WARE: a tool for the Reverse Engineering of Web Applications

Anna Rita Fasolino
G. A. Di Lucca, F. Pace, P. Tramontana, U. De Carlini

Dipartimento di Informatica e Sistemistica
University of Naples Federico II, Italy
Web Applications (WA): problems and open issues

• The pressing market demand of web applications
  – WAs developed in very short time

• The continuously changing needs of the evolving application domains
  – WAs frequently and rapidly modified with *ad hoc* approaches

• The lack of method in producing and maintaining Web applications
  – low quality software, with disordered architecture, and inadequate and incomplete documentation
Web applications: a recent classification

- Class 1: primarily static web sites

- Class 2: sites providing client-side interaction
  - Based on an event model exploiting some script code

- Class 3: applications with dynamic content
  - Pages are dynamically created on the fly, with the support of various WWW technologies (CGI, JSP, PHP, Javascript, XML, ODBC, ... ASP, ... )
Dynamic web applications

• Due to the large number of employed technologies, understanding, maintaining and evolving a Class 3 application is a complex task

• The need for specific Reverse Engineering techniques and processes to recover:
  – Static and dynamic aspects of the applications
  – Suitable representation models

• The need for reverse engineering tools that support the extraction and the abstraction of the needed information from a WA
Reverse engineering Web Applications

• A reverse engineering process to recover the following views:
  – The static architecture of the WA
  – The dynamic interactions between its components
  – The final behavior offered

• Extended UML diagrams to represent these views
  – Class diagrams to model the architecture
  – Sequence and Collaboration diagrams to represent the dynamic model
  – Use case diagrams to describe the behavior of the WA
The Reverse Engineering process

Web Application Source Code → STATIC ANALYSIS → Components and Direct Relations between them

Web Application Source Code → DYNAMIC ANALYSIS

WA in Execution → DYNAMIC ANALYSIS → Dynamic Interactions between Components

→ BEHAVIORAL ANALYSIS

→ Behavioral Model
Views obtainable by static analysis

• A coarse grained view:
  – Web pages and Hypertextual links between pages:
  – Pages are distinguished into server and client pages, static and dynamic pages, simple and framed pages

• A finer grained view:
  – inner page components and relationships
  – input/output form, text box, anchors, scripts, applets, text, images, multimedia objects (sounds, movies), …
  – Page components may be active components (e.g., scripts or applets)
  – The relationships include: submit, build, redirect, include

• A UML class diagram representing both views
The meta-model of a WA
The WARE tool

• Designed to support the reverse engineering of a WA
  – executes the static analysis of the WA source code
  – populates a repository with the extracted information
  – supports the user in abstracting the WA models

• Three main components:
  – Interface Layer
  – Service Layer
  – Repository
The Service layer

• The Extractor component
  – The Extractor parses the WA source code and produces an Intermediate Representation Form (IRF)
  – Several distinct parsers (for HTML, JavaScript, VBScript, ASP and PHP technologies)
  – New parsers can be added as the technology evolves

• The Abstractor component
  – It implements abstraction operations necessary for reconstructing the more abstract views of the WA
  – It includes three main components
    • A Translator that populates the relational database from the IRF
    • A Query Executor that implements predefined queries over the database
    • The UML Diagram Abstractor that produces the class diagrams of a WA at various degrees of detail and other relevant information
The conceptual schema of the Repository
The Interface Layer

- It provides the user interface for accessing the WARE facilities:
  - *Reverse Engineering* for parsing the WA and producing the IRF
  - *Comprehension* for executing comprehension-related activities, such as *exploring the inventory* of the WA components and their source code, computing the *reachability* set of a component, creating *clusters* of related components according to a given clustering criterion
  - *Query* (Predefined query and Parametric query) for activating the Abstractor’s functions, and graphically visualizing the recovered information
Figure 4.4 – b: WA Components reachable starting from a selected component

Computation of the Reachability set of a WA component

Grouping the WA components into cohesive clusters

CSMR 2002, Budapest, Hungary
An experiment

• Carried out for assessing the adequacy of the tool functions in supporting maintenance tasks

• The tool has been used to understand and redocument existing WAs

• An example: A WA implementing a ‘Juridical Laboratory’
  – 201 files, in 19 directories, sizing 4 Mbytes
  – 55 Static Pages (55 HTML files in 10 directories)
  – 19 Server Pages (19 ASP files in 4 directories)
  – No design documentation available

• The WARE tool was preliminarily used to perform:
  – The static analysis of the application, and for producing an inventory of the WA components
First step: the static analysis

• Main results:
  – The *population* of the repository automatically produced by the tool
  – The *inventory* of the WA components
  – *Graphical representations* of the class diagram of the WA at various degrees of detail

  • Coarse-grained representations (including just the pages)
  • Fine-grained representation (including inner page components)
The inventory of the WA components

<table>
<thead>
<tr>
<th>Component type</th>
<th># Detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server page</td>
<td>19</td>
</tr>
<tr>
<td>Client Static page</td>
<td>55</td>
</tr>
<tr>
<td>Client Built page</td>
<td>14</td>
</tr>
<tr>
<td>External web page</td>
<td>3</td>
</tr>
<tr>
<td>Client script block</td>
<td>53</td>
</tr>
<tr>
<td>Function in Client script block</td>
<td>19</td>
</tr>
<tr>
<td>Form</td>
<td>11</td>
</tr>
<tr>
<td>Input/ output field</td>
<td>71</td>
</tr>
<tr>
<td>Submit Operation (POST method)</td>
<td>4</td>
</tr>
<tr>
<td>Submit Operation (GET method)</td>
<td>7</td>
</tr>
<tr>
<td>Anchor to files to be downloaded</td>
<td>111</td>
</tr>
<tr>
<td>Anchor to Hypertextual link</td>
<td>49</td>
</tr>
<tr>
<td>Data File</td>
<td>61</td>
</tr>
<tr>
<td>Server script block</td>
<td>76</td>
</tr>
<tr>
<td>Function in Server scripts</td>
<td>4</td>
</tr>
<tr>
<td>Database Interface Object</td>
<td>29</td>
</tr>
<tr>
<td>Mail Interface Object</td>
<td>3</td>
</tr>
<tr>
<td>Image file</td>
<td>65</td>
</tr>
<tr>
<td>Redirect operation in server blocks</td>
<td>7</td>
</tr>
</tbody>
</table>
The Class diagram of the WA
The second phase: formulating and validating hypotheses about the WA behavior

- Driven by the graphical representations, notable sub-graphs were looked for (isolated sub-graphs, sub-trees, strongly connected components, …)

- A tentative hypothesis about the behavior implemented by each sub-graph was formulated with the support of:
  - the names of the components, their source code analysis, and by tracing the application execution

- Hypotheses were validated by the source code execution
Results from the sub-graph analysis

• Four isolated sub-graphs
  – Three small ones without static client pages
    • Two of them implemented server-side functions for the web administrator (management of the mailing list, and of the registered users list)
    • The third one resulted from an incorrectly made maintenance operation
  – A large one, rooted in the home page
    • Nine user functions could be associated with nine notable sub-graphs contained in it
The sub-graphs in the WA class diagram
Modeling the WA behavior

- A use case was defined and associated with each notable sub-graph

- The scenarios describing the use cases were defined with the support of the tool (the interactions between objects involved in the sub-graph were searched for and analyzed)
Modeling the WA dynamic view
Conclusions

• The experiments we carried out showed that the reverse engineering tool WARE can be used to support the comprehension of Web Applications to be maintained

• It supports the reconstruction of various UML diagrams from undocumented applications:
  – Static views are automatically produced by the tool
  – Behavioral and dynamic views can be semi-automatically obtained with the tool assistance

• A reverse engineering process based on the WARE tool could be defined
Future work

- Specific clustering criteria may support a more effective analysis of the class diagrams recovered by the tool.
- A clustering approach should take into account both the topology of the graphs, and the typology of the connections in the graph.
- The automatic clustering of Was will be investigated in the context of large size web applications.