Web Application Slicing

Filippo Ricca and Paolo Tonella
ITC-irst
Centro per la Ricerca Scientifica e Tecnologica
38050 Povo (Trento), Italy
{ricca, tonella}@itc.it

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. We propose to extend the extraction of slices to Web applications, in order to produce a reduced Web application which behaves as the original one with respect to some criterion, i.e., some displayed information of interest.

After presenting the theoretical implications of applying slicing to Web applications, we will demonstrate its usefulness with reference to an example, derived from a survey of a set of travel agency sites. Web application slicing helps disclosing relevant information and understanding the internal system structure.

1 Introduction

The economic relevance of Web applications increases the importance of controlling and improving their quality. Consequently, a high demand for methodologies and tools is emerging for the enhancement and assurance of the quality of Web based systems.

There are different ways to improve the quality of Web applications. Some of them are focused on the forward engineering step, proposing models and formalisms aimed at supporting the design of Web applications. Others assume that a Web application already exists and support its analysis, testing and evolution. While substantial effort was devoted to investigating the design of Web applications, only few works have insofar considered the problems related to analysis and testing of Web sites. Among the design models which have been proposed in support to Web development, that in [3] is relevant for this paper because the author uses a model similar to ours, where web pages are represented as objects, using UML [1]. One of the first systematic studies on Web site analysis is [13], where the evolution of the entities in Web sites is characterized by means of a set of metrics traced over time. In [11] we proposed a model for Web applications and several static analyses (structural and historical) based on it, derived from those used with software systems. In [12] we described a possible approach to structural testing of Web applications. A related work on Web application testing is presented in [10].

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 was introduced for the first time by Weiser [14] and has now many applications such as debugging, code understanding, program testing, reverse engineering, software maintenance, reuse, software safety and metrics [2, 4, 6, 8]. 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 can be extended to Web applications.

In this paper, we will present an approach to Web application slicing. 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 testing.

The remainder of this paper is organized as follows: the next section gives some information on our model for Web applications and on a tool providing automatic support for its extraction. Section 3 introduces the adopted approach to Web applications slicing, while its usage for the analysis of real Web applications is presented on an example in Section 4. Finally, Section 5 concludes the paper.

2 Background

Before considering Web application slicing a conceptual model of Web applications and a tool providing (partial)

automatic support to recover it, are presented.

2.1 Modeling Web applications

Figure 1 shows the meta model used to describe a generic Web application. It is given in the Unified Modeling Language (UML) [1]. The central entity in a Web site is the *Web Page*. A Web page contains the information to be displayed to the user, and the navigation links toward other pages. It also includes organization and interaction facilities (e.g., frames and forms). Navigation from page to page is modeled by the auto-association of class *WebPage* named *link*. Web pages can be static or dynamic. While the content of a *static* Web page is fixed, the content of a *dynamic* page is computed at run time by the server and may depend on the information provided by the user through input fields. The two subclasses of *WebPage* model such alternatives. When the content of a dynamic page depends on the value of a set of input variables, the attribute *use* of class *DynamicPage* contains them. A *frame* is a rectangular area in the current page where navigation can take place independently. Moreover the different frames into which a page is decomposed can interact with each other, since a link in a page loaded into a frame can force the loading of another page into a different frame. This can be achieved by adding a `target` to the hyperlink. Organization into frames is represented by the association *split into*, whose target is a set of *Frame* entities. Frame subdivision may be recursive (auto-association *split into* within class *Frame*), and each frame has a unary association with the Web page initially loaded into the frame (absent in case of recursive subdivision into frames). When a link in a Web page forces the loading of another page into a different frame, the target frame becomes the data member of the (optional) association class *LoadPageIntoFrame*. In HTML user input can be gathered by exploiting *forms*. A Web page can include any number of forms (association *include*). Each form is characterized by input and hidden variables (composed of name and value). In the case of input variables, values are provided by the user through the form, while in the case of hidden variables, values are not subject to user input, being constants embedded into the form. Variables (names and associated values) are submitted to the Web server via the special link *submit*, whose target is always a dynamic page. Since links, frames and forms are part of the content of a Web page, and for dynamic pages the content may depend on the input variables, even the organization of a page is, in general, not fixed and depends on the input. This is the reason for the association class *ConditionalEdge*, which optionally adds a boolean condition, function of the input variables, representing the existence condition of the association (which can in turn be a *link*, an *include* or a *split into*). The target, page, form or frame, is referenced by the source dynamic page only when the input values satisfy the condition in the *ConditionalEdge*.

2.2 The tool ReWeb

The **ReWeb** tool [11] consists of three modules: a Spider, an Analyzer and a Viewer. The Spider downloads all pages of a target web site, starting from a given URL, and builds the UML model of it, in agreement with the meta model showed in Figure 1. Each page found within the site host is downloaded, while HTML documents outside the Web site host are not considered. The user has to specify in a file the values of the inputs for each page containing forms. The Analyzer uses the UML model of the Web site and the downloaded pages to perform several structural analyses, described in [11]. The Viewer provides a Graphical User Interface (GUI) to display the Web application model as well as the output of the static analyses [11]. Web Spider and Analyzer are written in Java, while the Viewer is based on Dotty[1].

3 Web application slicing

According to its original definition [14], 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. The main requirements in this definition are that a slice be still a legal program and that its behavior be preserved with respect to a computation of interest.

Similar constraints have been enforced in introducing a notion of slice for Web applications. 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. Being a Web application, it can be browsed by the user. Moreover it has the property of providing the same outputs at a given page as the original Web site for every navigation path chosen and for every input provided.

Definition: *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.*

The criterion for slice computation is an information of interest displayed in a given Web page, and the resulting slice reproduces the same information, if the user performs

[1] Dotty is a customizable graph Editor developed at AT&T Bell Laboratories by Eleftherios Koutsofios and Stephen C. North.

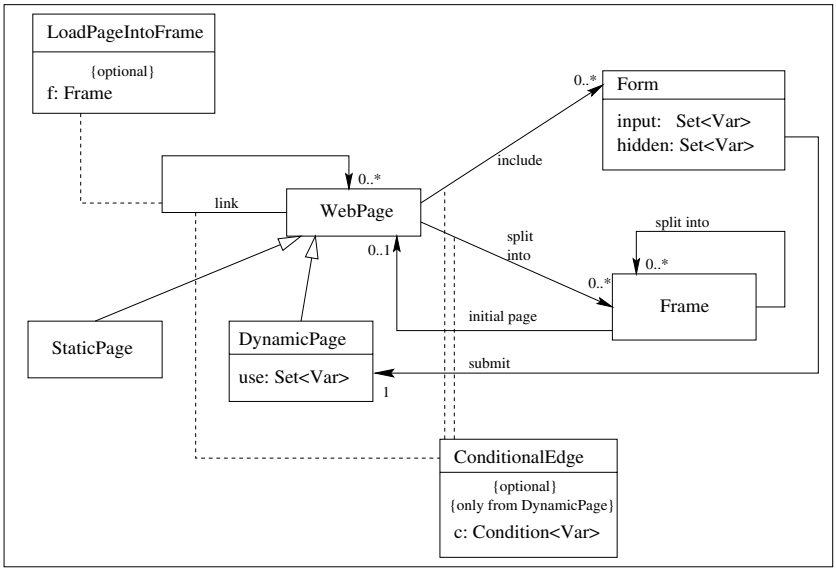

Figure 1. *Meta model of a generic Web application structure. The model of a given site is an instantiation of it.*

the same navigation actions and provides the same input in the original and in the sliced Web application.

Program slices can be computed by incrementally adding consecutive sets of transitively relevant statements. Statements that directly affect the computation at another statement are connected to the latter by a dependence relation, which can be explicitly represented in a graph called System Dependence Graph (SDG) [9]. Different kinds of dependences are distinguished in the SDG (control, intraprocedural flow, call, parameter-in, etc.) and the problem of static slicing is restated in terms of a two-step reachability problem in the SDG.

In the following we discuss the kinds of dependences which hold for Web applications and the main differences with respect to those considered for traditional software. A Web application slice is computed as a one-step transitive closure over all dependences in the associated SDG. In fact, the characteristics of Web applications allow a simplification of the original two-step algorithm, into a single step one.

Since Web page description languages and server side scripting languages are continuously changing and there is no accepted standard, we decided to adopt a fictitious language, called *PseudoVBScript*, which provides a limited set of instructions common to all available languages and rich enough to allow the definition of realistic Web applications. It can be easily mapped to any available alternative language (VBScript, Perl, etc.). PseudoVBScript is actually a language with a twofold purpose: defining Web pages and server side programs. This is achieved by mixing HTML code with instructions to be interpreted by the server (enclosed within the pseudo-tags `<% %>`). A similar feature is available from VBScript, which is the most similar among the available languages.

3.1 Control dependences

Control dependences 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. In fact, an HTML statement can be interpreted correctly by the browser only if all enclosing HTML tags are available. Therefore a control dependence holds between a tag and all directly enclosed statements. Since PseudoVBScript does not contain jump statements, it is possible to give a unique definition of control dependence, based on statement nesting, which is valid for both scripting and HTML instructions. In presence of jump instructions in the scripting language, a more complicated definition, based on the postdominance relation [5], would be needed. Self dependences associated with loop statements [5] are omitted, being not relevant for slicing.

Definition: *a* control dependence *holds between two PseudoVBScript statements if the former defines a scope which directly includes the latter.*

Figure 2 contains a fragment of PseudoVBScript code, in which the first 6 lines are executed by the server, while the last 4 lines are HTML statements directly passed to the client browser. The server execution consists of testing the value of variable `x`, and if it contains the string `'xx'`, the

cd.asp:

```
1   <%
2   if x = 'xx' then
3       y = 'yy'
4       z = 'zz'
5   endif
6   %>
7   <FORM method="POST" action="aa.asp">
8       <INPUT TYPE="Text" NAME="w" VALUE='ww'><BR>
9       <INPUT TYPE="Submit"><BR>
10  </FORM>
```

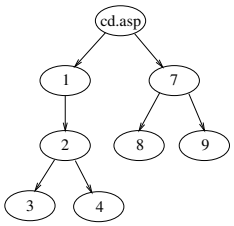

Figure 2. *Fragment of PseudoVBScript code and associated control dependences.*

dd.asp:

```
1   <%
2   x = z
3   if x = 'a' then
4       y = '1'
5   else
6       y = '2'
7   endif
8   %>
9   <FORM method="POST" action="aa.asp">
10      <INPUT TYPE="Text" NAME="w" VALUE=y><BR>
11      <INPUT TYPE="Submit"><BR>
12  </FORM>
```

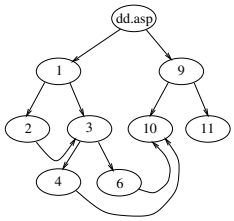

Figure 3. *Fragment of PseudoVBScript code and associated control dependences (straight lines) and data dependences (curve lines).*

two variables `y` and `z` are assigned a new value. The HTML code inserts a form in the page being constructed. Such a form includes a text field, named `w` and initialized with `'ww'`, for user input, and a submit button which executes the dynamic page `aa.asp`.

Control dependences for this code fragment are shown at the bottom. Node `cd.asp` is connected to the top level instructions. When a statement is inside a scope (e.g., the assignment at line 3), the associated node is connected as a child to the current scope node (e.g., the **if** at line 2), since there is a control dependence between the two. Statements which close a scope (e.g., the **endif** at line 5) have no associated nodes, in that the control dependence relation makes them unnecessary (e.g., nodes 3 and 4 are the children of node 2; no delimiter is required to enclose them inside the parent scope).

3.2 Data dependences

Data dependences 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 (i.e., a path containing no redefinition of the given variable). In addition to such a situation, which still can be found in server side code, data flows may occur in a Web application from server side statements to the HTML code. On the contrary, by no means a variable definition at an HTML statement can reach a script instruction inside the *same* dynamic page. It can be propagated to script instructions inside *other* dynamic pages as a submission parameter.

Definition: *a* data dependence *holds between two PseudoVBScript 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 path exists between the two.*

Figure 3 contains an example of both kinds of data dependences. The server side statement 2 defines the value of `x`, which is subsequently used (without intermediate redefinition) at statement 3. The two server side assignments to variable `y` at lines 4 and 6 have a data dependence with the HTML statement at line 10, where the default value of the input variable `w` is obtained from the value of `y`. The effect is that a text field is showed to the user inside the dynamic page `dd.asp` with an initial value equal to `y`. Collected input is then stored inside variable `w`. It should be noted that this is the main communication mechanism between server side statements and HTML variables, allowing the propagation of values from script code to Web pages. A variant of this mechanism, which is often exploited by Web applications, consists of the usage of a *hidden* HTML variable, the value of which is simply transmitted to the server side script invoked via form submission, being not subject to user input. In the example of Figure 3, this is achieved by replacing the input type at line 10 (`text`) with `hidden`. The form will not prompt the user for any input.

3.3 Call dependences

Interprocedural data dependences and slices for traditional software are made quite difficult to compute due to the *calling context problem*. This is associated to the additional constraints that must be applied to the interprocedural

data flows, when a procedure is invoked. Since a procedure may be invoked at more than one statement, it will receive several data flows from all calling instructions. Such data flows are transformed inside the called procedure and then must be returned back to the respective call sites. The problem of keeping the data flows associated to the different call sites separated is the calling context problem. The commonly adopted solution for slice computation is to determine the transitive data flows inside each procedure, in order to directly connect incoming with outgoing data. Then, the slice algorithm is split into two successive steps, which exploit the transitive dependences to allow interprocedural propagation, without mixing data flows from different call sites. More details on this algorithm are provided in [9].

Differently from traditional software, the calling context problem is completely absent in Web applications. In fact, by no means an invoked procedure (i.e., a dynamic page which is reached after form submission) can return some value to the calling page. Even more, there is no notion of *return* in Web applications, and dynamic pages are just traversed one after the other along some legal navigation path. As a consequence, data flows can only be propagated inside an invoked server side script (parameter passing is strictly by value), and never back from it. This property of Web applications highly simplifies interprocedural dependences computation and slicing. Transitive dependences are no longer required and the two step algorithm can be replaced by a single step one.

Definitions: (1) *a* call dependence *holds between each HTML input statement of type* `submit` *and the dynamic page specified in the associated action. (2) 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.*

In PseudoVBScript the mechanism for parameter transfer from a form to the invoked dynamic page is abstracted from all language-dependent variants into a generic *request-form* statement, which simply accesses the value of the parameter with a given name. The example in Figure 4 contains a form whose action is the server side script `bb.asp`. As a consequence, there is a call dependence between the submission statement (line 5) and `bb.asp`. Input parameters (`x` and `y`) are acquired inside the invoked script at lines 11 and 12, where their value is assigned to two new local variables, `x'` and `y'` (the same name, with an apostrophe, is retained just for convenience), which are declared at line 10. Consequently, a parameter-in dependence holds between nodes 6 and 11 and between 7 and 12.

```
aa.asp:
1   <%
2   z = 'aa'
3   %>
4   <FORM method="POST" action="bb.asp">
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.asp:
9   <%
10  dim x', y'
11  x' = Request.Form("x")
12  y' = Request.Form("y")
13  %>
```

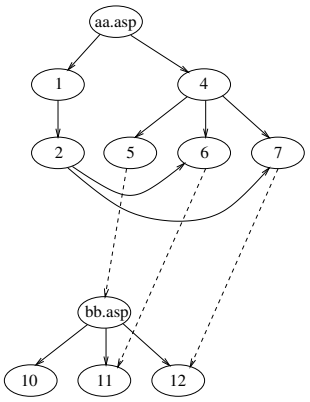

Figure 4. *Fragment of PseudoVBScript code and associated control/data dependences. Call/parameter-in dependences are depicted with dashed lines.*

3.4 Semantic dependences

If the information displayed in a Web page is considered part of the behavior of a Web application, then a slice should also include all text, images and other objects which provide relevant information on the ongoing computation. It is possible to formalize the dependence between PseudoVBScript statements and informative objects by means of semantic dependences. However, their determination can be only partially automated, e.g. by exploiting natural language processing techniques. They are described for completeness, but their full treatment is out of the scope of this paper.

```
sem.asp:
1   Provide a value for x:
2   <FORM method="POST" action="aa.asp">
3       <INPUT TYPE="Text" NAME="x"><BR>
4       <INPUT TYPE="Submit"><BR>
5   </FORM>
```

Figure 5. *Fragment of PseudoVBScript code with a semantic dependence between lines 1 and 3.*

Definition: *a* semantic dependence *holds between an informative object and a PseudoVBScript statement if the former provides information on the latter.*

In Figure 5, the meaning for the user of the text field at line 3 is given with the descriptive line number 1, which should be included in any slice on variable x, providing both computation and information about x. This can be achieved only if a semantic dependence is assumed between lines 1 and 3. In the following section, slices will be slightly less accurate, not including such dependences.

3.5 Slicing algorithm

```
SLICE(n: SDGNode): Set<SDGNode>
1   S ← {n}
2   while S increases
3       if a control/data/semantic dep. holds between n' and n ∈ S
4           S ← S ∪ {n'}
5       endif
6       if a parameter-in dep. holds between n' and n ∈ S,
            and the associated call dep. starts at n''
7           S ← S ∪ {n', n''}
8       endif
9   endwhile
10  return S
```

Figure 6. *Pseudocode of the slicing algorithm.*

Once control, data, call, parameter-in and (optionally) semantic dependences are available for a Web application, slices can be computed as the transitive closure of such dependences starting from a point of interest. A simple algorithm for this computation is described in Figure 6. In presence of a parameter-in dependence, the associated call dependence has to be traversed as well. Otherwise, call dependences are ignored.

3.6 Decomposition slicing

A decomposition slice [6] collects all the computation that a program performs on a variable of interest. Decomposition slices form a graph, which represents a decomposition of the computations in a program into maximal computations and dependent computations.

Definition: *a* decomposition slice $S(x)$ *on a variable* x *is the union of all slices computed at the nodes which output* x*.*

Output restricted decomposition slices, i.e., slices from which output statements are removed, can be ordered by set inclusion – such a relationship is called *strong dependence*. The strong dependence relationship can be associated with the edges of a direct acyclic graph, the nodes of which are decomposition slices, representing the decomposition of a program into high level computations and then down into included support computations. Top level slices are said *maximal*.

Extraction and visualization of the decomposition slices and the associated graph for a Web application is helpful in understanding its organization and in determining the main computations performed and the associated subcomputations. Such an information may help the programmers during the comprehension activities. Moreover, the decomposition slice graph highlights dependent and independent computations, allowing an evaluation of the impact of a modification inside a Web page on the remaining pages of the site [7].

4 Case study

To illustrate the proposed slicing approach, we will consider the Web application for a travel agency showed in Figure 7, and use slicing to isolate some of the ongoing computations. Moreover, the decomposition slice graph will be determined to highlight the dependences between the computations. This simplified Web site was obtained by abstracting the main characteristics present in a set of real travel agency applications downloaded from the Web.

4.1 Travel Agency Web application

In this Web site the users can select hotels, cars and flights and reserve/rent/book them on-line. For the sake of simplicity, the pages displaying error messages are not reported. When users select an item of interest from the static page `index.html` (Figure 7.a), a dynamic page `choose.asp` is created, as shown in Figure 7.b. If, e.g., the option *Hotels* was selected, users are asked to input the city where they want to go and the arrival and departure dates. After clicking the submit button the dynamic page `selectHotel.asp` (Figure 7.c) appears, showing a list of available hotels in the chosen city. The users can select one of them or change the option of interest passing from hotels to cars or flights. In the case of changing option, the page `selectHotel.asp` transmits arrival and departure dates via form to the page `choose.asp`, as well as the city. This values will be exploited by `choose.asp` to propose default values to the users in the reservation of cars or flights. Otherwise, if the users choose to select a hotel and press the related submit button, the dynamic page `hotelPrice.asp` is created, as shown in Figure 7.d. This page contains the chosen hotel, its daily price and the total cost for the entire stay. In the same page the user can insert name, surname and payment type to reserve a room in the selected hotel. The dynamic page `reserveHotel.asp` (Figure 7.e) appears after clicking the submit button in the previous page. It contains a simple

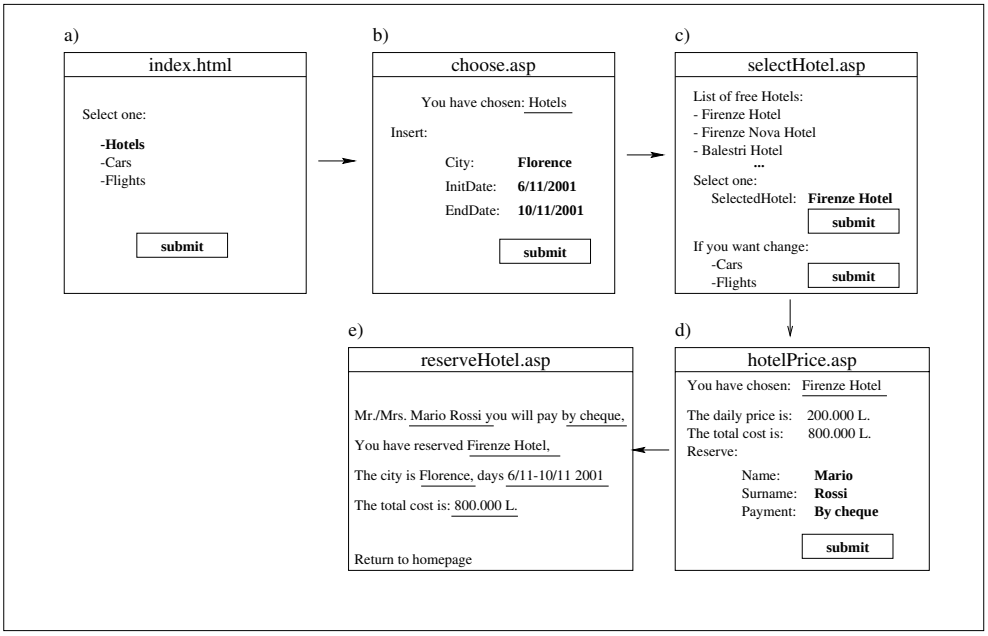

Figure 7. *A simple on-line Travel Agency Web application.*

report confirming the reservation. Then, users can return to the homepage and select cars or flights.

```
hotelPrice.asp:
220 <%Language=PseudoVBScript%><HTML><HEAD>
    <TITLE> Hotel costs</TITLE></HEAD><BODY><%
223,224,225,226 dim city', initDate', endDate', selectedHotel'
227,228 dim priceForDay, totalCost
229,230,231,232 city', initDate', endDate', selectedHotel' =
                Request.Form("city, initDate, endDate, selectedHotel")
233 Response.Write("you have choosen:", selectedHotel')
234 priceForDay = RetrieveHotel-PriceFromDataBase(selectedHotel')
235 totalCost = priceForDay'*Diff(endDate', initDate')
236 Response.Write("The daily price is:", priceForDay)
237 Response.Write("The total cost is:", totalCost)
238 Reserve:
239 <BR>
240 <FORM method="POST" action="reserveHotel.asp">
241     Name: <INPUT TYPE="Text" NAME="name" SIZE=20 ><BR>
242     Surname: <INPUT TYPE="Text" NAME="surname" SIZE=20 ><BR>
243     Payment: <INPUT TYPE="Text" NAME="payment" SIZE=20 ><BR>
244     <INPUT TYPE="hidden" NAME="city" VALUE=city'><BR>
245     <INPUT TYPE="hidden" NAME="initDate" VALUE=initDate'><BR>
246     <INPUT TYPE="hidden" NAME="endDate" VALUE=endDate'><BR>
247     <INPUT TYPE="hidden" NAME="selectedHotel" VALUE=selectedHotel'><BR>
248     <INPUT TYPE="hidden" NAME="cost" VALUE=totalCost><BR>
249     <INPUT TYPE="Submit"><BR>
250 </FORM>
251 %></BODY></HTML>
```

Figure 8. *Page* `hotelPrice.asp`.

Figure 8 contains the PseudoVBscript instructions of the dynamic page `hotelPrice.asp`, while Figure 9 shows the PseudoVBscript source of the dynamic page `reserveHotel.asp` [2]. Page `choose.asp` can be called by the static page `index.html` at the starting of the interaction with the Web site, or by the dynamic pages `selectHotel.asp`, `selectCar.asp` and `selectFlight.asp`, whenever the user decides to change option. In this latter case some values (for example arrival and departure date) will be passed to `choose.asp` which exploits them to propose default values to the user. The new option chosen by the user is passed via variable `x` and stored in the local variable `x'`.

[2] For simplicity, in these sources some of the tags `<% %>` have been omitted.

```
reserveHotel.asp:
278 <%Language=PseudoVBScript%><HTML><HEAD>
    <TITLE> Reserve Hotel </TITLE></HEAD><BODY><%
279,280,281 dim name', surname', payment'
282,283,284,285 dim city', initDate', endDate', selectedHotel'
286 dim totalCost
287,288,289 name', surname', payment' = Request.Form(name, surname, payment)
290,291,292,293 city', initDate', endDate', selectedHotel' =
                Request.Form("city, initDate, endDate, selectedHotel")
294 totalCost = Request.Form(cost)
295 Response.Write("Mr./Mrs ", name', " ", surname', "you will pay ", payment')
296 Response.Write("You have reserved ", selectedHotel')
297 Response.Write("The city is ", city', ", days ", initDate', "-", endDate')
298 Response.Write("The total cost is ", totalCost)
299 %><A HREF="index.html">Return to homepage</A></BODY></HTML>
```

Figure 9. *Page* `reserveHotel.asp`.

Figure 10 shows the model of the Web application. This model is a simplification (without pages containing error message and informative static pages) of the models produced by **ReWeb**, applied to the considered travel agency sites.

The code corresponding to pages `selectCar.asp` and `selectFlight.asp` is similar to `selectHotel.asp` (not shown here); `carPrice.asp` and `flightPrice.asp` are similar to `hotelPrice.asp`, while `rentCar.asp` and `bookFlight.asp` are similar

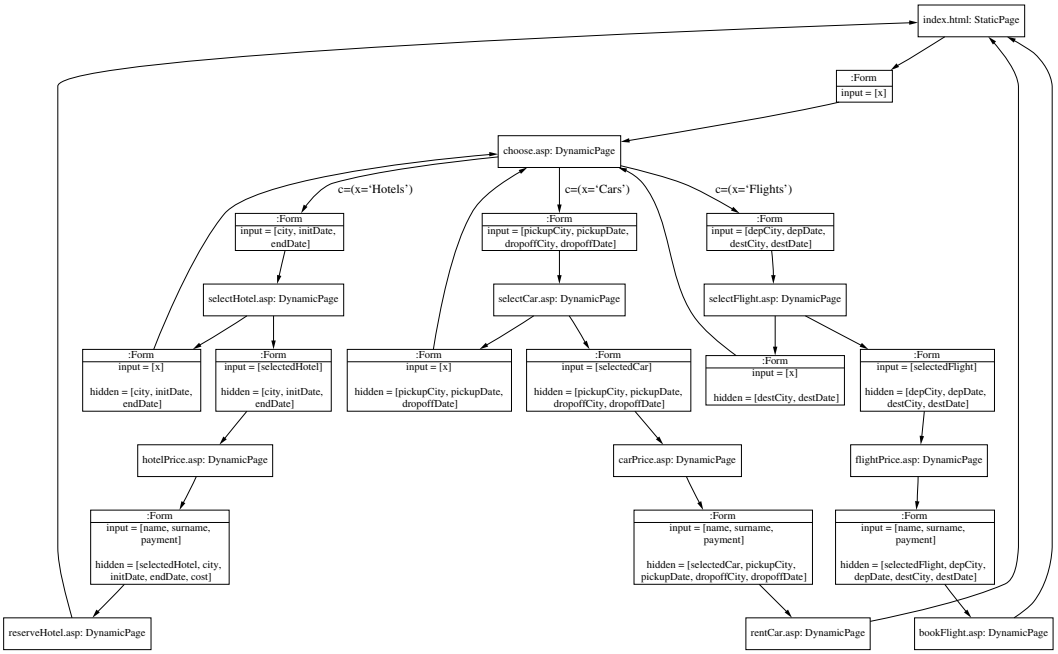

Figure 10. *Model of Travel Agency Web application.*

to `reserveHotel.asp`.

The only conditional edges present in the model are those outgoing from the node `choose.asp`. Their labels contain their existence condition, depending on the value of the variable `x` inserted by the user in the form contained in `index.html`, or in the form inside `selectHotel.asp`, `selectCar.asp` and `selectFlight.asp` and filled in when the users decides to change the selected option (Hotels, Cars or Flights).

4.2 Slice on variables name', surname' and payment'

The SDG of the travel agency Web application was built. A portion of it is shown in Figure 12 (limited to the dynamic pages `reserveHotel.asp` and `hotelPrice.asp`). The vertices of the slice taken with respect to the output node 295 (using the values of variables `name'`, `surname'` and `payment'`) are highlighted in Figure 12. This slice is the backward transitive closure of data, control and call dependences. The result of slicing is a simplified Web application (code is shown in Figure 11) containing only two pages and a form: `hotelPrice.asp`, containing a form node, where users can insert `name, surname and payment`, and page `reserveHotel.asp`, displaying name, surname and type of payment, as previously inserted. The processing of such values requires only to include the input form, which collects them for the user.

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
hotelPrice.asp:
220 <%Language=PseudoVBScript%><HTML><HEAD>
    <TITLE> Hotel costs</TITLE></HEAD><BODY><%
240 <FORM method="POST" action="reserveHotel.asp">
241     Name: <INPUT TYPE="Text" NAME="name" SIZE=20 ><BR>
242     Surname: <INPUT TYPE="Text" NAME="surname" SIZE=20 ><BR>
243     Payment: <INPUT TYPE="Text" NAME="payment" SIZE=20 ><BR>
249     <INPUT TYPE="Submit"><BR>
250 </FORM>
251 %></BODY></HTML>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
reserveHotel.asp:
278 <%Language=PseudoVBScript%><HTML><HEAD>
    <TITLE> Reserve Hotel </TITLE></HEAD><BODY><%
279,280,281 dim name', surname', payment'
287,288,289 name', surname', payment' = Request.Form(name, surname, payment)
295 Response.Write("Mr./Mrs ", name', " ", surname', "you will pay ", payment')
299 %><A HREF="index.html">Return HomePage</A></BODY></HTML>
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Figure 11. *Slice computed at line 295.*

4.3 Slice on variable totalCost

By applying the slicing algorithm at line 298 (variable `totalCost`), we obtain a Web Application, whose model is shown in Figure 14, simpler than the initial Web application, but with the same behavior on the variable `totalCost`. In each user interaction the resulting Web application displays the same values of the variable `totalCost` as the initial Web application. All statements in the original Web application involving the variables `name`, `surname`, and `payment` have been removed, being not relevant for the computation of the total cost.

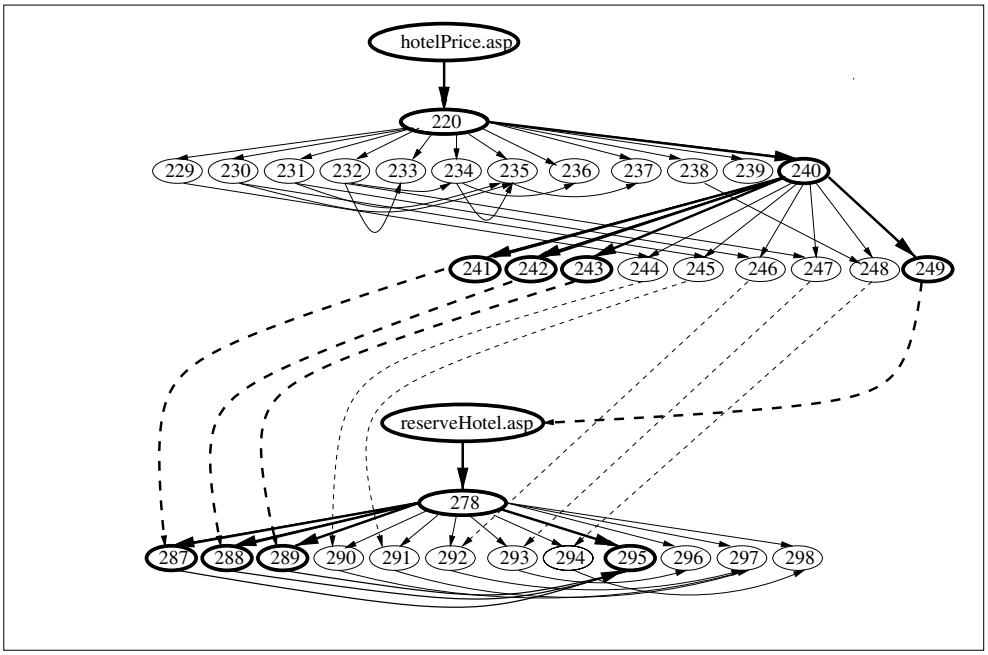

Figure 12. *SDG of a portion of Travel Agency Web application.*

4.4 Maximal and dependent computations

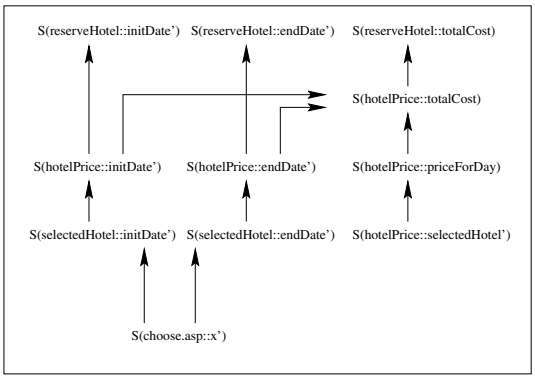

Figure 13. *Portion of Decomposition Slice Graph*

Figure 13 shows a portion of the decomposition slice graph of the travel agency Web application decomposition slices. S(reserveHotel::totalCost) (variable `totalCost` is local to the dynamic page `reserveHotel.asp`), S(reserveHotel::initDate') and S(reserveHotel::endDate') are maximal decomposition slices. S(hotelPrice::initDate'), S(hotelPrice::endDate') and S(hotelPrice::priceForDay) are strongly dependent on S(hotelPrice::totalCost), which is strongly dependent on S(reserveHotel::totalCost). S(choose::x') is strongly dependent on all the other decomposition slices (apart from S(hotelPrice::priceForDay) and S(hotelPrice::selectedHotel')). This graph highlights that the computation on `totalCost` needs the computation on `initDate'`, `endDate'` (which needs the computation on `x'`) and `priceForDay` (which needs the computation on `selectedHotel`). A change on the statements that define one of these variable has some consequence on the `totalCost` value.

5 Conclusions and future work

The computation of slices for the Travel Agency Web application provided very useful indications about the performed processing. The slice at line 295, focused on payment data, includes a very small fraction of the whole application. This is due to the locality of the manipulated data, which are directly provided by the user at the previous Web page. On the contrary, the information displayed at line 298 depends on several input forms and PseudoVBScript instructions which have to be included in the associated slice. The corresponding datum, the total cost of the reservation, is actually obtained along several alternative computation paths, inserted into the slice.

The lattice of the decomposition slices computed for the example application was also useful. It allowed an explicit visualization of the dependences between subcomputations, thus supporting impact analysis. Independent computations are also highlighted. Some of them, not shown in the figures for space reasons, are associated to the two alternative functions – car rental and flight booking – which, together with hotel reservation, make up the exam-

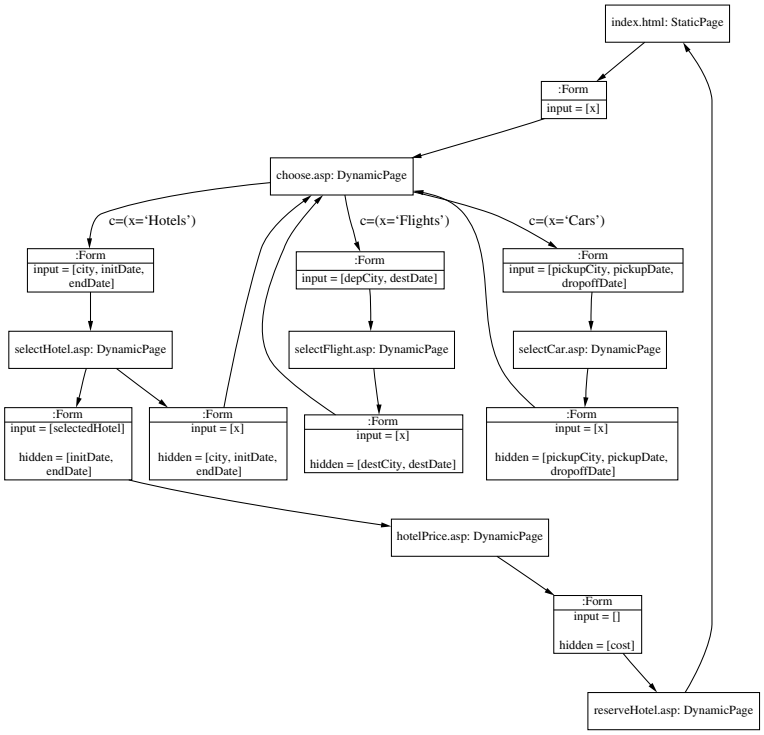

Figure 14. *Model of Travel Agency Web application sliced at line 298.*

ple application.

Our future work will be devoted to the extension of the current slicing tool so that it can handle some of the available server side languages (at present the conversion from a given language to PseudoVBScript is performed manually). Finally, the usage of natural language processing to include the semantic dependences into those considered during slicing is another stimulating research topic.

References

[1] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language – User Guide*. Addison-Wesley Publishing Company, Reading, MA, 1998.

[2] A. Cimitile, A. D. Lucia, and M. Munro. Identifying reusable functions using specification driven program slicing: A case study. In *Proceedings of the International Conference on Software Maintenance*, pages 124–133, Opio(Nice), 1995.

[3] J. Conallen. *Building Web Applications with UML*. Addison-Wesley Publishing Company, Reading, MA, 2000.

[4] S. Danicic, C. Fox, M. Harman, and R. Hieros. Consit: A conditioned program slicer. In *Proc. of ICSM 2000, International Conference on Software Maintenance, San Jose, California, USA, 11-14 October*, pages 216–226, 2000.

[5] J. Ferrante, K. J. Ottenstein, and J. D. Warren. The program dependence graph and its use in optimization. *ACM Transactions on Programming Languages and Systems*, 3(9):319–349, July 1987.

[6] K. Gallagher and J. Lyle. Using program slicing in software maintenance. *IEEE Transactions on Software Engineering*, 17(8):751–761, August 1991.

[7] K. B. Gallagher. Visual impact analysis. In *Proceedings of the International Conference on Software Maintenance*, pages 52–58, Monterey, 1996.

[8] R. Gupta, M. Harrold, and M. Soffa. Program slicing-based regression testing techniques. *Journal of Software Testing, Verification, and Reliability*, 6(2):83–112, June 1996.

[9] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Transaction on Programming Languages and Systems*, pages 26–61, 1 1990.

[10] C.-H. Liu, D. C. Kung, P. Hsia, and C.-T. Hsu. Structural testing of web applications. In *Proc. of ISSRE 2000, International symposium on software reliability engineering*, pages 84–96, San Jose, California, USA, October 2000.

[11] F. Ricca and P. Tonella. Web site analysis: Structure and evolution. In *Proceedings of the International Conference on Software Maintenance*, pages 76–86, San Jose, California, USA, 2000.

[12] F. Ricca and P. Tonella. Analysis and testing of web applications. In *Proc. of ICSE 2001, International Conference on Software Engineering, Toronto, Ontario, Canada, May 12-19*, pages 25–34, 2001.

[13] P. Warren, C. Boldyreff, and M. Munro. The evolution of websites. In *Proc. of the International Workshop on Program Comprehension*, pages 178–185, Pittsburgh, PA, USA, May 1999.

[14] M. Weiser. Program slicing. *IEEE Transactions on Software Engineering*, 10(4):352–357, July 1984.