An Approach for Reverse Engineering of Web-Based Applications

G.A. Di Lucca**, M. Di Penta*, G. Antoniol*, G. Casazza**
dilucca@unina.it dipenta@unisannio.it antoniol@ieee.org gec@unisannio.it

(*)University of Sannio, Faculty of Engineering - Piazza Roma, I-82100 Benevento, Italy
(**)University of Naples "Federico II", Dip. Informatica e Sistemistica
Via Claudio 21, I-80125 Naples, Italy

Abstract

The new possibilities offered by WEB applications are pervasively and radically changing several areas. WEB applications, compared to WEB sites, offer substantially greater opportunities: a WEB application provides the WEB user with a means to modify the site status. WEB applications must cope with an extremely short development/evolution life cycle. Usually, they are implemented without producing any useful documentation for subsequent maintenance and evolution, thus compromising the desired high level of flexibility, maintainability, and adaptability that is de-facto necessary to compete and survive to market shakeout.

This paper presents an approach inspired by the reverse engineering arena and a tool prototype supporting WEB application reverse engineering activities, to help maintain, comprehend and evolve WEB applications. The approach defines a set of abstract views, modeled using UML diagrams, organized into a hierarchy of different abstraction levels, depicting several aspects of a WEB application to facilitate its comprehension.

A real world WEB application was used as case study, and information previously not available was recovered, with encouraging results.

Keywords: WEB applications, Reverse Engineering, UML, WEB Engineering

1. Introduction

The World Wide Web ability to ubiquitously provide and gather information, the economy globalization together with the need of new marketing strategies have tremendously boosted the development of WEB applications (WA): software applications backboned on the WWW infrastructure.

The WEB boom pervasively and radically changed several areas. Information gathering/managing, commerce, software development, maintenance and evolution are just a few examples of human activities reshaped by the WEB technologies embedded in WA.

Software companies having a geographically distributed structure or geographically distributed customers are adopting WAs to communicate, share and exchange knowledge and information between different company branches and with customers. WAs are becoming the underlying engines of any e-business: business-to-customer or business-to-business applications. As clearly demonstrated by large companies (e.g., Oracle, Amazon, Yahoo, eBay), WAs represent a competitive advantage: they are critical and strategically relevant resources, not only to communicate the company image, but also to manage production and distribution.

WAs, compared to WEB sites, offer considerably greater opportunities: a WEB site may be thought of as a static site; it may sometimes display dynamic information (e.g., an access counter or the date). In contrast, a WA, usually backed up by a database, provides the WEB surfer with a means to modify the site status (e.g., by adding/updating information to the site).

WAs must cope with an extremely short development/evolution life cycle: a high level of flexibility, maintainability, and adaptability are de-facto necessary to compete and survive to market shakeouts. Unfortunately, to accomplish time-to-market, WAs are usually directly implemented without producing any useful documentation for their maintenance and evolution, and so those requirements are never satisfactorily met.

This paper proposes an approach and a tool prototype, inspired by the Reverse Engineering (RE) arena, to support the maintenance and evolution of WAs. Abstract representations proved to be valuable in the traditional software comprehension, maintenance and evolution tasks. Much in the same way, we believe that suitable representations automatically or semi-automatically extracted from existing

1095-1350/01 /$10.00 © 2001 IEEE

WAs may ease the task of WA evolution.

The approach proposes a process, clearly defining the reverse engineering activities, and is complemented by a set of views organized into a hierarchy of different abstraction levels; those views were cast into UML diagrams. Tools to cope with existing WAs and languages (client-side and server-side scripting languages) are currently under development.

The proposed RE approach deals both with new WAs developed from scratch and derived from legacy system. In either cases, these applications must evolve to cope with an ever-changing environment, but no one has the entire application picture or the idea of where business rules are coded.

The approach was applied to real world WAs, recovering information previously not available. This paper describes the approach, the preliminary tool and a case study.

The remainder of the paper is organized as follows: after a discussion on related works, an overview of a UML extension for the WEB is summarized, then the approach is presented in Section 4 and a conceptual model describing a WA is proposed in Section 5. To give concreteness, a preliminary tool set and a case study are presented in Sections 6 and 7.

2. Related Works

There are several papers in literature describing processes, methodologies and tools related to the development of a WA. Fewer contributions deal specifically with the problem of recovering information from existing WEB sites/applications.

Methodologies for engineering WAs are described in [10, 11, 12, 13], while Cloyd [6] described a process able to design WA, and Beker, Mattley [2] proposed a model to promote user satisfaction.

Some other works, similarly to this paper, focused on the extraction and analysis of the architecture of a WA. Chung and Lee [5] adopted the Conallen UML extensions [7, 8, 9] for WEB site reverse engineering, and extracted component diagrams (they considered each page as a component) and package diagrams (packages are directly mapped to WEB site directory structure). Ricca and Tonella [14, 15] proposed *ReWeb*, a tool able to perform several traditional source code analysis on WEB sites: dominance relation and reachability analysis were extended to graph representing WEB sites. Authors proposed to consider strongly connected components, and introduced the idea of pattern recovery over WEB representations.

Schobe et al. [16] proposed a framework to reuse design in WA, separating application behavior concerns from navigational modeling and interface design. Antoniol et al. [1] claimed the challenge to apply program comprehension and maintenance knowledge to WEB sites. In [1], a taxonomy of increasing complexity is discussed: at the highest levels, there are WAs encompassing databases and scripting languages; sites that are a mix between a program and a database.

Finally, some metrics-related works: Turau [17] analyzed the presence of some components, like forms, images, maps, applets, cookie in WA, analyzing differences between academic sites and commercial sites. Warren, Boldyreff and Munro [18] adapted some conventional metrics and hypertext metrics to the WEB, and developed a new set of specific WWW metrics.

Noticeably, the above papers are mostly focused on the static aspects of a WA, and mainly about a coarse grained WA representations centered around HTML pages.

Dynamic features are almost always disregarded (i.e., there is no identification of the dynamic interactions among the components of a WA) as well as a finer grained representation including pages' sub-components. However, those aspects are extremely relevant to fully comprehend a WA to maintain or evolve it.

Our contribution is geared toward site maintenance and evolution. We share with [14, 15] the idea of a tool to extract abstraction from exiting sites. However, much like [5], we adopted a representation based on UML. In contrast to [5], we propose different levels of granularity and our approach include the analysis and the representation of dynamic and behavioral aspects of a WA. We use a layered tool architecture, where a repository is used to store information about the *extraction* and *abstraction* processes. As a consequence, the user is not really tied to a specific set of queries; new queries may interactively be formulated and thus, new diagrams/views constructed.

3. Background Notions

As stated above, the number and the complexity of WAs are steadily increasing. By and large, the development effort has focused on the deployment of technologies, methodologies and tools mostly focused on the forward engineering phases, and more precisely, on the *coding*. The scripting languages proliferation, the adoption of new technologies for developing the client, e.g., Shockwave, or the introduction of XML to separate contents from presentation are just few examples of the aforementioned trend.

Other phases of the development process, and more generally, of the application life-cycle have not been deeply considered. The result is a lacking in the knowledge and in the understanding of how complex WAs should be developed, tested, documented to promote desired qualities such as maintainability, flexibility, portability, in a word to foster evolution.

J. Conallen introduced an extension and a tailorization of UML for modeling and developing WAs [7, 8, 9]. Briefly,

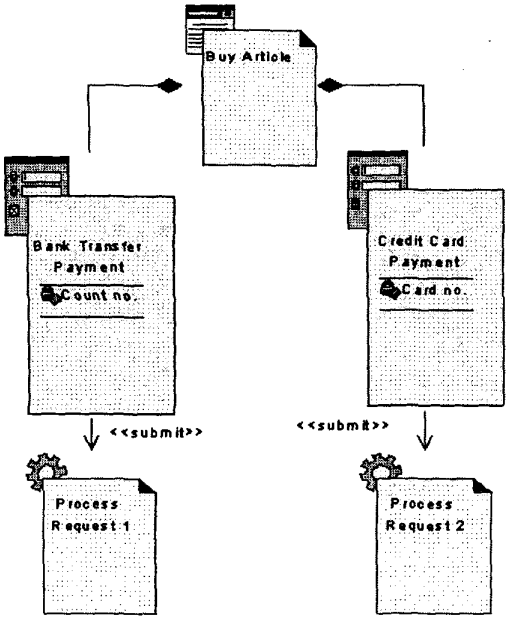

Figure 1. Composition between page and forms, and submit to server pages.

both static diagrams (use cases, class diagrams, component diagrams, deployment diagrams) and dynamic diagrams (sequence diagrams and collaboration diagrams) were extended and applied as follows:

- **Use Cases**, often developed in the first steps of the analysis, highlight the interaction of an actor with the system (the information content is the same of a traditional application development).They model the behavior of a WA from the user point of view.
- **Class diagrams** model pages and pages relations; any page is considered as a class; distinction among different kind of pages (client pages, server pages, framesets, etc.) is obtained by means of stereotypes. Page components (client and server scripts, forms, applets, etc.) are also represented by classes (and their instances by object in object diagrams). Relations model the usual notions in the usual way (e.g. a server page has an association named "build" with any client page it generates; a page may be an aggregation of forms; a radio button is a specialization of an input field). Class diagrams may be used to model the static architecture of a WA;
- **Component diagrams/package diagrams** represent relations between different resources (client pages, server pages, DBMS, libraries) composing the WA. Component diagrams Model the WA implementation architecture;
- **Deployment diagrams** focus on the configuration of different peers involved in the application, on their location and communication; they highlight where resources are located and/or executed;
- **Sequence diagrams and collaboration diagrams** highlight dynamic behavior of each use case, detailing the interaction of actors with the system. They model the interactions of WA components corresponding to the WA behavior.

The class diagram shown in Figure 1 represents composition relations between a page and two forms enclosed in it, and submit relations from the forms to server pages that process data.

4. The RE approach

Reverse Engineering (RE) processes are characterized by **goals**, **models** and **tools**. While tools aim to support the recovering process, goals and models specify the core of any RE process. Goals focus on RE motivations, they help to define a set of abstract views representing the reverse engineered application; models deal with the definition of the information to extract; models very often are complemented by intermediate representations upon which views are built [3, 4].

As in traditional software applications, WA behavior stems from static and dynamic elements, thus WA RE processes must recover:

- the static architecture;
- the dynamic interactions;
- the behavior.

Goals and models are therefore instantiated according to the above elements and consequently we propose a RE process (see Figure 2) encompassing the following phases:

1. Static Analysis;
2. Dynamic Analysis;
3. Behavioral Analysis.

The aforementioned phases recover views that can be adequately represented by extended UML diagrams (see Section 3). In particular, in this approach the following diagrams have been adopted:

- Class diagrams to represent the architecture of a WA;
- Sequence and collaboration diagrams to represent the dynamic model;
- Use Case diagrams to represent the WA behavior.

UML diagrams recovery requires as preliminary step the localization and identification of *elements* such as pages, frames, forms, scripts, (*elements* composing the application) and the relations among them. This is per-se a challenging activity in that it requires the parsing of multi-language files comprising a mixed of HTML, scripting languages, Java code etc. Moreover, to obtain independence from environments and tools, and to ensure a higher flexibility, recovered information is mapped into an intermediate representation stored in a repository.

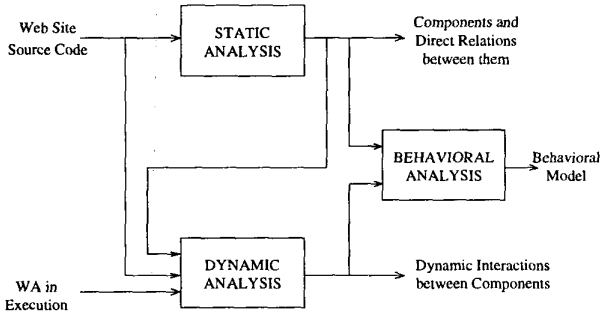

Figure 2. Reverse Engineering process of a Web Application.

4.1. Static Analysis

Static analysis does not require the execution of the application. It recovers WA architecture components and the static relations among them. HTML files, directory structure, scripting language sources as well as any other static information (e.g., database connections, use of applets/servlets) are processed. HTML pages and page sub-elements (frames, forms, widgets) composing the given page are localized, classified and recorded in an intermediate representation. Central to the RE process is the mapping between WA elements and object oriented entities, according to Conallen proposals [7, 8, 9], we made the assumption that HTML pages and relevant sub-elements (e.g., forms, script blocks, database components, etc.) are mapped into classes, while links are mapped into relations. In other words, each identified component is a candidate class, while the links between pages or page elements are translated into candidate relations. *Parameters* of each component are considered as class attributes or as new classes. *Elementary* parameters may be modeled as attributes, while other, such as database connections, may require the introduction of new classes, see [7, 8, 9] for details.

The static analysis phase may be decomposed in the following activities:

- **Inventory** (e.g., WA files, databases and more generally components);
- **Component localization;**
- **Relation recovery;**
- **Intermediate representation generation.**

Recovered intermediate representation populates a repository upon which queries and views are constructed. It is worth noting that at the end of the static analysis phase a first approximation of the WA class diagram is available. Unparsed COTS or databases may serve to generate HTML documents that are not discovered by the static analysis phase.

4.2. Dynamic Analysis

The dynamic analysis phase rely on the static analysis results. The WA is executed and dynamic interactions among the components described in the class diagram are recorded.

Dynamic analysis is performed *observing* the execution of the WA, *tracing* to source code (and, consequently, to the classes represented in the class diagram) any event or action. Traced events are those *observed* by the user or related to components external to the WA (e.g., third party databases or WEB sites). Events are the HTML pages/frames/forms visualization, the submission of forms, the processing of data, a link traversal, or a database query, etc.

All element responsible of these actions (typically links, scripts, applets) are localized and the actions given to the method of related classes. The sequences of actions fired by an event, deriving from WA code control flow (e.g., access to a database following a user form submission) or from user actions (e.g., clicking on a link or submitting a form) are associated to sequences of messages exchanged between the objects of the WA. These sequences can be represented by **sequence diagrams** (or by **collaboration diagrams**). Notice that dynamic information is also used to verify, validate and, eventually, complete the class diagram extracted by the previous phase. In other words, repository information is complemented, augmented and assessed by dynamic analysis phase.

The dynamic analysis phase may be decomposed in the following activities:

- **WA execution:** the WA is executed tracing the execution to the source code and to the class diagram.

- **Verification & Validation:** for each page displayed the class diagram has to include: a class corresponding to the page itself; a class for each component included into the page; appropriate relationships linking these classes. Hypertextual links are represented by associations among classes.
- **Detection of Interactions:** elements interacting by messages (caused by events fired from the control flow of the WA or from user actions) are detected and traced to the classes.
- **Abstraction of sequence/collaboration diagrams:** these diagrams are recovered to describe the operating scenarios.

It is worth noting that the *Detection of Interactions* step localizes those 'active' elements responsible of WA 'actions'. These components may be scripts, applets, hypertextual links, etc. Each action performed is associated with a service in the class who is responsible for that action. While static analysis may be automatically performed, dynamic analysis requires human intervention. WEB server log files may be used to extract sequence of events, that played back mimic the user interaction while the loading of a page in a browser may be recognized as an observable user event. However, at the present level of tool implementation, the WEB server was not hacked (e.g., values corresponding to form inputs are not saved) thus, for example, forms filling requires the human intervention.

4.3. Behavioral Analysis

The behavioral, or functional, analysis essentially consists in abstraction processes oriented to detect the behavior of the WA from the user point of view. The recovered behavior is described by use case diagrams. This phase may be decomposed in the following tasks:

- **Analysis of sequence/collaboration diagrams:** all interaction diagrams are analyzed to abstract functional behaviors grouped into use cases;
- **Use cases definition:** use cases, actors, uses and extends relations are defined on the basis of the functional behaviors;
- **Use case diagrams abstraction:** the use cases defined in the previous step are represented in diagrams, at different levels of details.

5. A Conceptual Model for WA

A WA conceptual model has to specify abstractions representing the application, its components and the relations between components.

At first, and coarse grain, level, WA could be thought of as composed by HTML pages. A page or a group of pages is/are responsible of a defined behavior of the WA. Pages are deployed on a WEB server; the WEB server processes client requests sending back HTML code, client scripts, applets, images, etc. What the WEB server sends to the client may or may not correspond to a physical file stored on the server: CGI bins, Servlets, server-side includes, ASP pages and related technologies and tools may generate pages on-the-fly.

More precisely the following preliminary taxonomy concerning pages was considered:

- *server pages* (i.e., pages that reside on the server) as opposed to *client pages* (i.e., the pages that are actually sent to a client);
- *static pages* as opposed to *dynamic pages*: a static page content is fixed, while a dynamic page content varies over time;
- *simple pages* as opposed to *framed pages*: a page may be divided into *frames* using a particular page, a *frameset*. A page appears in a frame specified by a *target*.
- *unlinking pages* as opposed to *linking pages*: a page, or a page component, may have *hypertextual links* to itself or to other pages. Links, in turn, may be both static and dynamic, i. e. the linked component is always the same or it is defined at run time.

A page will always be an aggregation/composition of finer grained components, such as *text, images, input/output form, text box, multimedia objects (sounds, movies), anchors, scripts, applets*, and so on. Page components (e.g., scripts or applets) may be *active components*. An active component is a component performing some processing action, for example, it may exchange data with other pages.

A page, usually a server page, may be linked to other objects allowing the connections of the WA to a DBMS, managing the data of the WA, or to other systems.

To obtain WA abstractions, it is necessary to extract, from the *source code*, all the information to identify pages, their components, the relations (both static and dynamic) existing among pages and components. WA abstractions must represent:

- The WA static architecture;
- The component dynamical interactions;
- The behaviors of the WA, assigning components to any given behavior.

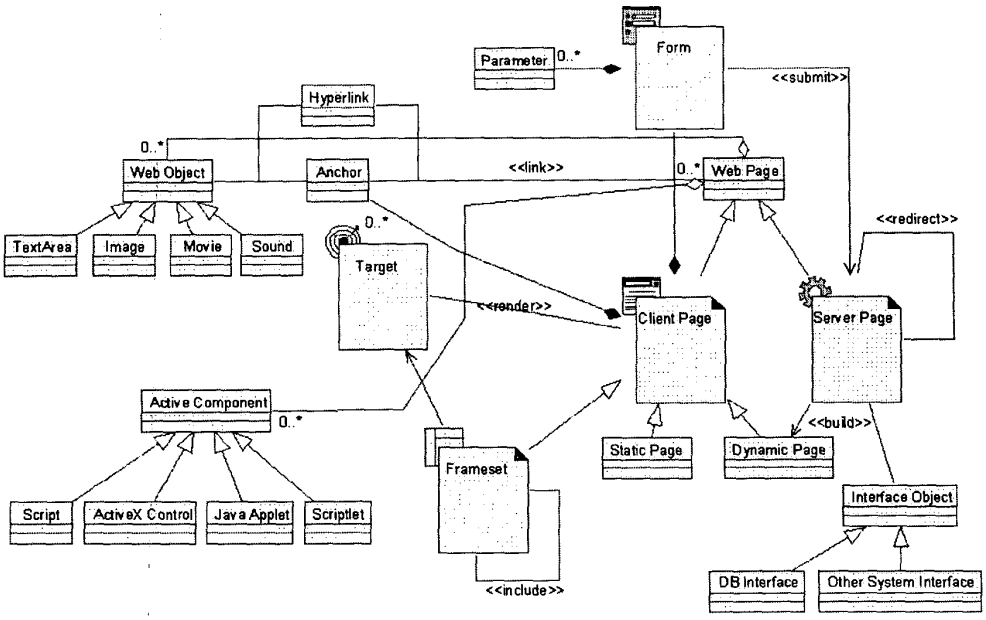

Figure 3. The Conceptual Model of a WA

The proposed RE approach focuses on a recover of components that may affect the behavior (and, thus the comprehension) of the WA; i.e. components like pages, scripts, applets, input/output forms, frames, links are recovered, while other aspects like text formats, fonts, tables are not considered.

Figure 3 shows a class diagram describing the conceptual model of the intermediate representation we used to represent a WA. Each page, as well as each component of a page, corresponds to a class, while associations describe the links among pages (or page components); composition and aggregation relations are used to describe the inclusion of page components into a page.

The representation adopted in this paper is similar to the one proposed by J. Conallen in [7, 8, 9]. Conallen proposal was focused on forward engineering, he presented a model to build WA using UML, with some extensions to better fit it to WA modeling.

While the UML extensions proposed by Conallen have been adopted, the representation proposed in this paper is focused on a RE process; it better highlights those elements that are responsible of both the behavior and the dynamic interactions of the WA. In this representation the difference between static client pages (i.e. client pages defined by static HTML files) and dynamic client pages (i.e. client pages dynamically instantiated at run-time) is considered an essential part of our taxonomy. Furthermore, differences between passive WEB objects (such as images, sounds, movies) and active WEB objects (such as scripts) are remarked, as well as the presence of interface objects (objects that interfaces the WA with DBMS or external systems).

6. Tool Support

RE tools aim to support RE activities, automatically extracting information from a WA. Recovered information produces WA abstract representations, at different level of granularity, highlighting WA objects and their relations. Abstract representations promote the comprehension of the application and of the interactions between application components, thus helping programmers during the different phases of the WA life-cycle (supporting maintenance, evolution and testing activities).

The main drawback of existing WA reverse engineering tools is that usually they do not encompass with dynamic pages modeling and, where this feature is present, peculiarities of WA are not highlighted (i.e. traditional object oriented methodologies are applied instead of the appropriate extensions described in Section 3).

The architecture of the proposed tool, as shown in Figure 4, is composed by: an extractor, an intermediate representation repository, and an abstractor. The input is a WA subject to the RE process (static and dynamic pages, images, applets, etc.); the output is a set of abstractions.

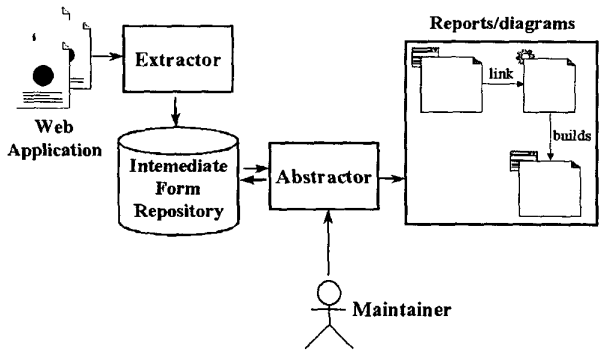

Figure 4. Architecture of the proposed tool.

- **Extractor:** It parses HTML, client-side and server-side scripting languages, and produces an intermediate representation of any useful information;
- **Intermediate form repository:** It stores the intermediate representation form obtained from the extracted information.
- **Abstractor:** Executes abstraction operations on the intermediate form, and recovers UML diagrams described in Section 3.

The tool is currently under development and implementation up to the repository is currently available; widely adopted client-side and server-side scripting languages (Javascript and VBScript) are supported. A fixed set of queries, to obtain the class diagram, was implemented; more flexible queries and views are under development as well as the parsing of other scripting and programming languages (e.g., PHP and Java).

7. Case Study

Ultimately, the issue generally comes down to a trade-off between manual intervention required to recover abstract representations and representations ability to effectively support WA life-cycle. Preliminary to any in-field evaluation of advantages/disadvantages of any RE process is the process feasibility assessment. This latter issue is the focus of the case study performed and described in the remaining part of the Section.

To evaluate WA RE process feasibility, the WEB site of the research project *LINK* (*http://serglink.ing.unisannio.it*) was used. *LINK* is a research project involving the University of Sannio, other Italian Universities and several industries. *LINK* main focus is on technology transfer by means training and a tight cooperation between project partners. *LINK* partners are spread out fairly evenly on the Italian territory, furthermore, *LINK* results must be made available to the widest possible arena. Therefore, *LINK* WEB site was conceived as a WA where project partners may exchange information. Moreover, an entire sub-section of the WA is devoted to distance learning, encompassing dynamic and static components, thus making the site suitable for experimenting the proposed RE process.

The RE process was performed following the steps described in Section 4. The WA contains 92 files, organized into 8 directories, for a total of about 500 Kbytes.

The static analysis of source code was performed automatically using the tool described in Section 6. Detected components are listed in Table 1.

Component type	# Detected
HTML page	17
Server page	41
Dynamically built web page	39
External web page (referred in some links)	9
Image file	34
Form	86
Form parameter	238
VBScript block	381
Javascript block	18
Anchor (corresponding to hypertextual links)	75
Submit operation (all by POST method)	86
Redirect operation (all in VBScript blocks)	6
Interface object (all ADODB connections)	38

Table 1. LINK WA detected components.

Components and relations were stored, using the intermediate representation form, in a repository; by querying this representation, class diagrams describing the architecture of the WA at different granularity levels were obtained.

Figure 5 shows a class diagram representing the pages of the entire WA. Associations between *classes* are due to:

- Hypertextual links;
- Submit operations;
- Build operations;
- Redirect operations.

Figure 6 shows an excerpt of a more detailed class diagram containing the forms included in the pages, the composition and submit relations due to those forms.

In the diagram, each class corresponding to a page has been named after the source file defining it. The names of forms were created appending to the keyword *Form* a number: the number corresponding to the identifier the extractor assigns to each form it discovers. UML stereotypes were used to distinguish the different types of pages and associations.

Class diagrams were analyzed and assessed by people having no previous knowledge of the WA. This activity

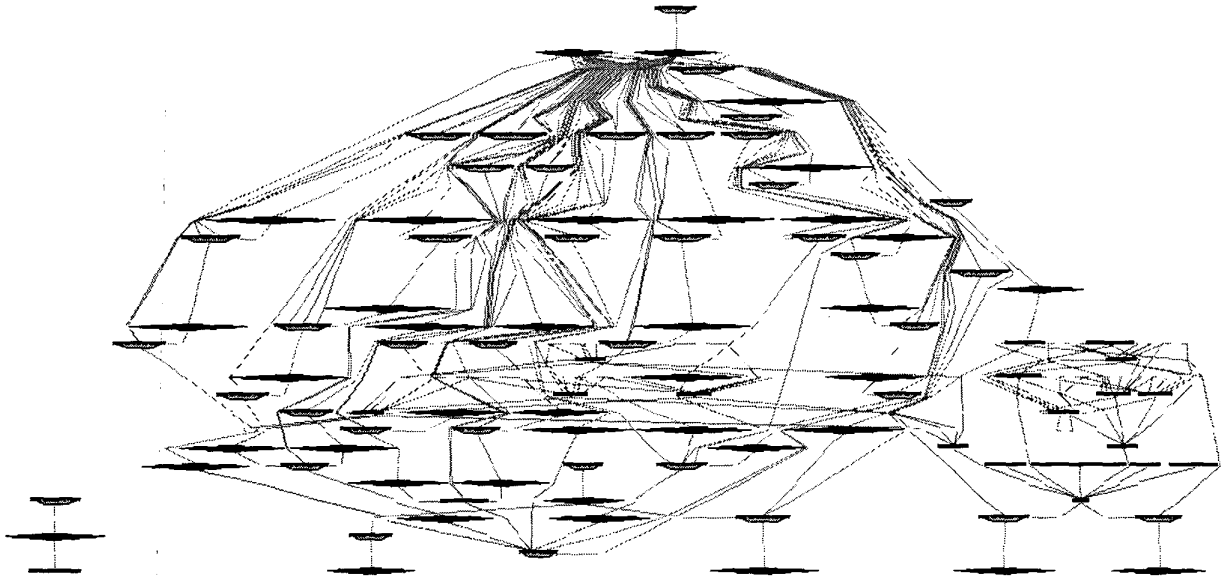

Figure 5. LINK WA class diagram

helped to formulate hypothesis about the behavior of the WA; indirect indication on the comprehension level using class diagrams was obtained in the meantime.

It is worth noting that the class diagram presented in Figure 5 is composed by two unconnected graphs. The smaller one (composed by 3 nodes) is related to the users registration; users are classified according to different access and operation privileges.

Class diagram inspection together with class names, allowed to partition classes into two sets: a set responsible to present general information about the project (only static HTML pages), and a set related to distance learning activities.

Pages related to distance learning were furtherly analyzed. Classes were grouped according to related activities:

- The login of users;
- Add and delete exercises;
- Choose and view exercises;
- Modify exercises;
- Add and delete lessons;
- Choose and view lessons;
- Add and delete curriculum.

For each group of classes, the existence of a corresponding WA functional behavior was hypothesized. ADODB objects (identified by the tool) interfacing database were inspected.

Afterwards, dynamic analysis was performed, navigating the WA, in order to validate recovered information and hypothesis formulated:

1. HTML pages related to project general information were navigated: for each page the existence of a corresponding class in the class diagram was verified (as well as for links between pages). No discrepancy between the class diagram and the real system was found.
2. A similar approach was used for the distance learning WA pages/components. In this case, actions caused by events related to user interaction (e.g., data input and submission, data processing, item selection in a list, etc.) were also verified and validated. Even in this case, no discrepancy with respect to the class diagram was found.
3. The sequences of page visualizations were observed to establish dynamic interaction among system objects, thus recovering sequence and collaboration diagrams.
4. Finally, WA use cases and actors were defined. Basically, actors represent both guest users (who visit only

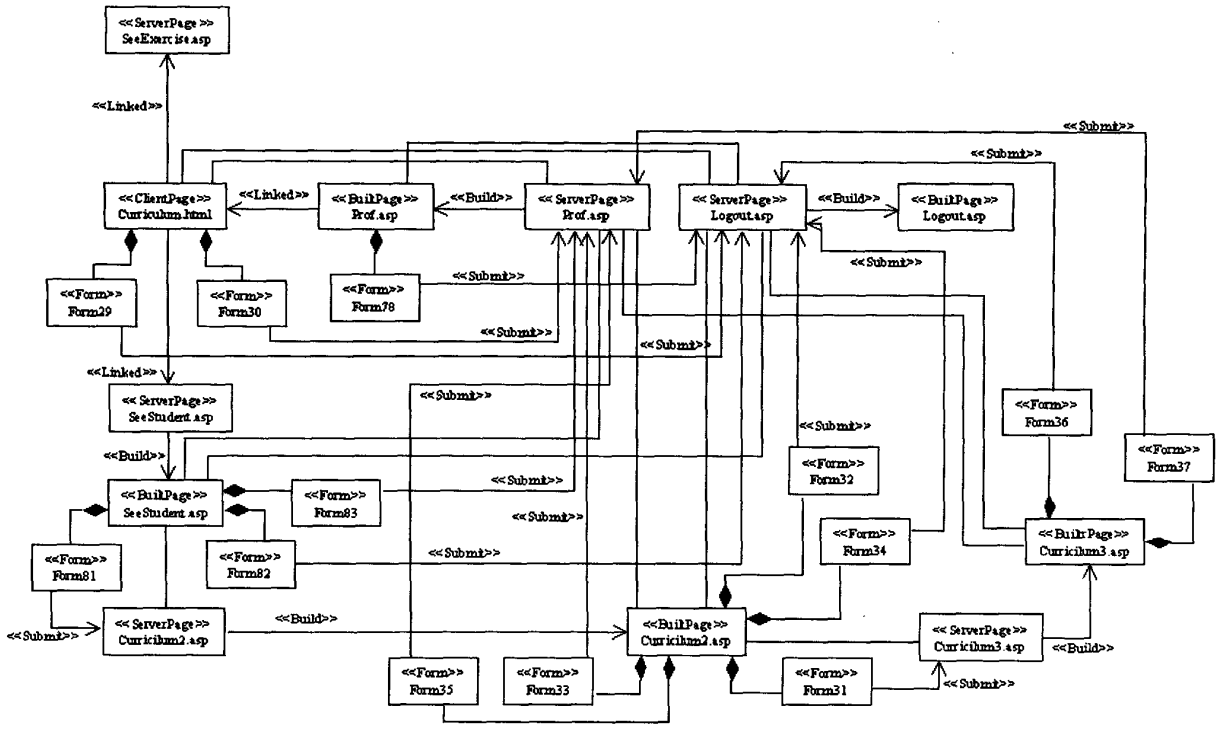

Figure 6. An excerpt of LINK WA class diagram

project general information pages and do not log-in), and authorized users (possessing a login), classified into lecturers and students. The following use cases were defined:

- Project general information visit;
- Lessons and exercises creation;
- Lessons visit;
- Exercises visit and solving.

8. Conclusions

WAs number and economic relevance are steadily increasing. It is very likely that the ratio between development and maintenance/evolution costs of traditional software application will reflect somehow in this new area. In other words, the maintenance and evolution costs of WAs are likely to constitute the most relevant life-cycle cost. In the meantime, UML is becoming a de-facto standard of the software industry and a developer *linguafranca*. For this reason, we believe that it may be convenient to represent WEB-related information by means of UML extensions.

This paper presents a RE process and a tool architecture to extract UML diagrams from existing WAs, dealing not only with static content, but also with the more challenging dynamic content. The economic relevance of the RE process needs to be validated by further studies: the paper assumes that recovering documentation, reflecting the *as-is* structure of a WA may cut down the costs related to the post-delivery WA life.

Presently, the tool set is under development and the main weakness of the available implementation is in the limited number of supported scripting languages, and in the manageability of the diagram visualization.

On the other hand, the proposed tool allows to execute customizable queries on the repository containing WA information, and to automatically generate user-defined diagrams. Indeed, using the proposed approach and the current tool, we were able to analyze real world applications, extracting information, building an intermediate form repository and constructing UML views. In this paper, we reported the main case study results. This, together with other controlled experiments carried out, gave us very good and encouraging results.

Future work will be addressed to parse other scripting

languages (e.g., PHP), and the recognize the whole set of HTML tags (e.g., text formatting tags). Further work will also be devoted to the implementation of an abstractor not only to automatically obtain UML diagrams defined in the background section but also to annotate and better visualize the diagrams, thus enhancing the knowledge about the WA.

9. Acknowledgments

The authors thank to Gerardo Canfora and Pierpaolo Gallucci for giving us the source files and any information about LINK Project Web site, and to Fabio Pace and Porfidio Tramontana for their support in tool development.

References

[1] G. Antoniol, G. Canfora, A. Cimitile, and A. De Lucia. Web sites: Files, programs or databases? In *1st International Workshop on Web Site Evolution*, Atlanta, GA, October 1999.

[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] P. Benedusi, A. Cimitile, and U. De Carlini. A reverse engineering methodology to reconstruct hierarchical data flow diagrams for software maintenance. In *Proceedings of IEEE International Conference on Software Maintenance*, pages 180–191, Miami, FL, 1989.

[4] P. Benedusi, A. Cimitile, and U. De Carlini. Reverse engineering process, document production and structure charts. *Journal of Systems and Software*, 19:225–245, 1992.

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

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

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

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

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

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

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

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

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

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

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

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

[17] V. Turau. What practices are being adopted on the web? *IEEE Computer*, 31(15):106–108, May 1998.

[18] P. Warren, C. Boldyreff, and M. Munro. The evolution of websites. In *1st International Workshop on Web Site Evolution*, Atlanta, GA, October 1999.