
```

Figure 4: JSP program for calculating factorial of a number

Algorithm 4 StringCatPropagation

1: Create global HashMap strMap.
2: **if** line is a string assignment:
3: a. Create a new entry in strMap with variable name as key and assigned string as value.
4: **else if** line is a string concatenation:
5: a. Update strMap with new value of string.
6: **else if** line contains System.out.print:
7: **if** argument of print contains variable from strMap:
8: i. Replace variable with corresponding value from strMap.
9: ii. codeExtrusion(line).
10: **Else**
11: i. codeExtrusion(line).

Algorithm 5 FlowInformation

1: Create global HashMap altMap.
2: **if** line conatins if or line conatins else:
3: a. Create multiple value entry in altMap with variable name as key and values from if and else.
4: **else if** line conatains loop:
5: a. Create multiple value entry in altMap with variable name as key and values from k iterations.
6: **else if** line contains System.out.print:
7: **if** argument of print contains variable from altMap:
8: i. Replace print statement with <ALT><CASE> construct using values from altMap.
9: ii. codeExtrusion(line).

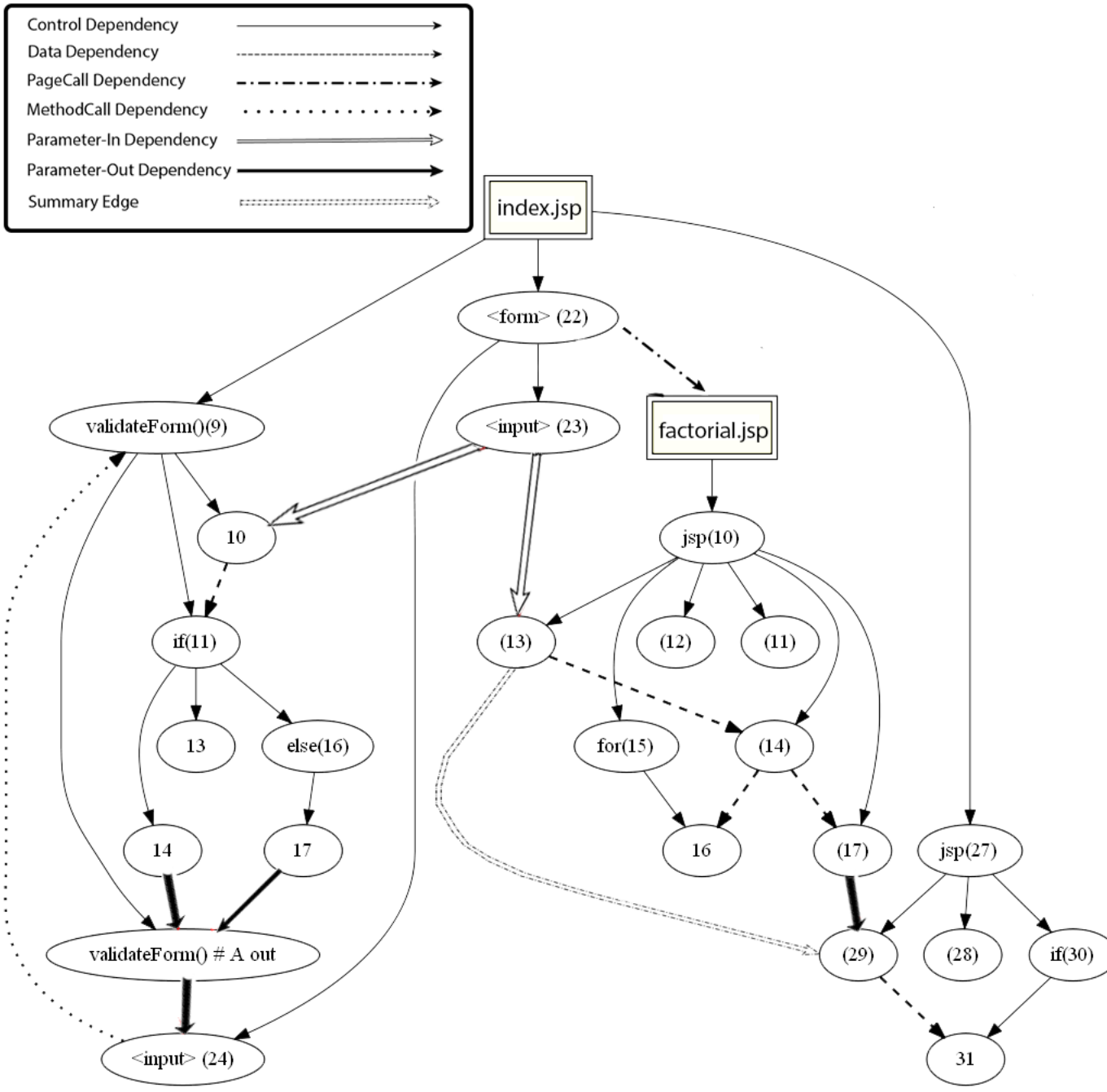

Figure 5: Generated WADG for the example programs in fig-3 and fig-4

Algorithm 6 Slicing Algorithm : Two Phase

INPUT: G- A WADG, s- Slicing Criterion,

OUTPUT: The Slice S for s.
lists and the result set

```
1:      W1 ={s}, W2 = {}, S ={s} //two work lists and the result set
       /* phase 1 */
2: repeat
3:      W1 =W1/ {n} // process the next node in W1
4:      for all m− >e n // handle all incoming edges of n
5:          if m ∉ S // m has not been visited yet
6:              S = S∪{m} // if e is not a param-out edge, add m to W1, otherwise, add
   m to W2
7:                  if e ∉ {po}
8:                      W1 = W1 ∪ {m}
9:                  else
10:                     W2 = W2 ∪ {m}
11:     Until W1 = ∅
       /* phase 2 */
12: repeat
13:     W2 =W2/ {n} // process the next node in W2
14:     for all m− >e n // handle all incoming edges of n
15:         if e ∉ {pi, call}
16:                 W2 = W2 ∪ {m}
17:                 S = S ∪ {m}
18:     Until W2 = ∅
19:     return S
20:
```

7 Slicing Algorithm

After the construction of WADG for a given JSP program, we have to compute slices by taking different slicing criteria. We have used a two-phase slicing algorithm [15], as shown in Algorithm-2, for our slice computation. Slicing as described in previous section is the process of selecting statements from the given web application with respect to an information of interest(slicing criterion) that replicates a part of the application itself. Sticking to this definition, Algorithm 2 takes an WADG of the web application and a slicing criterion as input, and gives a selected set of nodes from the WADG called slice as output. It performs the slicing operation in 2 phases, hence the name. The Algorithm recursively marks nodes of the WADG, starting with the slicing criterion itself, then proceeding to all its incoming edges.

In the first phase, all edges except parameter-out edges are considered for marking.In the second phase, the parameter-out edges left in phase one are considered separately. This is done in order to avoid some undesirable nodes from being marked, as they are unrelated to the slicing criterion and should not be marked. As seen from the phase two of the algorithm, all the parameter-in and call edges are ignored and not added to slice.Think of this like a portion of code in one page gives an output which gets received and stored at some statement(say ST) of some other page. Now, in this situation, that particular portion of code has absolutely no relation with any other code which calls ST or gives an output to ST, hence should not be included in the slice, as achieved by the second phase of the algorithm.

To explain the working of the slicing algorithm, we have used the same example WADG shown in Figure-5. Suppose the slicing criterion for this example is $s = 31$. By using the slicing algorithm we have computed slice and shown it in Figure-6. Here in this figure, the nodes included in the slice is shown as shaded nodes.

8 Experimental Study

We have developed a partial tool for automatic generation of WADG for a given JSP program. In the following section we are presenting the detailled implementation of our tool. Later, we have taken some real case studies to validate the working of our developed tool and slicing algorithm. We have performed the case studies with a personal computer having Intel Core $i5$ processor, clock speed 2.40GHz, primary memory 4 GB and Windows 7 Home Basic (64 bit) operating system.

8.1 Experimental Setup

In the block diagram, shown in Figure-7, there are six sections that reads and analyzes each and every file of the web application sequentially and generates the WADG.
At first the file is processed by JavaScript Analyzer that finds all the function names and variables names and stores them in different hash maps in key value pair. It also computes control and data dependence between JavaScript statements and sends results to the combined output section.

In the next section, HTML analyzer looks for two tags, i.e,$< input >$ and $< form >$ tags. If they contain any JavaScript function calls, a call dependency is added between them and matching nodes of the function map generated by the JavaScript section. If that function contains parameters, then a param-in dependence is added between nodes from HTMLmap and JSMap. Moreover, it computes control dependence between nodes and sends the results to the combined output section.

Finally, JSP analyzer process the JSP part. Control and data dependence between nodes is computed followed by a check for param-in dependence by checking the methods. Furthermore, it also checks if a page returns any parameter from which it was called and adds param-out dependence. The total processed result is sent to the combined output section of 3 analyzers. In the combined output of 3 sections, page call dependence is added between pages by checking the entry of the hash map provided by HTML section. Then it gives its output to the WADG generator section.

The WADG generator section receives the output of the previous sections and generates the WADG by writing a .gv file in the system. It adds all the dependencies according to the .gv file and also provides different dependencies by different notations. It generates the WADG completely and it gives its output to the WebSlicer. The WebSlicer receives the generated WADG file and the slicing criterion as input. Then, it process the WADG file and finds the node given by the slicing criterion. Finally, it generates the sliced WADG by applying the slicing algorithm on the input WADG and writes it to a .gv file in which the sliced nodes are marked.

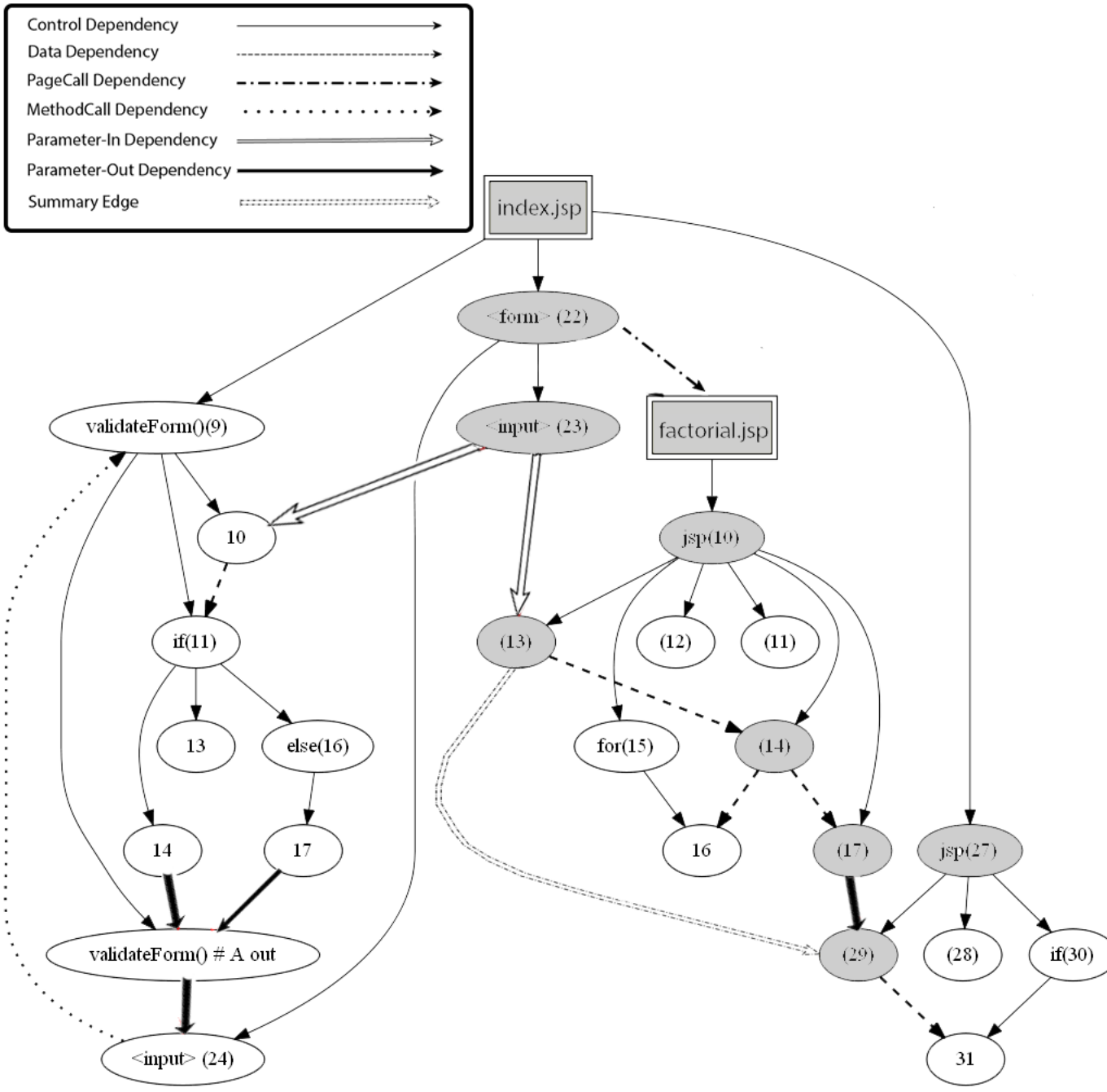

Figure 6: Generated slice for WADG in fig-5 w.r.t. $s = 31$

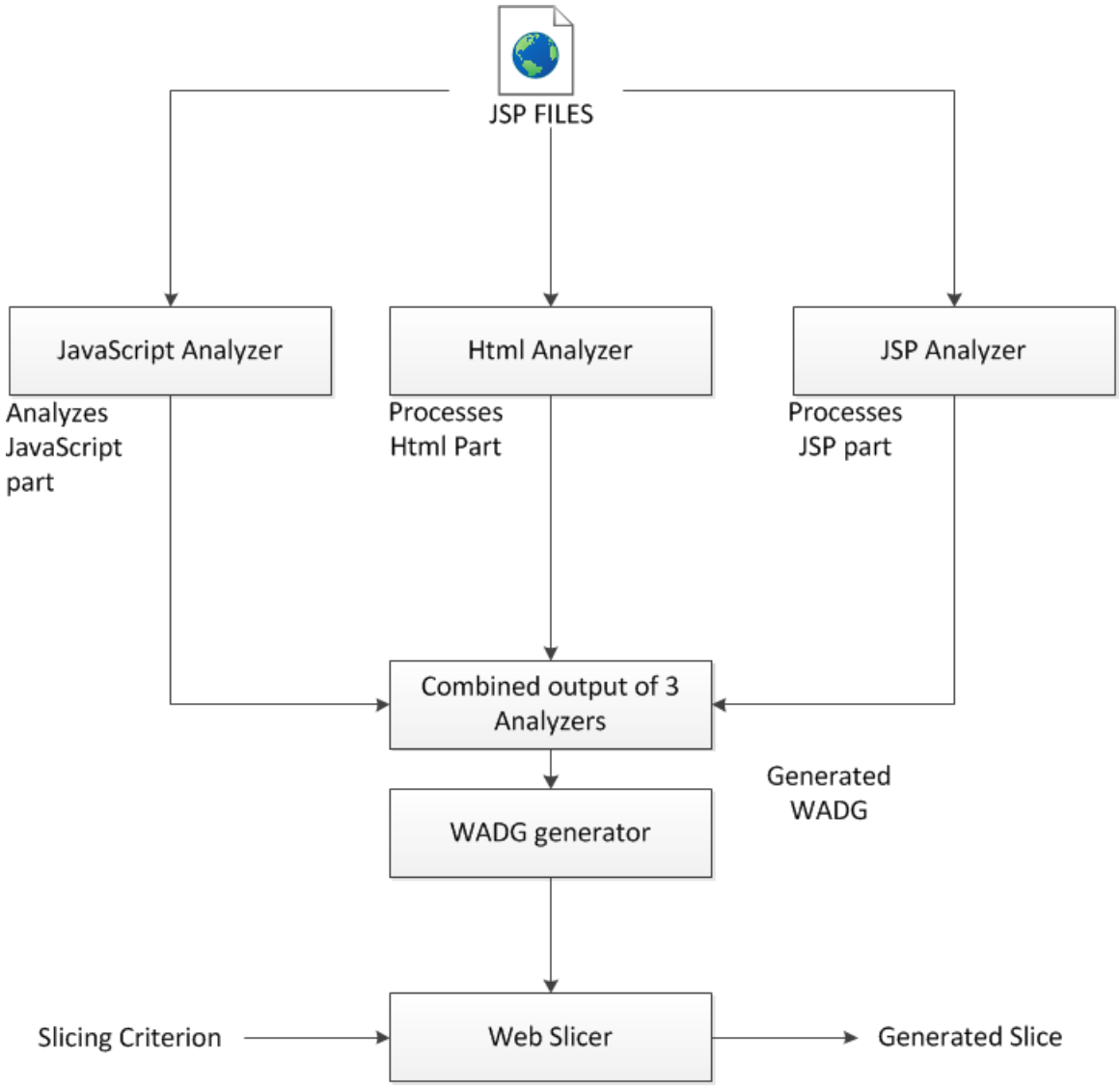

Figure 7: A systematic block diagram for WebSlicer

8.2 Case Study

In-order to verify the ability of our designed WebSlicer tool, we have preformed three case studies. In each case study, we have taken one JSP program, then generated it's WADG. After that we have computed slices for the same program and WADG, by taking different slicing criteria. Choosing of slicing criteria is a difficult task, but we have fixed the output statements, like print statement or method return statements, as slicing criteria. The details of case studies is givien in Table-1.

Table 1: Case study of various applications

Sl. No.	**Name of Application**	**Details**
1	Calculator	Performs mathematical operations
2	Book Management System	Storage and issue of text books for an institute
3	Java EE Training	Tutorials and practice questions of Java

8.3 Findings

First we have generated the WADG for each JSP applications, and the details are given in Table-2. Then we have computed several slices by supplying different slicing criteria. To summarise our finding, we have calculated the average slice size, which is the addition of individual slice size and number of slices computed. Similarly, we have computed the average slicing time, which is required to compute slices. Table-3 consists of all these findings.

Table 2: Details of generated WADG for case studies

Name of Application	**LOC**	**No Of nodes in WADG**	**No Of Edges in WADG**	**Time to Generate WADG**
Calculator	250	130	149	31.07 ms
Book Management System	834	527	612	114.39 ms
Java EE Training	435	281	288	54.57 ms

Table 3: Outcome of the case studies

Sl. No.	**Name of Application**	**Average Slice Size**	**Average Slice Time**
1	Calculator	7	2.61 ms
2	Book Management System	24	5.83 ms
3	Java EE Training	18	3.17 ms

9 Conclusion & Future Work

In this thesis, we have proposed a technique for slicing of JSP programs. Most of the existing techniques are based on byte-code analysis of the program, which causes more complex SDGs to represent and design. In our approach, we have used the source-code based program analysis for construction of the WADG for a given JSP program. We have applied a two-phase slicing algorithm to compute slices. Then to verify our developed tool, we have conducted some case studies. The slices computed by our tool is check manually to be correct. We have found that all the slices computed by our technique is precise and correct. This technique can be further enhanced and used for dynamic slicing and regression testing of web applications.

References

[1] W. Tsai, Xiaoying Bai, Ray Paul, and Lian Yu. Scenario-based functional regression testing. In *Computer Software and Applications Conference, 2001. COMPSAC 2001. 25th Annual International*, pages 496–501. IEEE, 2001.

[2] Wes Masri, Andy Podgurski, and David Leon. Detecting and debugging insecure information flows. In *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*, pages 198–209. IEEE, 2004.

[3] Lei Xu, Baowen Xu, Zhenqiang Chen, Jixiang Jiang, and Huowang Chen. Regression testing for web applications based on slicing. In *Computer Software and Applications Conference, 2003. COMPSAC 2003. Proceedings. 27th Annual International*, pages 652–656. IEEE, 2003.

[4] Junhua Wu, Baowen Xu, and Jixiang Jiang. Slicing web application based on hyper graph. In *Cyberworlds, 2004 International Conference on*, pages 177–181. IEEE, 2004.

[5] Mark Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering*, pages 439–449. IEEE Press, 1981.

[6] Mark Weiser. Program slicing. In *Proceedings of the 5th international conference on Software engineering*, pages 439–449. IEEE Press, 1981.

[7] Filippo Ricca and Paolo Tonella. Web application slicing. In *Proceedings of the IEEE International Conference on Software Maintenance (ICSM'01)*, page 148. IEEE Computer Society, 2001.

[8] Chengying Mao. Slicing web service-based software. In *Service-Oriented Computing and Applications (SOCA), 2009 IEEE International Conference on*, pages 1–8. IEEE, 2009.

[9] David W Binkley and Keith Brian Gallagher. Program slicing. *Advances in Computers*, 43:1–50, 1996.

[10] Andrea De Lucia. Program slicing: Methods and applications. In *2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 0144–0144. IEEE Computer Society, 2001.

[11] Paolo Tonella and Filippo Ricca. Web application slicing in presence of dynamic code generation. *Automated Software Engineering*, 12(2):259–288, 2005.

[12] Filippo Ricca and Paolo Tonella. Construction of the system dependence graph for web application slicing. In *Source Code Analysis and Manipulation, 2002. Proceedings. Second IEEE International Workshop on*, pages 123–132. IEEE, 2002.

[13] Madhusmita Sahu and Durga Prasad Mohapatra. Slicing java server pages application. 2008.

[14] H Casalánguida and JE Durán. Aspect oriented navigation modeling for web applications based on uml. *Latin America Transactions, IEEE (Revista IEEE America Latina)*, 7(1):92–100, 2009.

[15] Susan Horwitz, Thomas Reps, and David Binkley. Interprocedural slicing using dependence graphs. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(1):26–60, 1990.