WARE: a tool for the Reverse Engineering of Web Applications

G. A. Di Lucca, A. R. Fasolino, F. Pace, P. Tramontana, U. De Carlini
{dilucca/ fasolino/ fabpace/ ptramont/ decarl}@unina.it

Dipartimento di Informatica e Sistemistica, Università di Napoli Federico II
Via Claudio 21, 80125 Napoli, Italy

Abstract

The development of Web sites and applications is increasing dramatically to satisfy the market requests. The software industry is facing the new demand under the pressure of a very short time-to-market and an extremely high competition. As a result, Web sites and applications are usually developed without a disciplined process: Web applications are directly coded and no, or poor, documentation is produced to support the subsequent maintenance and evolution activities, thus compromising the quality of the applications.

This paper presents a tool for reverse engineering Web applications. UML diagrams are used to model a set of views that depict several aspects of a Web application at different abstraction levels. The recovered diagrams ease the comprehension of the application and support its maintenance and evolution. A case study, carried out with the aim of assessing the effectiveness of the proposed tool, allowed relevant information about some real Web applications to be successfully recovered and modeled by UML diagrams.

Keywords: Web applications, Reverse Engineering, UML, Web Engineering

1. Introduction

In the Internet era, Web Applications (WA) are playing a major part for allowing the enterprises to reach a leading position in the marketplace, or to carry out their business processes more effectively. Due to the pressing market demand, WAs usually need to be developed from scratch in very short time, as well as existing WAs are modified frequently and quickly, in order to satisfy the continuously changing needs and requirements deriving from the evolving application domain.

Unlike a web site, that provides its users with just the opportunity to read information through the WWW window, a WA is an actual software system that exploits the WWW infrastructure to offer its users the opportunity to modify the status of the system and of the business it supports.

Although several methodologies have been proposed to develop WAs so far, there is a general lack of method in producing applications for the Web, and suitable development processes are not usually defined or carried out. Existing methods and techniques, with a few exceptions, do not support adequately all the activities required to engineer a WA, or they are tailored for web sites. The main consequence of this lack of support is the low quality development documentation that is generally produced. The inadequate documentation, in turn, makes it harder the application's comprehension that is needed for maintaining and evolving it, and contributes to ineffective and expensive maintenance processes.

The research community, therefore, needs to address both the problem of defining and validating development approaches suitable to produce high quality WAs, and the issue of defining and experimenting methods, techniques and tools to support the maintenance of existing applications. New and specific reverse engineering techniques and tools need to be defined for automatically, or semi-automatically, recovering information and documentation from a WA, and for simplifying its comprehension. In order to achieve a full comprehension of a WA, a reverse engineering process should support the recovery of both the static and dynamic aspects of the applications, and suitable representation models should be used for the best rendering of this information.

In this paper, the problem of reverse engineering WAs is addressed, and a representation based on UML diagrams is adopted to depict the static, dynamic and behavioral aspects of the applications. The diagrams, that provide information at different degrees of granularity, are obtained with the support of a tool that analyzes the source code of

Proceedings of the Sixth European Conference on Software Maintenance and Reengineering (CSMR'02)
1534-5351/02 $17.00 © 2002 **IEEE**

the application, extracts and abstracts relevant information from it, and populates a repository with the recovered information. The architecture of the tool and its functions are presented in the paper.

In order to assess its effectiveness and adequacy, the tool has been used for maintaining a number of existing WAs. During the experiments, the tool supported the maintainers in the recovery of the WA UML class diagrams, sequence diagrams, and use case diagrams. The results of the experiments and lessons learned are presented in the paper, whose remaining part is organized as follows. After a discussion on related works presented in section 2, section 3 proposes an approach for reverse engineering WAs and a suite of models that depict the static and dynamic aspects of a WA. Section 4 provides the description of the proposed tool, while in section 5 a case study carried out with the support of the tool is discussed. Some conclusive remarks are given in section 6.

2. Related Works

In the last years, several methodologies and tools for the development of WAs have been proposed. Some methodologies derive from the area of hypermedia applications, such as RMM [12, 13] and OOHDM [21, 22], while other ones, such as the one proposed by Gnaho and Larcher [11], have been developed specifically for WAs. Other methodologies for engineering WAs are illustrated in [14, 15, 16], while Cloyd [4] describes a user-centered design process for WA, and Beker [2] proposes a development model, based on web site usability, to promote user satisfaction. The W3I3 project defined a methodology and a notation language, the Web Modeling Language (WebML), for specifying complex Web sites and applications at the conceptual level and along more dimensions [1, 5]. Some extensions to the UML notation have been proposed by Conallen [6, 7, 8] to make the UML suitable to model WAs.

Most proposed methodologies focus on web sites, and support just the definition of the information content, the navigational structure, or the user interface of the application. Available methods and techniques, with a few exceptions, do not support adequately the design activities that should be responsible for producing high-level and low-level design documentation, that is typically necessary for comprehending an application. This lack of support, together with the high pressure of a very short time-to-market, are the main causes of the undisciplined WA development processes usually carried out, and of the inadequate documentation available.

On the other side, WAs need to be modified frequently and quickly during their lifetime, to cope with the necessities of an ever-changing environment. The lack of documentation impacts the effectiveness of the maintenance interventions. Maintainers usually operate without a precise model of the application architecture or a correct idea of where the business rules are coded, and they may be unable to control the potential side-effects of implementing a given change. In order to obtain an adequate comprehension of the application to be modified, maintainers undergo the necessity of recovering the needed information from the source code of the application. The complexity of such a recovery task is amplified by the variety of technologies currently used for implementing WAs, that forces the maintainer to dominate a wide technological landscape.

Reverse engineering techniques and tools may be used to support the automatic, or semi-automatic, recovery of information and documentation from WAs. However, the literature provides a few contributions tackling this topic. Some of them address the extraction and analysis of the architecture of a web site. Chung and Lee [3] for instance propose an approach for reverse engineering web sites and adopt the Conallen's UML extensions for describing their architecture. According to their approach, each page of the web site is associated with a component in the *component diagram*, while the web site directory structure is directly mapped into *package diagrams.*

Ricca and Tonella [17, 18, 19] propose the *ReWeb* tool to perform several traditional source code analyses over Web sites: they use a graphical representation of web sites and introduce the idea of pattern recovery over this representation. The dominance and reachability relationships are used to analyze the graphs, in order to support the maintenance and evolution of the web sites. The same authors in [20] have used this graphical representation, together with a UML meta-model of a WA, to drive the testing of WAs.

Schwabe et al. [23] define a framework to reuse the design of a WA, by separating application behavior concerns from navigational modeling and interface design.

The referred contributions mostly focus on web site reverse engineering, or on the recovery of the static structural aspects of a WA. The recovered information is used to obtain coarse grained representations centered around HTML pages, while the page sub-components are not usually taken into account by these representations. As a consequence, the behavior of a WA, that is due to the dynamic interactions among all its components, cannot be recovered and adequately documented even by reverse engineering processes.

3. Reverse engineering web applications

The comprehension of a WA can be achieved through

reverse engineering processes that allow more abstract representations of the application to be reconstructed from its artifacts. These representations should describe the structural view and the behavioral view of the application, and support the mapping of structural components into the behavior they implement. Moreover, these views should provide all necessary information at different degrees of granularity, depending on the specific task to be accomplished.

A first and coarse grained structural view of a WA includes its pages, and distinguishes between pages that are deployed on a Web server, i.e., the *server pages*, and pages that a Web server actually sends back to a client requests, i.e., the *client pages*. Client pages can be classified as *static pages*, if their content is fixed and stored in a persistent way, or *dynamic pages*, if their content varies over time and is generated on the fly. Moreover, a page may be divided into *frames* using a particular page structure, the *frame-set.* The relationships among pages depicted in this view include those produced by *hypertextual links* connecting pages, or page components. Links, in turn, may be both static and dynamic. At this abstraction level, even the links between a server page and any other object allowing the connections of the application to a DBMS, or to other systems, need to be highlighted.

At a finer grain level, the structural view will have to depict the inner page components, such as *text, images, input/output form, text box, multimedia objects (sounds, movies), anchors (*implementing *hypertextual links), scripts, applet*s, and so on. Scripts and applets represent page *active components,* since they perform some processing action that contributes to the WA behavior. The relationships between page components, such as the *submit* one between a form and the server page that elaborates the form data, the *redirect* relationship between a script and a web page, or the *include* relationship between a client script and a client module will have to be represented too.

As to the behavior exhibited by a WA, this is generally achieved through collaborations and interactions between its structural components. The interactions are triggered by events deriving either from the code control flow, or from user actions. The sequences of interactions implementing each behavior will have to be identified in the code and represented with suitable models too. Examples of possible representations are reported by Systa in [24], and by Di Lucca et al. in [9, 10].

The extensions to the UML diagrams proposed by Conallen [6, 7, 8] for modeling and developing a WA provide a suite of models to depict structural, dynamic and behavioral views of the application. The *class diagram* is used to model the architecture of the application, made up of structural components and relationships among them. *Sequence diagrams* are suitable representations of the dynamic interactions between pages, page components, and the user of the application. *Use case diagrams*, finally, provide a representation of the different behaviors exhibited by the WA.

These UML diagrams can be obtained by reverse engineering existing WAs: a possible approach exploits both *static* and *dynamic analysis* of the source code.

Static analysis will not require the execution of the application: it recovers the components and the static relationships between them by parsing the code. The results of the static analysis allow a first approximation of the structural views of the WA to be obtained. Dynamic analysis vice-versa requires that the application is executed and the interactions among its components are detected and recorded during the execution. Dynamic analysis provides further details for the structural views, and supports the recovery of the dynamic and behavioral model of the WA.

Tools for automatically, or semi-automatically, extracting and abstracting information from the WA are able to simplify the execution of the reverse engineering process. The tool we propose to support the recovery of several UML diagrams from a WA source code is described in the following section.

4. The Web Application Reverse Engineering tool

UML diagrams describing the different views of a WA can be recovered with the support of the reverse engineering tool WARE (Web Application Reverse Engineering), that automatically extracts information from the application and allows more abstract representations to be reconstructed.

WARE is an integrated tool whose architecture comprises the following main components:

(1) *Interface Layer*
(2) Service Layer
(3) Repository.

The *Interface Layer* implements a user interface providing the access to the functions offered by the tool, and the visualization of recovered information and documentation, both in textual and graphical format.

The *Service Layer* implements the tool services, and includes two main components: the *Extractor* and the *Abstractor*. The former parses the WA source code and produces an *Intermediate Representation Form* (IRF), that provides a representation of the data items extracted from the WA. The *Abstractor* operates over the IRF, and implements several abstraction tasks necessary to support the recovery of the UML diagrams of the application. The

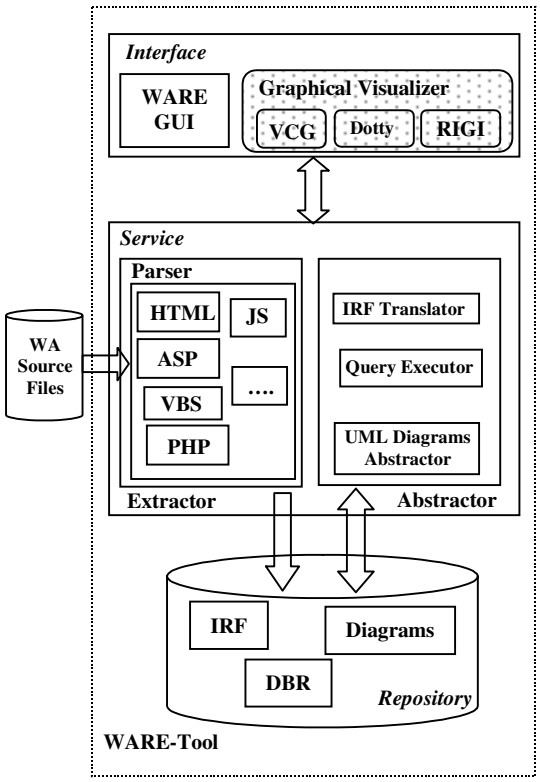

Figure 4.1: The WARE tool architecture

information produced by the Abstractor is stored in a relational database.

The *Repository* includes the IRF, the relational database populated by the Abstractor, and the recovered diagrams. Figure 4.1 shows the architecture of the tool.

4.1 The Extractor

The Extractor is a source code analyzer implemented in C++ language using the Microsoft Visual C++ development environment. Its size is of about 10,000 LOC. The Extractor parses the HTML ver. 4.0 language, and some server side and client side scripting languages. As to the client side, the current version of the tool parses the Javascript and VBScript languages, while ASP and PHP scripts can be analyzed on the server side.

The Extractor parses the source code and extracts just the information needed to identify the WA components and the relationships among them. The information recovered is stored in the Intermediate Representation Form (IRF) that includes, for each item, a description of its meaningful attributes. HTML tags and client side script instructions producing changes in pages or in DOM objects, besides function declarations and activations, are extracted by the Extractor. As far as the server side is concerned, server pages, redirection towards new pages, modules included in the pages, instructions for interfacing the database or other

```
<html>
        <form name=auth method=post
        action="auth.asp">
Login:<input name=login type=text>
Password:<input name=pwd type=text>
<input type=submit>
<input type=reset>
</form>
</html>
```

```
<OPEN>
        <FILENAME="\auth.htm">
</OPEN>
<OPEN FORM>
        <LINE=2>
        <NAME="auth">
        <METHOD="post">
        <ACTION="auth.asp">
</OPEN FORM>
<INPUT>
        <LINE=3>
        <NAME="login">
        <TYPE="text">
</INPUT>
<INPUT>
        <LINE=4>
        <NAME="pwd">
        <TYPE="text">
</INPUT>
<INPUT>
        <LINE=5>
        <TYPE="submit">
</INPUT>
<INPUT>
        <LINE=6>
        <TYPE="reset">
</INPUT>
<CLOSE FORM>
        <LINE=7>
</CLOSE FORM>
<CLOSE>
        <LINE=8>
</CLOSE>
```

Figure 4.2: An example of IRF tagged file

server objects, output on web pages, data flow from client pages, such as form parameters, and session variables, are extracted from the application.

4.2 The Intermediate Representation Form

The Intermediate Representation Form of a WA is implemented with a set of tagged files, one for each source file of the application. In the IRF files, each tag depicts a specific type of item (such as pages, page components, direct relations between the items, page parameters) and its meaningful attributes (such as code line numbers, form names, methods and actions associated with a form). IRF tags are described according to the following syntax:

<Tag Name>
[<Attribute1=value>]
[<Attribute2=value>]
[…]
[<AttributeN=value>]
</Tag Name>

As an example, Figure 4.2 shows an HTML file portion and the IRF fragment describing it. The HTML code includes a form, two input fields, and two buttons. The corresponding IRF tagged file includes eight tags, where the first one describes the HTML file name, the second one provides the description of the starting HTML form tag and its attributes, while the third and fourth ones describe the input fields, the fifth and sixth are associated with the buttons in the form, and the last two ones represent the ending HTML form tag and the closing tag of the file, respectively.

The IRF makes independent the Abstractor and the target database from the Extractor. Different Abstractors can be defined and implemented using the defined IRF.

4.3 The Abstractor

This component of WARE implements the abstraction operations that are necessary for producing more abstract views from the WA's IRF. It has been implemented in C++ language, for a global size of about 13,000 LOC and 80 classes.

The Abstractor includes three main components: a *Translator,* a *Query Executor,* and a *UML Diagram Abstractor.* The *Translator* translates the IRF into the relational database. The *Query Executor* relies on the database and executes predefined SQL queries for retrieving data about the application, such as the list of the page hyperlinks, page components, page or form parameters, functions activated in a page, and sequences of function calls or link activations.

Finally, the *UML Diagram Abstractor* component produces the class diagram of a WA at different degrees of detail (e.g., providing only client pages, or static pages, or filtering out the forms, etc.), and provides useful information that is necessary for reconstructing the sequence diagrams and the use case diagrams of the application. This information is obtained by user-customizable SQL queries.

4.4 The Repository

The WARE Repository is made up of the IRF, the relational database, the recovered diagrams and any data about the application that is produced during its analysis.

The database is implemented with the Microsoft Access DBMS. The WA is stored in the database according to the conceptual model represented by the class diagram in Figure 4.3. In the diagram, each page, page component, and any other object belonging to the WA is represented by a class, according to the Conallen's UML extensions. The Client page class and the Server page class represent two specializations of the Web Page class. The Client page class in turn can be specialized into Static page or Built page

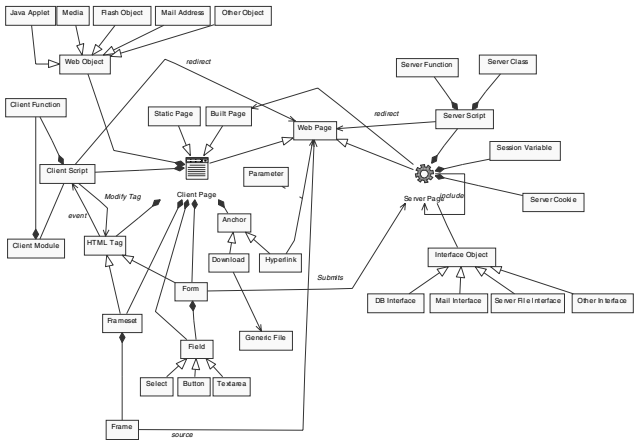

Figure 4.3: The database conceptual model of WA

classes, and is composed of objects of several classes, such as the Client Script, HTML tag, Frameset, Field, Form, and Anchor classes. Moreover, a Client page may be composed of various instances of the Web Object class, that specializes into Java Applet, Media, Flash Object, Mail address, and Other Object. Analogously, a Server page is composed of instances of the Server Script, Session Variable, Server Cookie classes, and a Server Script may comprise Server Function or Server Class objects. In addition, each type of interaction between classes is represented in the diagram by an association relationship. The main associations include the *redirect* one, that may link a Client Script to a Web Page, or a Server Script to a Built Page, the *submit* association between a Form and a Server Page, the *source* association between a Frame and a Web Page. Actually the database comprises 74 tables.

4.5 The Interface Layer

The Interface Layer provides the user interface for activating the WARE functions. The user interface has been implemented with the Microsoft Visual Basic language, and allows a simple interaction of the user with the tool.

The Figure 4.4.a shows the WARE main screen that allows the selection of different menus, each one providing a group of related functions. The first and the last menu, *File* and *Settings,* respectively provide the preliminary operations necessary to start a session of work. The *Reverse Engineering* menu activates the parsing of the application, and the translation of the IRF into the database. The *Comprehension* menu provides the access to some comprehension-related activities, such as exploring the inventory of the WA components and viewing their source code, computing the reachability relationship of a given component, or selecting a subset of components responsible for a behavior of the WA. The user can create a subset by

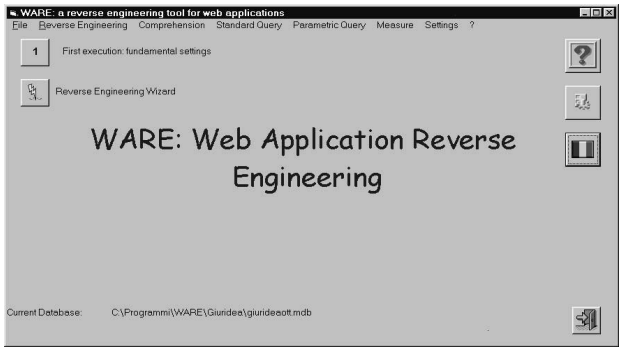

Figure 4.4-a: The WARE Main window

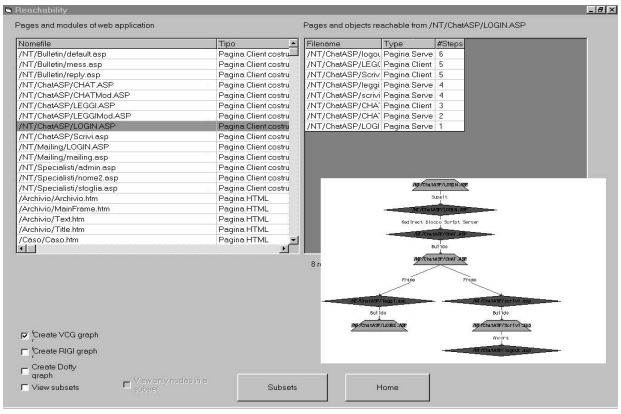

Figure 4.4 – b: WA Components reachable starting from a selected component

selecting its components from the inventory list or according to a well defined selection criterion. As an example, the tool allows the automatic identification of component subsets whose files are included in the same directory, or belonging to the same reachability set. The obtained subsets are validated by the user, associated with a description and stored in the database. Figure 4.4.b shows the components reachable starting from a selected component (i.e., the client page selected in the left box in the figure) and the graphical visualization of this subset, while Figure 4.4.c shows the forms allowing the user to create a subset by selecting the components from the inventory list.

The *Standard Query* menu provides some of the Abstractor functions, such as the production of the Conallen's class diagram of the WA and its visualization, or the list of the links included in the model, and their eventual parameters. The graphical visualization of the diagram is achieved with diagram is achieved with the support of some freeware graph displayers, such as VCG, RIGI, and Dotty. Moreover, the tool is able to show the source code associated with each item in the database, as well as to add some annotations to a selected component.

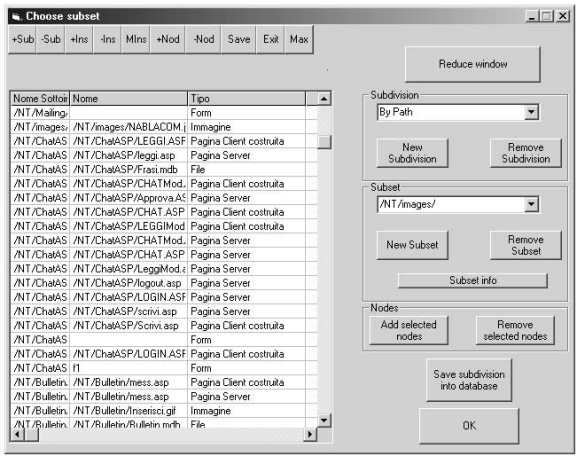

Figure 4.4 – c: Definition of sub-sets of WA components

The *Parametric Query* menu allows the user to formulate customizable queries over the database, by choosing the type of application component, link, or parameter to be searched for and displayed. Finally, the *Measure* menu provides summary measures of the WA, such as the number of WA pages, scripts, or the LOC count.

5. A Case study

In order to assess the effectiveness of the WARE tool, several experiments were carried out with the aim of collecting data about the adequacy of the functions provided by the tool, and for exploring possible scenarios of using them. An example of the experiments we carried out is shown in this section, where the results of reverse engineering a real web application are presented and discussed.

The web application used in this experiment implements a '*Juridical Laboratory*' that supports the job of professional lawyers. The application includes 201 files, distributed in 19 directories, with an overall size of about 4Mbytes. The static pages are implemented by 55 HTML files with *htm* extension contained in 10 directories, while 19 files with the *asp* extension, contained in 4 directories, implement 19 server pages. The remaining files include data or other objects, like images, logos, etc., to be displayed in the pages or downloaded by the users.

The application was not described by any documentation, and just its source code was available. The maintainers involved in the experiment, with no previous knowledge of it, were asked for using the tool for recovering the UML diagrams of the application. The static analysis of the source files was automatically performed by the tool, the IRF was generated, and the database populated. The inventory of the main application components was

obtained. Table 1 reports the list of detected components.

Table 1: The '*Juridical Laboratory*' WA components

Component type	# Detected
Server page	19
Client Static page	55
Client Built page	14
External web page	3
Client script block	53
Function in Client script block	19
Form	11
Input/ output field	71
Submit Operation (POST method)	4
Submit Operation (GET method)	7
Anchor to files to be downloaded	111
Anchor to Hypertextual link	49
Data File	61
Server script block	76
Function in Server scripts	4
Database Interface Object	29
Mail Interface Object	3
Image file	65
Redirect operation in server blocks	7

In this example, the client script blocks included in the application were all coded in Javascript language, while the server script blocks were coded in VBScript.

The class diagram describing the application was automatically generated by the tool, and the maintainers were able to obtain its graphical representation by the graph visualizers. Figure 5.1 shows the class diagram displayed by the VCG tool. The Figure depicts the class diagram at a first level of detail, where only the classes corresponding to pages and forms have been represented. Each node represents a class, and different shapes are used to distinguish the different classes: boxes are associated with static client pages, diamonds with server pages, trapezoids with dynamically built client pages, and triangles with forms. Moreover, edges representing different relationships among classes are marked with different labels.

In order to produce a description of the behavior exhibited by the WA, the maintainers used the structural view provided by the class diagram for formulating hypotheses. Notable sub-graphs were looked for, with the aim of associating each of them with a tentative description of the function they were responsible for. This task was accomplished also by observing the execution of the WA and tracing the observed events into the classes responsible for them.

As the Figure 5.1 shows, the graph was composed of four unconnected sub-graphs: a large one, rooted in the home page of the WA, and three small ones with no static client page. The smaller sub-graphs were first analyzed: the names of the pages they included, and their execution revealed that two of them implemented functions available only on the server side and used by the web administrator to manage a mailing list and an archive of registered professional users, respectively.

The third sub-graph was composed of two pages and a form. A detailed analysis of these components revealed that a wrong maintenance intervention had produced this isolated sub-graph, by leaving out the right link reaching it. This defect was therefore fixed by inserting the right hyperlink; the components belonging to this sub-graph were correctly associated with a function description.

The large sub-graph rooted in the WA home page included elements responsible for the functions available on the client side. This sub-graph was processed too, and 9 sub-graphs were detected. The classes from each sub-graphs were analyzed and, with the support of the WA execution, the function implemented by each sub-graph was defined. The descriptions of the functions associated with the 9 sub-graphs follow:

- Home: it provides the WA home page;
- News: it provides news useful to lawyers;
- Professional's Yellow Pages: it allows the registration of a new professional, and the search for professionals in the Yellow Pages;
- Judicial Proceedings: it allows a web user to download files reporting juridical norms and papers;
- Judicial Case Studies: it presents notable juridical case studies;
- Web Searching: it allows the web user to search for juridical information in specialized juridical web sites;
- Chat: it allows registered users to chat on the net;
- About: it provides general information about the WA;
- Forum: it allows the management of a bulletin board about juridical matters.

The class diagram in Figure 5.2 shows the nine sub-graphs rounded by boxes, and labeled with the associated function. A tenth unconnected sub-graph includes the classes responsible for the mailing list management. Moreover, the Professionals' Yellow Pages box also includes the small sub-graph implementing the professional archive management on the server side. The classes belonging to each identified sub-graph were associated to a sub-set of the WA's components having the responsibility of the WA function they implemented; each identified sub-set was registered in the repository together with the description of the WA functions associated with them.

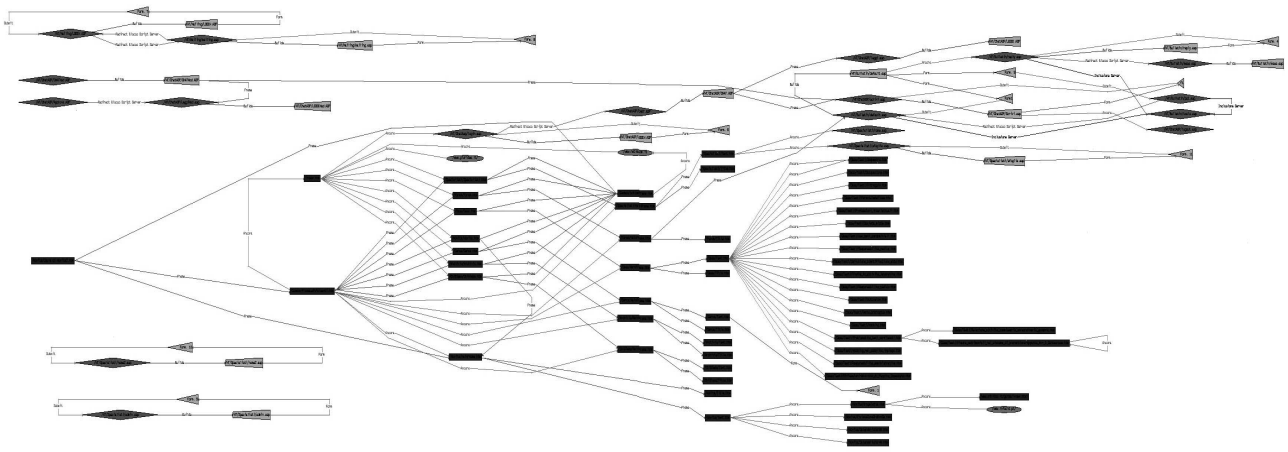

Figure 5.1: The class diagram of the WA '*Juridical Laboratory*'

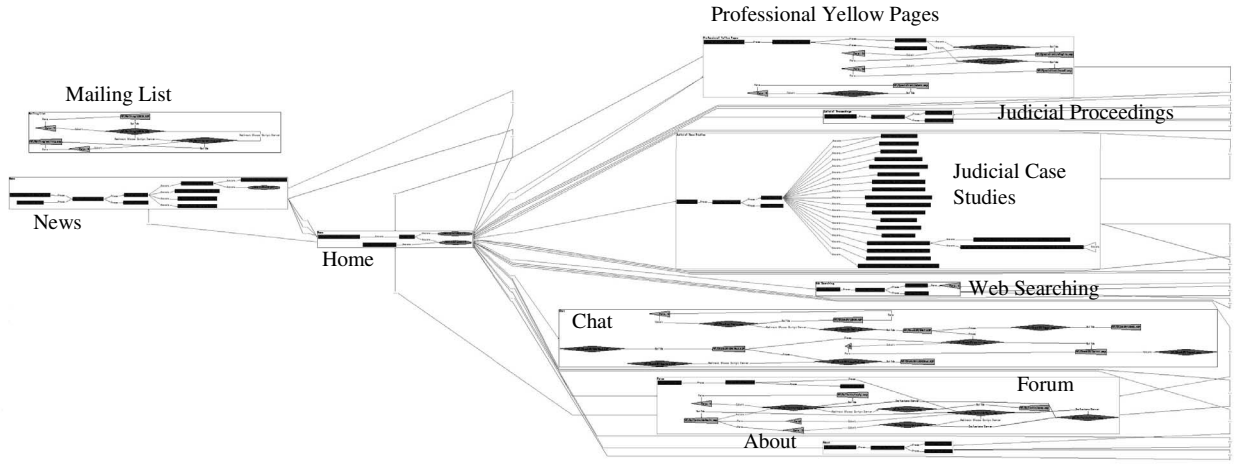

Figure 5.2: The sub-graphs identified in the class diagram of the WA '*Juridical Laboratory*'

In order to obtain a more detailed class diagram, the attributes and methods of each class had to be defined. The maintainers used the predefined and parameterized queries implemented by the tool for retrieving the sub-components of each class (such as scripts, parameters, etc.). The parameters were assumed to be the class attributes, while the scripts or the events associated to an active component (script, applet, etc.) included in the page provided the class methods.

As to the modeling of the dynamic and behavioral views of the WA, maintainers used the following approach. In order to obtain the use cases of the application, a candidate use case was defined and associated with each sub-graph from the class diagram. The scenarios describing the use cases had to be reconstructed too. Each scenario was described by a sequence diagram including the objects corresponding to the items (pages, forms, interface objects) in the sub-graph, and the interactions were deduced from the relationships among them. The database was queried to retrieve the needed information, while the time sequence of the interactions was obtained by analyzing the code by hand. The execution of the WA was also necessary for validating the recovered sequence diagrams.

Figure 5.3 shows the top level use case diagram of the WA, while Figure 5.4 reports, as an example of the recovered sequence diagrams, the sequence diagram describing the interactions among the WA components when a registered user logs in the chat service, or a new user registers himself at the chat service. The effort needed to recover the use case diagram and the top-level sequence

diagrams was of about five man hours.

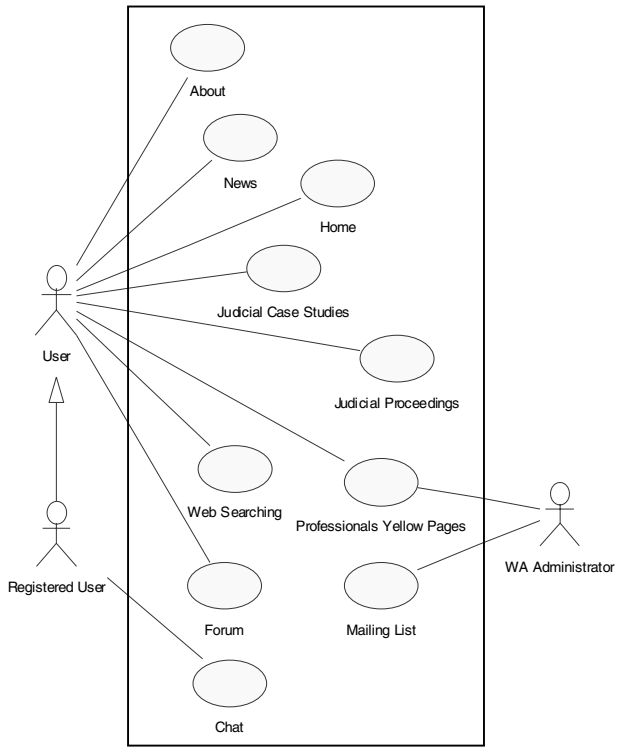

Figure 5.3: The top level use case diagram of the WA

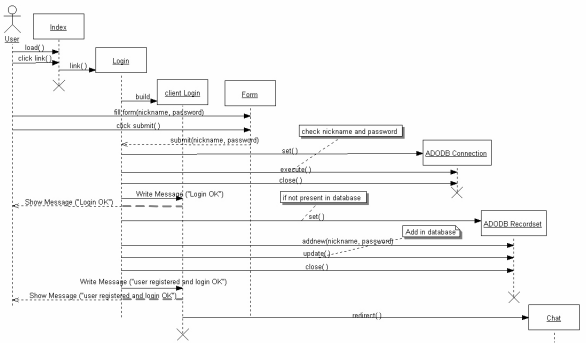

Figure 5.4: The sequence diagram for the login to the WA chat service

The UML diagrams obtained were stored in repository and, after, used to support the subsequent maintenance and evolution of the WA. During the maintenance interventions, they aided the maintainers that were able to localize the components impacted by the changes just reading the recovered documentation.

6. Conclusive remarks

The number and the economic relevance of WAs are incessantly increasing. The maintenance and evolution of WAs will become more and more a dominant task in the future. Unfortunately, WAs are usually developed without a disciplined process, and no or poor documentation is produced. This inadequacy makes hard and expensive the maintenance of existing WAs. The lack of documentation forces the maintainers to extract information about the application from its source code and, therefore, reverse engineering techniques and tools are needed to support this task.

This paper presented the reverse engineering tool WARE, whose main purpose is to provide a support to the recovery, from existing WAs, of UML diagrams dealing not only with static content, but also with the more challenging dynamic content. The tool has been submitted to validation experiments, carried out for exploring its adequacy to the needs of a maintainer of existing WAs. The experiments produced the following results.

The tool provided a precious support for the reconstruction of UML diagrams of undocumented web applications. Some of the required reverse engineering activities were automatically performed by the tool, while other activities were carried out semi-automatically, with the assistance provided by the tool. The UML diagrams obtained could be used to support the subsequent maintenance and evolution of the WAs.

We also learned some lessons from these experiments.

Some useful indications for improving the effectiveness of the reverse engineering approach supported by the tool emerged. As an example, a criterion for identifying sets of interconnected classes implementing a well defined behavior, is to explore the directory structure of the WA, since the files of recovered clusters of related classes often belonged to a same directory. Of course, as shown by other case studies, this criterion cannot be successfully applied when the WA developers did not group the related components into a single directory.

Moreover, since strongly connected components in the recovered class diagram were often involved in the same function, dominance and cluster analysis based criteria could be suggested to identify candidate sets of classes to be associated with a single function. These criteria might be more successful than the directory based one, but of course this hypothesis should be experimentally validated.

On the basis of the data collected during the experiments, an effective reverse engineering process supported the WARE tool could be defined. The process includes four main steps:

- WA static analysis and class diagram recovery;
- identification of notable sub-graphs (i.e., sets of

classes) in the class diagram, where each sub-graph (set) will be responsible for a WA functionality;
- use cases recovery, by associating each set of classes to a single use case;
- sequence diagrams recovery, for obtaining several scenarios of using the WA, by analyzing the dynamic interactions among the WA components.

Future work will be addressed to improve the tool in order to reduce its main weakness, that is the limited number of analyzed scripting languages, and the manageability of the diagram visualization.

References

[1] A. Bangio, S. Ceri, and P. Fraternali, "Web Modeling Language (WebML): a modeling language for designing Web sites", *Proceedings of the 9th International Conference on the WWW (WWW9), Amsterdam, May 2000.*

[2] S. A. Becker and F. E. Mottay, "A global perspective on web site usability", *IEEE Software, IEEE Computer Society Press, 18:54–61, Jan/Feb 2001.*

[3] S. Chung and Y. S. Lee, "Reverse software engineering with uml for web site maintenance", *1st International Conference on Web Information Systems Engineering, Hong-Kong, China, June 2001.*

[4] M. H. Cloyd, "Designing user-centered web applications in web time", *IEEE Software, IEEE Computer Society Press, 18:62–69, Jan/Feb 2001.*

[5] S. Comai, and P. Fraternali, "A semantic model for specifying data-intensive Web applications using WebML", *Web Workshop, Stanford, California, July 2001.*

[6] J. Conallen, "Modeling web applications with UML", *White paper, Conallen Inc., http://www.conallen.com/ whitepapers/webapps/ModelingWebApplications.htm, March 1999.*

[7] J. Conallen, "Modeling web application architectures with UML", *Communications of the Association for Computing Machinery, 42(10), October 1999.*

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

[9] G. A. Di Lucca, A. R. Fasolino, and U. De Carlini, "Recovering Use Case models from Object-Oriented Code: a Thread-based Approach", *Proc. of 7th Working Conference on Reverse Engineering, IEEE CS Press, Los Alamitos, CA, 2000.*

[10] G. A. Di Lucca, A.R. Fasolino, and U. De Carlini, "An algebraic notation for representing threads in object oriented software comprehension ", *Proc. of 9th IEEE Workshop on Program Comprehension, IWPC 2001, Toronto (Canada), May 2001, IEEE Comp. Soc. Press.*

[11] C. Gnaho and F. Larcher, "A user centered methodology for complex and customizable web applications engineering", *1st ICSE Workshop on Web Engineering, Los Angeles, May 1999.*

[12] T. Isakowitz, E. A. Stohr, and P. Balasubramanian, "RMM: a methodology for structured hypermedia design", *Communications of the ACM, August 1995.*

[13] T. Isakowitz, A. Kamis, and M. Koufaris, "Extending the capabilities of RMM: Russian dolls and hypertext", *Proc. of 30 th Hawaii International Conference on System Science, Maui, HI, January 1997.*

[14] D. Jones and T. Lynch, "A model for the design of web-based systems that supports adoption, appropriation and evolution", *1st ICSE Workshop on Web Engineering, Los Angeles, May 1999.*

[15] S. Murugesan, Y. Deshpande, and Hansen, "Web engineering: Beyond cs, is and se, *1st ICSE Workshop on Web Engineering, Los Angeles, May 1999.*

[16] S. Murugesan, Y. Deshpande, S. Hansen, and A. Ginlge, "Web engineering: A new discipline for development of web-based systems", *1st ICSE Workshop on Web Engineering, Los Angeles, May 1999.*

[17] F. Ricca and P. Tonella, "Visualization of web site history, *2nd International Workshop on Web Site Evolution, Zurich, Switzerland, March 2000.*

[18] F. Ricca and P. Tonella, "Web site analysis: Structure and evolution", *Proceedings of IEEE International Conference on Software Maintenance, San Jose, CA, Oct 2000.*

[19] F. Ricca and P. Tonella, "Understanding and Restructuring Web Sites with ReWeb", *IEEE Multimedia, Apr.-Jun. 2001.*

[20] F. Ricca and P. Tonella, "Analysis and Testing of Web Applications", *Proceedings of the 23rd IEEE International Conference on Software Engineering, Toronto, Ontario, Canada, May 12-19, 2001.*

[21] D. Schwabe and R. de Almeida Pontes, "A Method-based Web Application Development Environment", *Position Paper, Web Engineering Workshop, WWW8, 1999.*

[22] G. Rossi, D. Schwabe, and F. Lyardet, "Web application models are more than conceptual models", *Proceedings of the First International Workshop on Conceptual Modeling and the WWW, Paris, France, November 1999.*

[23] D. Schwabe, L. Esmeraldo, G. Rossi, and F. Lyardet, "Engineering web applications for reuse", *IEEE Multimedia, 8(1):20–31, Jan/Mar 2001.*

[24] T. Systä, "Understanding the Behavior of Java Programs", *Proc. of 7th Working Conference on Reverse Engineering, IEEE CS Press, Los Alamitos, CA, 2000, pp.214 – 223.*