A. D. MCCXXIV

UNIVERSITA' DEGLI STUDI DI NAPOLI FEDERICO II

Dipartimento di Informatica e Sistemistica

Reverse Engineering of Web Applications

Porfirio Tramontana

Ph. D. Thesis

Table of contents

Table of contents **1**

PART 1: INTRODUCTION **6**

Chapter 1: Introduction **7**

1.1 Aim of the Thesis *8*

1.2 Thesis Structure *8*

Chapter 2: Background: Web Applications and Web Engineering **11**

2.1 Web Applications *11*

2.1.1 A Classification 11

2.1.2 Architecture 12

2.1.4 Technology 13

2.2 Web Engineering *14*

PART 2: MODELING WEB APPLICATIONS **16**

Chapter 3: Models describing Web Applications **17**

3.1 Web Application models *17*

3.2 Conallen's extensions to UML *22*

3.3 The proposed model *24*

PART 3: REVERSE ENGINEERING WEB APPLICATIONS **26**

Chapter 4: The WARE approach **27**

4.1 Introduction *27*

4.2 Related works *27*

4.3 Applying the Goals/Models/Tools paradigm *29*

4.3.1 Goals 29

4.3.2 Models 29

4.3.3 Tools.......29

4.4 The WARE's Reverse Engineering process *30*

4.4.1 Static Analysis.......32

4.4.2 Dynamic Analysis.......32

4.4.3 Automatic Clustering of the Web Application 33

4.4.4 Abstraction of UML diagrams.......33

Chapter 5: Web Application Clustering.......35

5.1 Introduction.......35

5.2 Background on clustering approaches for software systems.......36

5.3. A Clustering Methodology for Web Applications 37

5.3.1 Web Application Connection Graph.......38

5.3.2 Defining the coupling between Web Application components 38

5.3.3 The clustering algorithm.......41

5.4 A clustering example.......44

Chapter 6: The tool WARE.......47

6.1 Introduction.......47

6.2 Architecture of tool WARE.......47

6.2.1 WARE Service Layer.......48

6.2.2 WARE Interface Layer. 50

6.3 Analysis of a Web Application with the tool WARE.......52

Chapter 7: Experimenting the WARE Reverse Engineering Process 59

7.1 Assessing the effectiveness of the WARE Reverse Engineering process.......59

7.2 Carrying out Static Analysis 60

7.3 Carrying out Dynamic Analysis.......61

7.4 Carrying out Clustering.......62

7.5 Discussion 63

Chapter 8: Abstraction of Business Level Diagrams.......67

8.1 Introduction.......67

8.2 Recovering Business Level Class Diagrams *68*

8.2.1 Identifying candidate classes and their attributes 69

8.2.2 Associating methods to classes 72

8.2.3 Identifying relationships between classes 74

8.3. Recovering UML Use Case and Sequence diagrams *74*

8.3.1 Recovering Use Case Diagrams 74

8.3.2 Recovering Sequence Diagrams 75

8.4. A case study *76*

8.5 Future Works *82*

Chapter 9: Concept Assignment for client pages 83

9.1 Introduction *83*

9.2 A specialized conceptual model describing a Web Application *84*

9.3 Identifying the Concepts *85*

9.3.1 Separating the Control Component from the Data Component 87

9.3.2 Text Normalisation 87

9.3.3 Computation of the Concept Weights 88

9.3.4 Concept Clustering 92

9.3.5 Selecting the Concepts 94

9.4 The Web Concept Finder Tool *94*

9.5. Experimental results *96*

9.6 Future works *98*

Chapter 10: Recovering Interaction Design Patterns 99

10.1. Introduction *99*

10.2. Background *101*

10.2.1 Characterization of Patterns' features 101

10.3. The approach *104*

10.4 The supporting environment *106*

10.5. An explorative experiment *107*

10.5.1 Feature Definition 108

10.5.2 Training Phase 108

10.5.3 Preliminary Identification Test 109

10.5.4 Identification Test 111

10.6 Future works *112*

Chapter 11: Identifying Cloned Components 113

11.1 Software clones and clone analysis. *113*

11.2 Clones in Web Applications. *114*

11.3 Identifying clones *115*

11.3.1 Separation of the control component from the data component. 115

11.3.2 Clone Detection 115

11.3.3 Techniques for detecting Inner Page Clones 117

11.3.4 Clone Validation. 118

11.4. An experiment. *118*

11.5 Future Works *120*

Chapter 12: Identifying Cross Site Scripting Vulnerabilities 122

12.1 Introduction 122

12.2. Background *124*

12.2.1 Related works 124

12.2.2 Cross site scripting (XSS) 125

12.3. Detection of XSS vulnerabilities *127*

12.3.1 Assessing a server page XSS vulnerability by static analysis 127

12.3.2 Vulnerability of a Web Application 132

12.3.3 Using dynamic analysis for assessing a Web Application vulnerability 132

12.4 A case study *134*

Chapter 13: A maintainability model for Web Applications 138

13.1 Introduction *138*

13.2 A maintainability model for Web Applications *140*

13.2.1 The Oman and Hagemeister maintainability model 140

13.2.2 Traditional Systems and Web Applications 142

13.3 Adapting Oman and Hagemeister maintainability model to Web Applications............143

13.4 Case Study...148

13.5 Future Works...151

PART 4: CONCLUSIONS AND FUTURE WORKS...152

Chapter 14: Conclusions 153

14.1 Future Works...154

References...156

PART 1: INTRODUCTION

Chapter 1: Introduction

Web Applications are complex software systems providing to users access to Internet contents and services. In the last years they are having a large diffusion due to the growing of the diffusion of World Wide Web: nowadays the quantity of information and services available on the Internet is very remarkable.

Consequently, the diffusion of Web Applications in various different contexts is growing more and more, and the way business processes are carried out is changing accordingly. In the new scenario, Web Applications are becoming the underlying engine of any e-business, including e-commerce, e-government, service and data access providers. The complexity of the functions provided by a Web Application has also increased since, from the simple facility of browsing information offered by the first Web sites, a last generation Web Application offers its users a variety of functions for manipulating data, accessing databases, and carrying out a number of productive processes.

The increased complexity of the functions implemented by Web Applications is now achieved with the support of several different technologies. Web Applications generally present a complex structure consisting of heterogeneous components, including traditional and non-traditional software, interpreted scripting languages, HTML files, databases, images and other multimedia objects. A Web Application may include both ‘static’ and ‘dynamic’ software components. ‘Static’ components are stored in files, whereas ‘dynamic’ components are generated at run time on the basis of the user inputs. The Web Application components may reside on distinct computers according to a client-server or multi-tier architecture, and may be integrated by different mechanisms that generate various levels of coupling and flow of information between the components.

The high pressure of a very short time-to-market often forces the developers of a Web Application to implement the code directly, using no disciplined development process, and this may have disastrous effects on the quality and documentation of the delivered Web Application. This situation cannot be considered different from the one occurring for traditional software produced using no disciplined development process, and without respecting software engineering principles. Poor quality and inadequate documentation have to be considered the main factors underlying ineffective and expensive maintenance tasks, burdened by the impossibility of applying more structured and documentation-based approaches.

Reverse Engineering methods, techniques and tools have proved useful to support the post delivery life-cycle activities of traditional software systems, such as maintenance, evolution, and migration. The software community is now seriously addressing the problem of defining and validating similar approaches for Web Applications. Reverse Engineering allows to recover and abstract documentation from an existing Web Application, to achieve comprehension, to assess quality factors, and so on.

1.1 Aim of the Thesis

The Reverse Engineering of Web Applications is a complex problem, due to the variety of languages and technologies that are contemporary used to realize them. Indeed, the benefits that can be obtained are remarkable: the presence of documentation at different abstraction levels will help the execution of maintenance interventions, migration and reengineering processes, reducing their costs and risks and improving their effectiveness. Moreover, the assessment of the maintainability factor of a Web Application is an important support to decision making processes.

The main aim of this Thesis is the proposition and the description of a Reverse Engineering approach applicable to existing Web Applications.

In the following chapters, a Reverse Engineering approach, including several methods and techniques addressing different aims, is proposed for Web Applications. Some tools developed to automates the specific tasks to fulfil support the approach.

In particular, the approach defines:

- a reference conceptual model for Web Applications, needed to organize information extracted and abstracted;
- a Reverse Engineering process for Web Applications;
- a set of tools supporting the execution of the process;

Some experiments have been carried out to verify and validate the effectiveness of the approach.

1.2 Thesis Structure

The Thesis is subdivided in four main parts.

In the first part a general background is provided. Chapter 1 reports a discussion about the nature of the Web Applications. In Chapter 2 a taxonomy of Web Applications is reported, the most common architectures and technologies used to implement Web Applications are synthetically described and a definition of Web Engineering is specified.

The second part of the Thesis presents some of the main reference models and methods used in the development of Web Applications. Chapter 3 reports a survey of such main models and methods.

The third part of the Thesis reports the description of the proposed Reverse Engineering approach, the adopted models, the supporting tools realized and the experiments that have been carried out.

In Chapter 4 the proposed Reverse Engineering approach called WARE (Web Application Reverse Engineering) is outlined. This approach follows the Goals/Models/Tools paradigm and it defines the Reverse Engineering process needed to obtain a set of views of a Web Application at different abstraction levels. These views are represented as UML diagrams. Techniques and algorithms needed to perform this process are described in the following chapters.

In Chapter 5 the methodology defined to cluster the components of a Web Application in subsets realizing a specific user functionality is presented. This method is based on the analysis of the relationships between the pages of the Web Application. A heuristic algorithm has been defined to realize it.

Chapter 6 describes the tool called WARE. This tool has been developed to support the approach described in chapter 4, allowing the extraction of information from the source code of Web Applications, the clustering of Web Application components and the abstraction of UML models. A complete list of the functionalities of WARE is reported in Chapter 6 with some examples of the use of WARE.

Chapter 7 reports the results of the experimentation of the Reverse Engineering process, methods and tools on a number of medium sized Web Applications. A discussion of the results is presented, where some development techniques needed to improve the understandability of the application are proposed.

In Chapter 8 a method to recover UML diagrams at business level of a Web Application is defined and described. Methods and heuristic algorithms to recover UML class diagrams, use case diagrams and sequence diagrams at business level are described.

Some methods and techniques to provide automatic support to the comprehension of Web Applications in order to reduce the human effort required for the task are reported in chapters 9, 10 and 11. In particular the problem of assigning a concept to the artefacts recovered by Reverse Engineering is faced and addressed.

In Chapter 9, a method providing automatic support in the assignment of concepts to documents is presented. This method is based on both Information Retrieval principles and the text format of Web pages browsed to users. The method defines some heuristic algorithms and includes a tool realized to support it. Results of some experiments carried out to validate the method are also reported in this chapter.

In Chapter 10 a method supporting the identification of Interaction Design Patterns in the source code of Web pages browsed to users is presented. The method, the reference model, the supporting tool that have been developed and the results of some explorative experiments are reported.

In Chapter 11 a method based on clone analysis techniques with the aim to identify Web Application components with identical or similar structure is presented. Four techniques to measure the degree of similarity have been defined and described. The results of some explorative experiments are also reported.

In Chapter 12 a method is presented to assess the vulnerability of Web pages with respect to Cross Site Scripting attacks. The method, based on some secure programming rules, exploits source code analysis to verify that those rules are actually present in the code. A case study, based on a real world Web Application is reported and discussed.

Chapter 13 presents a maintainability model for Web Applications. The model is an adaptation of the one proposed by Oman and Hagemeister [Oma92] for traditional software systems. New metrics have been defined while existing metrics have been adapted to Web Applications. These metrics can be automatically evaluated from the information recovered by the Reverse Engineering approach described in the previous chapters. A case study reporting the value of these metrics for some Web Applications is discussed, with the aim to compare the maintainability of those applications.

Conclusions and future works are presented in Chapter 14.

Chapter 2: Background: Web Applications and Web Engineering

In this chapter some basic definitions are reported. In particular, a definition of Web Application by distinguishing between Web Applications and Web Sites is provided. Moreover, the main architectures and technologies typically used for implementing Web Application are described as well as a definition of Web Engineering is given.

2.1 Web Applications

A Web Application is a software product designed to be executed in the World Wide Web environment.

A Web Application can be considered as an extension of a Web Site. A Web Site is a collection of hypertextual documents, located on a web server and accessible by an Internet user.

Unlike a Web site that simply provides its users the opportunity to read information through the World Wide Web (WWW) window, a Web Application can be considered as a 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 [Con99b].

2.1.1 A Classification

A large number of taxonomies have been proposed to classify Web Application. Tilley and Huang [Til01] proposed an interesting taxonomy for Web Applications. According to this taxonomy, three classes of Web Applications with increasing complexity can be distinguished. Class 1 applications are primarily static applications implemented in HTML, and with no user-interactivity. Class 2 applications provide client-side interaction with Dynamic HTML (DHTML) pages, by associating script actions with user-generated events (such as mouse clicking or keystrokes). Finally, class 3 applications contain dynamic content, and their pages may be created on the fly, depending on the user interaction with the application. A class 3 application is characterised by a large number of employed technologies, such as Java Server Pages (JSP), Java Servlets, PHP, CGI, XML, ODBC, JDBC, or proprietary technologies such as Microsoft’s Active Server Pages (ASP).

2.1.2 Architecture

Several architectures and technologies have been proposed to implement and deploy Web Applications.

Figure 2.1 shows a general architecture for Web Applications. According to this architecture, a user interacts with a Web Browser, generating a URL request. This request is translated by the browser in a http request and it is sent to the Web Server. Web Server parses the http request and retrieves the code of the server page corresponding to the requested URL. The server page is sent to the Application Server. Application Server is an active component, usually located on the same machine of the Web Server. It interprets the code of the server page, generating as output a Built Client Page, that is a client page generated on fly and sent as response to the client. During the interpretation of a server page, Application Server can communicate with a Database server through Database Interface objects, or it can request services to a third part, such as a Web Service. Web Server sends Built Client Page to client browser, packed in an http response message. Web Browser comprehends some active plug-ins that are able to interpret code written using a client scripting language, such as Javascript code, Java applet bytecode, Flash code, and so on. If the Built Client Page has scripting code, then the result of its execution is shown to the user, else the Web browser displays directly the result of HTML rendering.

Four conceptual layers may be recognized in a Web Application:

1. Presentation layer, that is responsible of the user interface;
2. Content layer, that is responsible of the textual part of the application;
3. Application layer, that is responsible of the business logic of the application:
4. Data and Service layer, that is responsible of the data exchanges between the application and third parties, such as databases or service providers.

Often, it is difficult to separate these layers: as an example, client scripting code and server scripting code, too, may provide business logic. Moreover, at the same time, server pages may contain code related to each of the four layers. These situations must be avoided, because they make very difficult to maintain the Web Application.

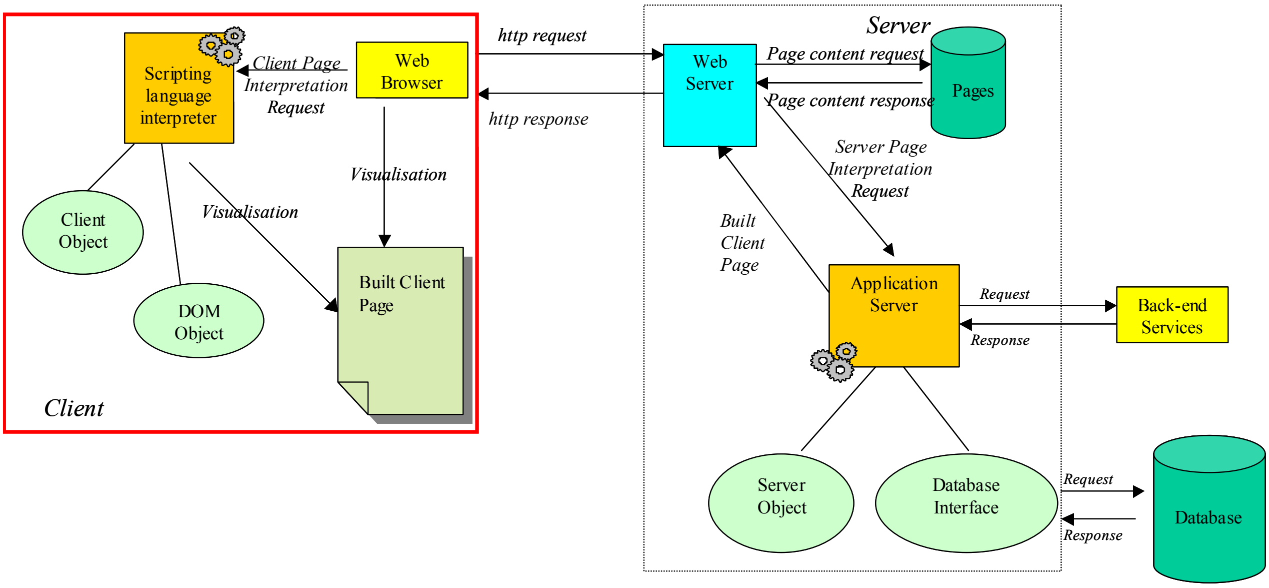


Figure 2.1: Thick Web Server Architecture for Web Applications

2.1.4 Technology

The general architecture presented in the previous subsection can be realized using different components, provided by different software houses, and the code of a Web Application may be written using different languages.

The language that is universally used to code the hypertextual content of a client page is HTML.

HTML (HyperText Markup Language) is a tagged language, defined by W3C consortium as a specialization of SGML (Standard Generalized Markup Language). A complete specification of HTML syntax and semantic can be found at [Html]. HTML language is used to code hypertextual documents and to describe guidelines for its rendering. HTML syntax is very simple, but HTML parsing isn't an easy task because HTML interpreters are syntax tolerant: there are many non-standard extensions to HTML. Moreover, incorrect HTML pages are rendered anywhere by browsers.

Client pages are usually written alternating HTML code with client script code. The most common client scripting language is Javascript, but VBScript and Jscript are sometimes used. Javascript is a simple language, with syntax similar to Java syntax, that allows executing simple elaborations, interacting with user interface object and sending http requests to a URL. Javascript provides dynamic behaviour to client pages, because their behaviour depends on the script execution, instead of the rendering of static HTML code. In particular, client script code may be used to modify properties of HTML objects, or to instantiate new objects. This characteristic makes

Web Applications flexible, but it makes them hard to analyse, because the structure of the pages may vary during execution. Client scripting languages can also handle user events.

Client script languages support functions and classes, but their scope is limited to the execution of the page. Two pages (or two different instances of a page) can share a variable using cookies. Cookies are little data structures that can contain data and information about the session of the user that wrote this data and the domain of the Web Application that wrote the cookies. Cookie values can be read only during execution of pages belonging to the same domain.

A large number of technologies have been adopted on the server side of Web Applications. Examples of server scripting languages are PHP, ASP, JSP, Perl, Python, .NET languages and so on. Each of these languages has its peculiarities, but there are some fundamental common characteristics:

- they are page-based languages: the fundamental component of the code is the page. A server page is a component comprehending a main body and a set of functions and classes. A server page can generate only an output client page, that is sent to the client at the end of the execution;
- they usually are interpreted languages. As a consequence, the scope of variables, functions and classes is limited to the page declaring them: this is a great obstacle for object oriented programming of Web Applications with scripting languages;

Server script languages are used in conjunction with a connection-less protocol, such as http. Since, it couldn’t be exchange of data between different pages or different instances of the same page. To overcome this limitation, session variables are used on the server side of Web Applications as global variables. They can be global application variables or they can be correlated to a specific session of a specific user. However, when a great amount of data have to be saved, database supports are used.

2.2 Web Engineering

Web Engineering is the discipline that studies processes, methodologies and techniques related to the Engineering of Web Applications and Sites.

Several discussions took place in the last years to establish if Web Engineering must be considered as a separate discipline or a specialization of Software Engineering. Former Web Applications were collections of hypertextual documents, so they weren’t like traditional software, because presentation aspects were predominant. In this period (when scripting languages hadn’t a great diffusion), Web Engineering was considered as a separate discipline, whereas today Web

Engineering is considered as a specialization of Software Engineering: Web Engineering processes, models and techniques are adaptations of the Software Engineering ones. A discussion about the nature of Web Engineering can be found in [Gin01].

Web Applications have some peculiarities that influence their life cycle and differentiate them from traditional application. An indicative list of these peculiarities could be the following:

- The main purpose of a Web Application usually consist in data storing and browsing;
- Web Applications are always interactive applications: usability is a fundamental quality factor for them;
- Web Applications are always concurrent applications and the number of contemporary users may vary in unpredictable way: scalability is another fundamental quality factor;
- Web Application developers are usually low-skilled people, subject to a frequent turnover;
- Many technologies doesn't encourage separation between logic layers: often peoples with different skills must work together (i.e. programmers and graphic artists);
- Web Applications need a continue evolution, for technological and marketing reasons, too;
- Web Applications must be developed in a very short time, due to the pressing short time-to-market.

This, incomplete, list of factors gives an idea of the problems related to Web Application developing. Life cycles commonly adopted for Web Applications are incremental ones, and all phases follow an iterative developing.

In the following chapter the problem of the adoption of a model describing the structure and the semantic of a Web Application is addressed.

PART 2: MODELING WEB APPLICATIONS

Chapter 3: Models describing Web Applications

The fundamental starting point to address analysis, reverse engineering, comprehension and quality assessment of Web Applications is the definition of an appropriate reference model.

In this chapter, a survey of the models proposed to describe Web Application structure and behaviour is proposed. Each of these models shows a Web Application from a particular point of view. For the purposes of this Thesis, detailed models are needed. In this chapter a model that can describe appropriately the information extracted by means of reverse engineering processes is presented.

3.1 Web Application models

During the development of Web Applications, modelling problems have been continuously faced. Isakowitz et al. [Isa97] proposed RMM (Relationship Management Methodology), that is a methodology supporting the development of Web Sites. RMM is based on RMDM (Relationship Management Data Model), that is a language to describe the application domain and the navigational mechanisms of a Web Site.

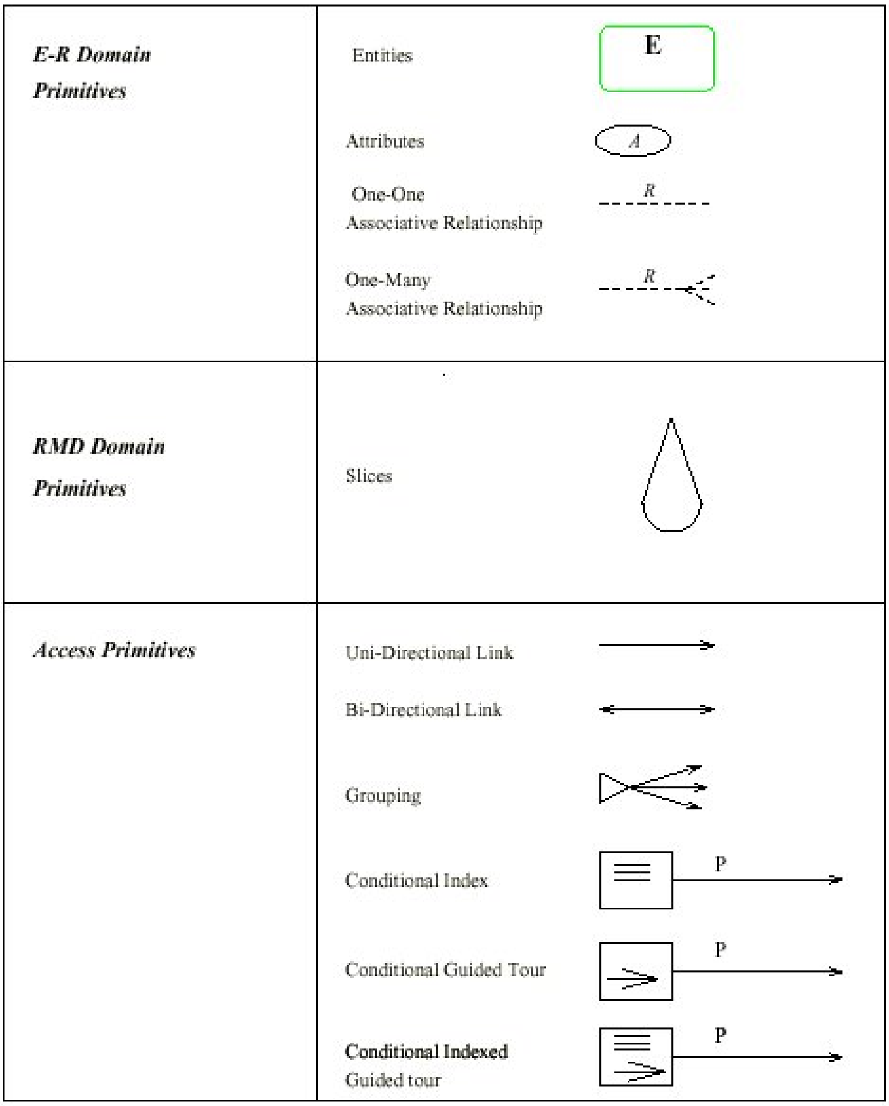
RMDM primitives and corresponding symbols are showed in Figure 3.1. There are three categories:

- Entity Relationship Domain Primitives, which are the primitives of Entity-Relationship model. They are needed to describe the informational domain of the application;
- Relationship Management Data Domain Primitives. This category comprehends the slice primitive. Slice is used to group together attributes of different entities, which is shown together to the user;
- Access Primitives, such as Links, Grouping, Conditional Index, Conditional Guided Tour, Conditional Indexed Guided Tour primitives. These primitives define the navigational context of the application.

RMM defines a seven-step design process. The steps are the following:

- E-R Design, in which a model based on E-R Domain primitives is produced;
- Slice Design, in which RMDM primitives are added to the model;
- Navigational Design, in which navigational functionalities are established;
- Conversion Protocol Design, in which abstract navigational structures are converted in pages and other real navigational structures;

- User-Interface Design, in which the graphical layout of the user interfaces is designed;
- Run-Time Behaviour Design, in which dynamic interactions are designed;
- Construction and Testing, in which the application is implemented and navigational paths are tested.



E-R Domain Primitives	Entities	E
	Attributes	A
	One-One Associative Relationship	R
	One-Many Associative Relationship	R
RMD Domain Primitives	Slices	
Access Primitives	Uni-Directional Link	
	Bi-Directional Link	
	Grouping	
	Conditional Index	P
	Conditional Guided Tour	P
	Conditional Indexed Guided tour	P

Figure 3.1: RMDM primitives

The proposed methodology is applicable only to Web Applications with navigational and information browsing purposes. So, other models have been proposed, such as OOHDM [Ros99]. OOHDM is the acronym for Object Oriented Hypermedia Design Method. It represents the first methodology that tries to match Object Oriented design and Web Application design. This methodology is structured in four phases:

- Conceptual Design;
- Navigational Design;
- Abstract Interface Design;
- Implementation.

During Conceptual design phase, classic object oriented models are produced, describing the domain of the application. During Navigational Design and Abstract Interface Design phases, the navigational view of the application is described; some specific documents and diagrams are produced, such as

- o context diagrams, which show elements needed to realize a use case,
- o specification cards, which list the characteristics of an elements,
- o navigational class schemas, which show the structure of the application from the point of view of an user.

The main limitations of this model are the following:

- great emphasis is set on the description of navigational aspects with respect to elaboration aspects;
- models and methodologies are much different from common used models and methodologies (such as UML).

Bangio et al. ([Ban00], [Com01]) proposed WebML, an XML-based modelling language for Web Applications. WebML is a language, used for specification of a data-intensive Web Applications. The main objective of WebML is the conceptual separation between data model, architectural model and implementational solutions. As a consequence of this approach, many tools have been developed to generate customized Web Applications from WebML specifications.

According to WebML approach, a Web Site has four views:

- Structural Model, that is an entity-relationship model;
- Hypertext Model, that describes the structure of the Web Site, in terms of pages and navigational relationships between them. This model comprehends two views:
    - Composition Model, that specifies pages and contents;
    - Navigation Model, that specifies navigation relationships.
- Presentation Model, that describes the layout of the pages, independently from target output device;

- Personalization Model, that specifies specialization points of customized versions of the Web Site.

Models are specified using WebML language, that is an XML-based language, which grammar has been established by the authors. WebML approach has been used in industrial context with good results, but it suffers for the same limitations of OOHDM:

- non-compliance with UML specifications;
- difficulties to adapt the approach to the description of a Web Application that isn't data intensive.

A model describing a Web Application in a more detailed way is the one proposed by Ricca and Tonella ([Ric01], [Ric01b]) and reported in Figure 3.2.

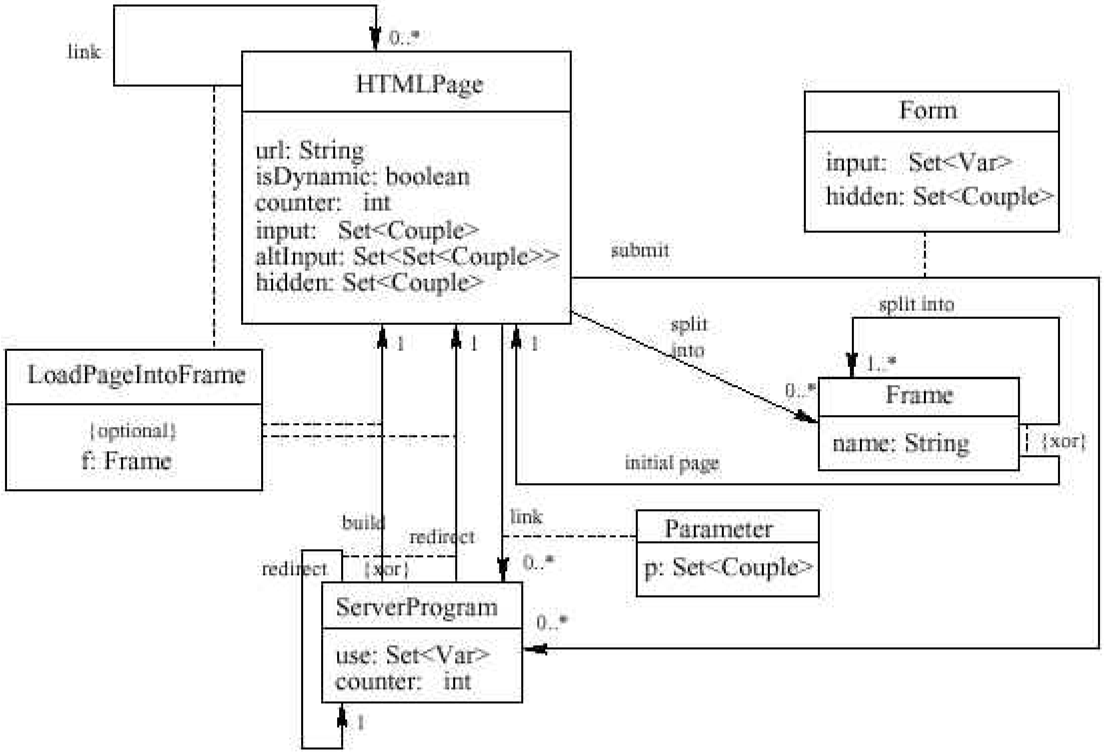


Figure 3.2: The model of a Web Application proposed by Ricca and Tonella

This model reports the main components of a Web Application: HTML pages, server programs (corresponding to server pages), frames, forms (with their input fields) and the main relationships between components, such as links, submissions, redirections and frame loading relationships.

Ricca and Tonella propose models to depict the dynamic behaviour of Web Applications, too.

Figure 3.3 reports an example of an implicit state diagram while Figure 3.4 reports an explicit state diagram.

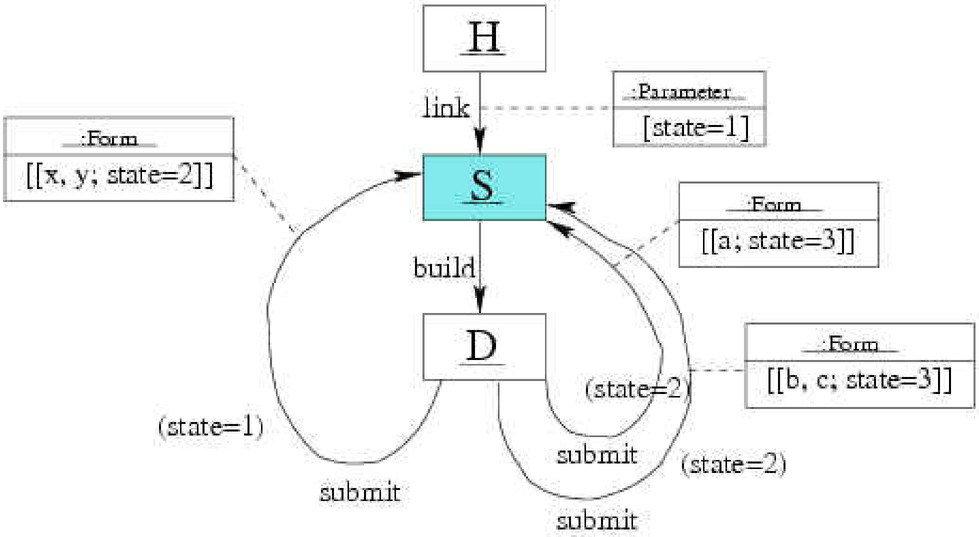


Figure 3.3: Web Application models proposed by Ricca and Tonella: implicit state diagrams

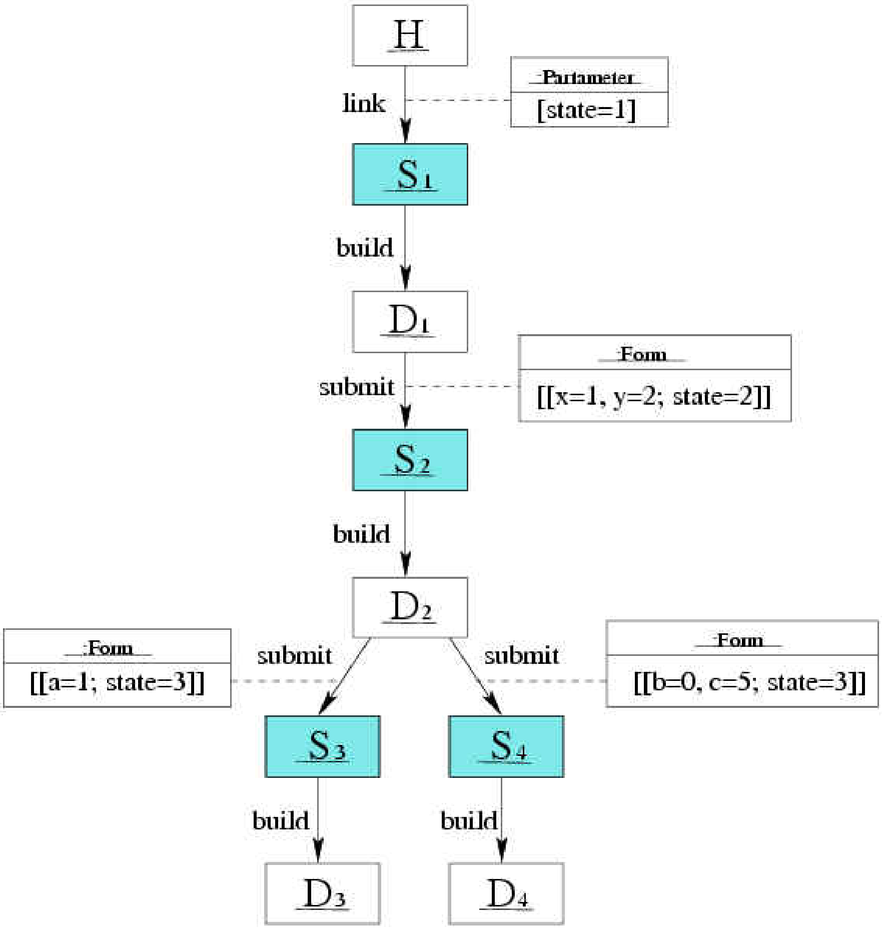


Figure 3.4: Web Application models proposed by Ricca and Tonella: explicit state diagrams

State diagrams proposed by Ricca and Tonella are similar to UML statechart diagrams. State diagrams have nodes corresponding to the entities of the model and transitions corresponding to relationships of the model. In the explicit diagram, an entity can be represented by more than one node. As an example, figure 3.3 reports a node labelled D, representing the class of the pages that can be built by server page S. Vice versa, in Figure 3.4 nodes D_1, D_2, D_3, D_4 represents a set of different client pages that can be built by the server page S.

3.2 Conallen's extensions to UML

Jim Conallen [Con99] proposed a more general model to describe Web Applications. He tells that the evolution of Web Application technologies make possible to realize complex distributed application, so that a general modelling language is needed to describe structure and behaviour of Web Applications. So, he proposed to adopt Unified Modeling Language (UML) to describe Web Applications, too. Classical UML extension mechanisms, such as stereotypes, tagged values, constraints and decorators (such as icons) have been used to take into account Web Application peculiarities.

Figure 3.5 reports an excerpt of the model proposed by Conallen. This model is more detailed than the ones previously described, and it comprehends server and client elements, too.

Conallen's extensions to UML yield to reduce the semantic distance between UML and Web Applications. Conallen's extensions are used to trace design diagrams at a detail level of a Web Application in a UML compliant way. According to Conallen, high-level design phases of a Web Application are similar to those of a traditional application, and the same UML models can be used.

Class diagrams can describe static structural views of the application. In a class diagram server pages and static client pages are depicted as static classes (classes with only one possible object). These classes are characterized by stereotypes <<*Server page*>> and <<*Client page*>> or by an appropriate icon. Local variables of these pages are depicted as private attributes of the respective classes, while functions are depicted as private methods. These classes can have neither protected, nor public attributes nor method.

Server pages can generate as output a client page that is sent to the client. This page isn't a static object: it isn't stored anywhere. These pages are depicted as classes with the stereotype <<*Built Client Pages*>>. They are specializations of client pages and it can have attributes and methods in the same way of static client pages. Another fundamental stereotyped class is the Form. A Form is a structure collecting input data. It is aggregated to a client page.

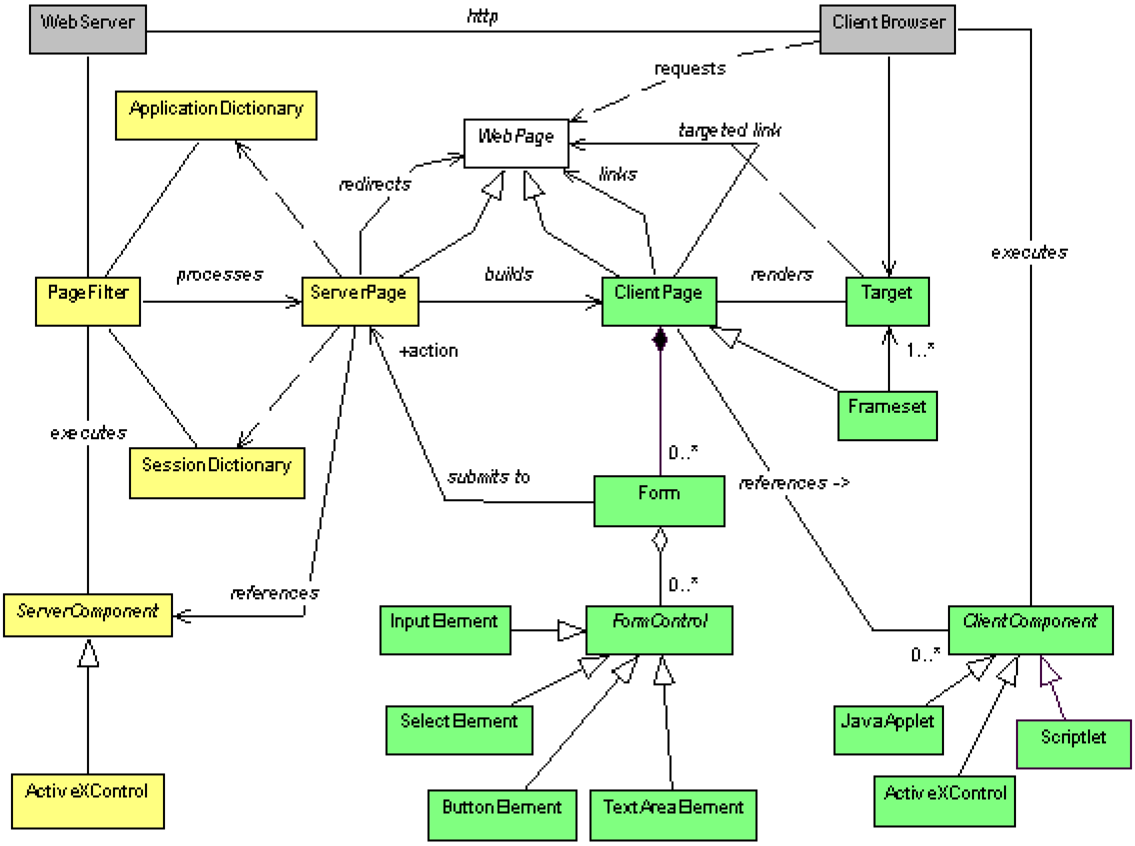


Figure 3.5: Conallen's model

Relationships between stereotyped classes are also stereotyped. Conallen's model lists the following categories of relationships, which are depicted as generic association:

- <<*builds*>> relationships, between a server page class and the class representing its built client pages;
- <<*link*>> relationships, between a client page and another page;
- <<*submit*>> relationships, between a form and the server page to which data are sent;
- <<*include*>> relationships, between two pages or a page and a library module.

Conallen's extensions can be used to depict the static architecture of the Web Applications (by means of UML class diagrams) but also the dynamic behaviour (by means of UML sequence, collaboration, activity, statechart diagrams).

Adoption of Conallen's model presents the following advantages:

- Conallen's model can describe Web Sites and Web Applications, too;
- Conallen's model can be also useful during detail design phase;
- Conallen's model is compliant with UML.

Conallen's model can be furtherly extended. In the following section, an extended model is presented, allowing the description of Web Application at a greater granularity level.

3.3 The proposed model

In this section a model extending the Conallen's model is described. This is the model that has been adopted in the Reverse Engineering approach described in this Thesis.

UML class diagram can be used to model the main entities of a Web Application and the relationships among them: each entity, such as a Page or a Page inner entity (like Forms or Scripts included in the Page), will correspond to a class, while associations will describe the relationships among Pages (or Page components); composition and aggregation relations are used to describe the inclusion of an entity in another entity.

Figure 3.6 shows a UML class diagram that is assumed as the reference conceptual model of a Web Application. In the diagram, each class represents one of the entities described above, while the associations have been given a name describing the semantics of the association itself. As to the composition and aggregation relationships, their multiplicity is always one-to-many, except for those cases where the multiplicity is explicitly shown in the Figure.

The main entities of a Web Application are the Web Pages, that can be distinguished as Server Pages, i.e. pages that are deployed on the Web server, and Client Pages, i.e., pages that a Web server sends back in answer to a client request. As to the Client Pages, they can be classified as Static Pages, if their content is fixed and stored in a permanent way, or Client Built Pages, if their content varies over time and is generated on-the-fly by a Server Page. A Client Page is composed of HTML Tags. A Client Page may include a Frameset, composed of one or more Frames, and in each Frame a different Web Page can be loaded. Client Pages may comprise finer grained items implementing some processing action, such as Client Scripts. A Client Page may also include other Web Objects such as Java Applets, Images and Multimedia Objects (like sounds or movies), Flash Objects, and others. A Client Script may include some Client Modules. Both Client Scripts and Client Modules may comprise Client Functions, or Client Classes. A Client Script may redirect the elaboration to another Web Page. In addition, a Client Page may be linked to another Web Page, through a hypertextual link to the Web Page URL: a link between a Client Page and a Web Page may be characterised by any Parameter that the Client Page may provide to the Web Page. A Client Page may also be associated with any Downloadable File, or it may include any Form, composed of different types of Field (such as select, button, text-area fields and others). Forms are used to collect user input and to submit the input to the Server Page, that is responsible for elaborating it. A Server

Page may be composed of any Server Script (that may include any Server Class or Server Function) implementing some processing action, which may either redirect the request to another Web Page, or dynamically build a Client Built Page providing the result of an elaboration. Finally, a Server Page may include other Server Pages, and may be associated with other Interface Objects allowing connection of the Web Application to a DBMS, a File Server, a Mail server, or other systems.

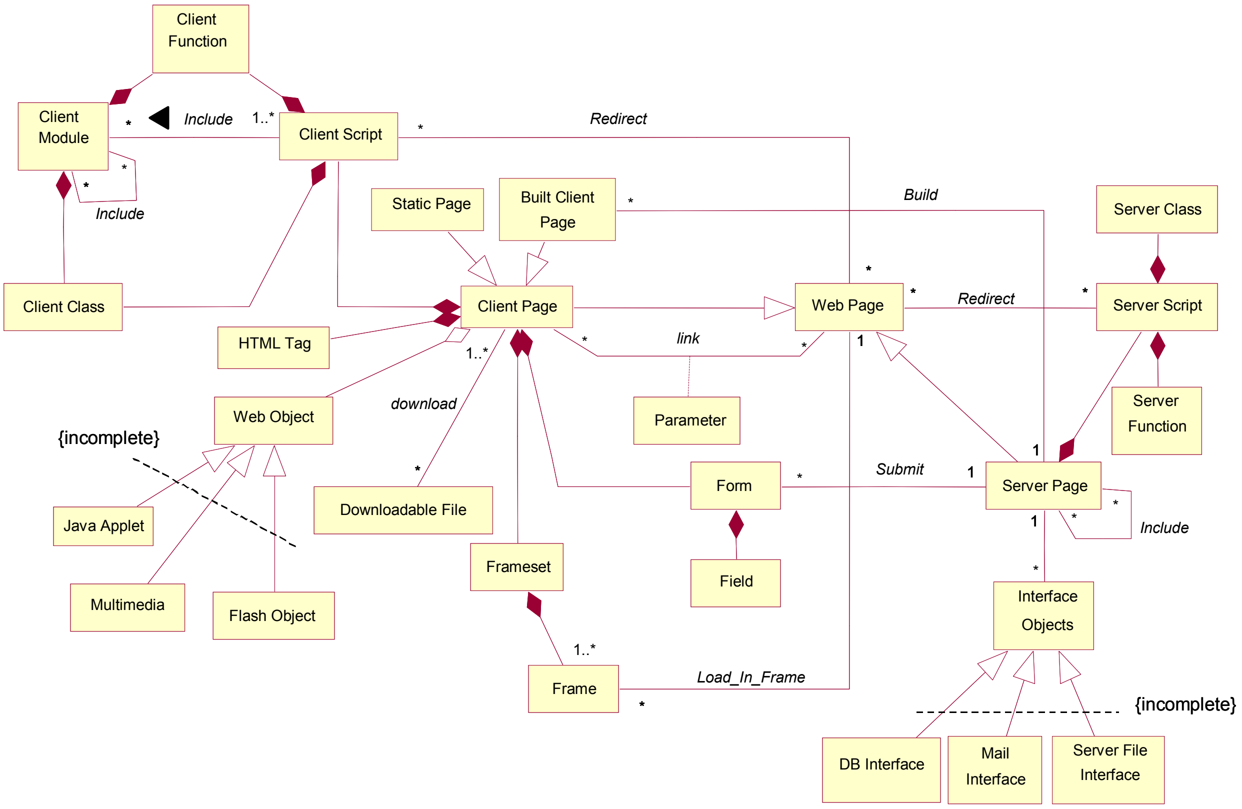


Figure 3.6: The reference model of a Web Application

This model differs from the one proposed by Conallen because it models a Web Application at a more detailed degree of granularity, allowing those application items responsible for functional behaviour to be better highlighted. Moreover, this representation explicitly shows the difference between static client pages and dynamically built client pages, as well as the difference between entities that are responsible for any processing (such as Client or Server scripts and functions) and classes of ‘passive’ objects (such as images, sounds, movies). Finally, in this model, the presence of interface objects (e.g., objects that interface the Web Application with a DBMS or other external systems) is explicitly represented, too.

Further details about this model are reported in [Dil04]. Specializations of this model are presented in the next chapters, supporting some specific Reverse Engineering methodologies.

PART 3: REVERSE ENGINEERING WEB APPLICATIONS

Chapter 4: The WARE approach

In this chapter a methodology for Reverse Engineering Web Applications based on the Goals/Models/Tools paradigm is presented. This methodology is called WARE (Web Application Reverse Engineering). The Reverse Engineering process needed to obtain a set of views of a Web Application is outlined in this chapter. The views are cast into UML diagrams. A survey of the main approaches in literature to the reverse engineering of Web Applications is presented. The methodologies adopted to realize the tasks of the process and the tool supporting them are described in the next chapters.

4.1 Introduction

The Reverse Engineering is a fundamental activity needed to improve the quality of a software system. Reverse Engineering is a set of theories, methodologies and techniques allowing the reconstruction of the documentation of an existing software system.

As declared in Chapter 1, the main aim of this Ph.D. Thesis is the proposition and the description of Reverse Engineering approaches applicable to existing Web Application.

In this chapter and in the following the problem of the Reverse Engineering of an existing Web Application is faced and a general Reverse Engineering approach, named WARE (Web Applications Reverse Engineering) is defined, described and the results of its experimentation are reported and discussed.

4.2 Related works

In this section reverse engineering techniques and tools existing in literature are briefly listed. They allow several kinds of information to be retrieved from the code of an existing Web Application, including information about its structural organization, behaviour, and quality factors. This information is usable for supporting various maintenance tasks: of course, depending on the specific task to be accomplished, the maintainer will be in charge of selecting the most suitable analysis tool and carrying out the necessary tuning activity that allows the selected tool to be correctly integrated in the maintenance process to be carried out.

The problem of analysing existing Web sites and Web Applications with the aims of maintaining, comprehending, testing them or assessing their quality has been addressed in some recent papers. New analysis approaches and tools, as well as adaptations of existing ones to the field of Web Applications, have been proposed. For example, Hassan and Holt [Has01] describe the modifications made to the Portable Bookshelf Environment (PSB), originally designed to support architectural recovery of large traditional applications, to make it suitable for the architectural recovery of a Web Application. Analogously, Martin et al. [Mar01] propose reusing the software engineering tool Rigi [Mul88] as a means of analysing and visualising the structure of Web Applications.

Other techniques and tools have been defined ad hoc for managing existing Web Applications. Chung and Lee [Chu01] propose an approach for reverse engineering Web sites and adopt Conallen's UML extensions to describe 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 ([Ric00], [Ric01]) present the ReWeb tool to perform several traditional source code analyses of Web sites: they use the graphical representation described in the previous chapter and and introduce the idea of pattern recovery over this representation. The dominance and reachability relationships are used to analyse the graphs, in order to support the maintenance and evolution of the Web sites. Schwabe et al. [Sch01] define a framework for reusing the design of a Web Application, by separating application behaviour concerns from navigational modelling and interface design. Boldyreff et al. [Bol01] propose a system that exploits traditional reverse engineering techniques to extract duplicated content and style from Web sites, in order to restructure them and improve their maintainability. Vanderdonckt et al. [Van01] describe the VAQUISTA system that allows the presentation model of a Web page to be reverse engineered in order to migrate it to another environment.

Other approaches address Web Application analysis with the aim of assessing or improving the quality of these applications. An analysis approach that allows the test model of a Web Application to be retrieved from its code and the functional testing activity to be carried out is proposed in [Dil02b]. Kirchner [Kir02] tackles the topic of accessibility of Web sites to people with disabilities, and presents a review of some tools available for checking Web pages for accessibility. Tonella et al. [Ton02] propose techniques and algorithms supporting the restructuring of multilingual Web sites that can be used to produce maintainable and consistent multilingual Web sites. Paganelli et al. [Pag02] describe the TERESA tool, that produces a task-oriented model of a Web Application by source code static analysis, where each task represents single page functions triggered by user requests. The resulting model is suitable for assessing Web site usability, or for tracing the profile of the Web site users.

4.3 Applying the Goals/Models/Tools paradigm

A reverse engineering process is usually run to extract and abstract information and documents from existing software, and to integrate these documents and information with human knowledge and experience that cannot be automatically reconstructed from software.

According to the Goals/Models/Tools (GMT) paradigm described in [Ben89], [Ben92], a reverse engineering process is characterised by goals, models, and tools. Goals focus on the reasons for the reverse engineering and they help to define a set of views of the applications to be reverse engineered. Models provide possible representations of the information to be extracted from the code, while Tools include techniques and technologies aiming to support the information recovery process.

The Goals/Models/Tools has been adopted to define the reverse engineering process needed to analyse existing Web Applications. In this section the concepts of Goals, Models and Tools are specified for the reverse engineering of Web Applications.

4.3.1 Goals

In the field of Web Applications, possible goals of a reverse engineering process include supporting maintenance of undocumented or poorly documented applications by extracting from their code the information and documents needed to plan and design the maintenance intervention correctly. Reverse engineering processes may ease the task of comprehending an existing application, providing useful insights into its architecture, low-level design, or the final behaviour offered to its users. Moreover, a reverse engineering process may aid assessment of the characteristics of an existing application, in order to be able to evaluate its quality attributes, including reliability, security, or maintainability [Off02].

4.3.2 Models

The choice of the information to be extracted from the code and the models to be reconstructed will vary according to the specific goal to be achieved. In the previous chapter several models proposed in the literature for representing a Web Application were briefly discussed and a specific model, extending the Conallen's model, were specified.

4.3.3 Tools

The recovery of information from an existing Web Application and the production of models documenting its relevant features cannot be effectively accomplished without the support of suitable techniques and Tools that automate, or partially automate, the Web Application analysis. However, the heterogeneous and dynamic nature of components making up the application, and the lack of effective mechanisms for implementing the basic software engineering principles in Web Applications, complicate this analysis and make it necessary to address specific methodological and technological problems.

More precisely, heterogeneous software components developed with different technologies and coding languages require techniques and tools implementing multi-language analysis to be used. The existence of dynamic software components in a Web Application, such as pages created at run time depending on user input, will impose the application of dynamic analysis techniques, besides static analysis of the code, in order to obtain more precise information about the Web Application behaviour. In addition, the absence of effective mechanisms for implementing the software engineering principles of modularity, encapsulation, and separation of concerns, will make the use of suitable analysis approaches, such as clustering (cfr. chapter 6), necessary in order to localise more cohesive parts in the Web Application code.

According to the GMT paradigm, a reference approach for defining Web Application reverse engineering processes will prescribe that as a preliminary step, the goals of the process be precisely defined and hence the software views allowing these goals to be achieved be identified. After accomplishing this step, the software models representing the required views of the application have to be defined, and the techniques and tools needed for instantiating these models are selected or defined ex novo. Finally, on the basis of the models and tools identified, the sequence of activities composing the reverse engineering process, their input, output and responsibilities are precisely set out.

4.4 The WARE's Reverse Engineering process

In this section, an original reverse engineering process for Web Applications is described. This approach has been called WARE (Web Application Reverse Engineering). This approach has also been descripted in [Dil02] and [Dil04].

In the WARE approach, the GMT paradigm has been used to specify a reverse engineering process aiming to support the comprehension and maintenance of an existing, undocumented Web Application. In this case, the Goal of the process consisted of retrieving, from the source code of the Web Application, all kinds of information that could then enable the maintainers to accomplish a maintenance task more effectively. This information included the specification of all functional requirements implemented by the application (e.g., its behaviour), a description of the organization of the application in terms of its relevant entities (such as Web pages, Client or Server scripts, Forms in client pages, and other Web objects) and of their relationships and, moreover, an explicit representation of the traceability relationship that enables simplified localisation of the set of software entities that collaborate to implement the functional requirements of the application. The information extracted are those presented in the model described in the previous chapter.

After defining the Goal, software models offering a suitable representation of the required information had to be selected. As to the behaviour of the Web Application, UML Use Case diagrams were chosen to specify the functional requirements in terms of use cases and actors. As to the description of the organisation of the relevant entities of the Web Application, UML Class Diagrams using Conallen's extensions were adopted for representing it: in such Class Diagrams, different types of Web pages and Web page entities (including scripts, forms, applets, etc.) can be distinguished by means of stereotypes, and syntactic or semantic relationships among these items can be represented by UML relationships (i.e., association, aggregation, composition and specialisation relationships).

In addition, UML Sequence Diagrams were adopted to document the dynamic interactions between Web Application items responsible for implementing the functional requirements of the application. Each Sequence Diagram has to be associated with a specific use case, and a traceability relationship is deduced between the use case and the Web Application items involved in the Sequence Diagram.

In order to complete the reverse engineering process definition, techniques and tools able to support the extraction and abstraction of the information required for reconstructing the selected models had to be identified. Techniques of static and dynamic analysis of the source code were taken into account. Finally, the specifications of the tools required to support these analyses could be defined.

The Reverse Engineering process implementing the sequence of activities and tasks necessary to obtain the selected models was defined accordingly. The process includes four steps: the first two steps are devoted to Static Analysis and Dynamic Analysis of the Web Application, respectively, the third one focuses on the Clustering of the Web Application, while the last one executes the Abstraction of UML diagrams on the basis of the information retrieved in the previous steps. The process is supported by a tool, named the tool WARE, that partially automates the execution of most of the process tasks: this tool is described in Chapter 6. Figure 4.1 illustrates the process, while additional details about each step of the process are provided below.

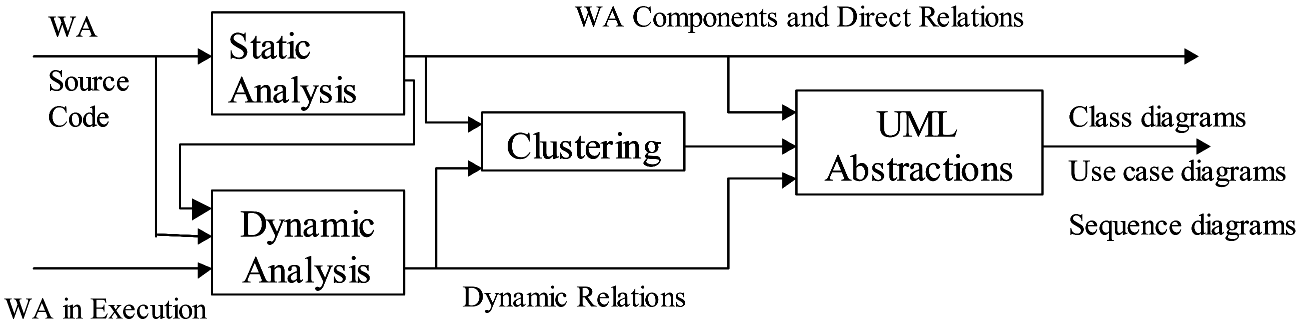


Figure 4.1: The Reverse Engineering process in the WARE approach

4.4.1 Static Analysis

In the first step of the process, the Web Application source code is statically analysed in order to instantiate the reference model of a Web Application, described in the previous section. In this phase, all the information necessary to obtain the inventory of the Web Application entities, and the static relations between them, is extracted from the code. According to the reference model adopted, Web Application pages and inner page entities, such as forms, scripts, and other Web objects, are identified, as well as the statements producing link, submit, redirect, build, and other relationships are identified and localised in the code.

This kind of analysis can be carried out with the support of multi-language code parsers, that statically analyse the code of the application, including HTML files, and scripting language sources (such as Vbscript, Javascript, ASP and PHP source code), and record the results in a suitable intermediate representation format simplifying further processing. Intermediate representation forms may be implemented by using the XML eXtensible Markup Language, or the GXL Graph Exchange Language, which enable the exchange of information derived from programs, which are conveniently represented in a graph, or by using any tagged syntax format designed to represent the necessary information efficiently.

4.4.2 Dynamic Analysis

In a dynamic Web Application, the set of entities making up the application can be significantly modified at run-time, thanks to the facility offered by script blocks, of producing new code that is enclosed in the resulting client pages, or exploiting the possibility of producing dynamic results offered by active Web objects (such as Java applets or ActiveX objects). Therefore, in the second step of the process, dynamic analysis is executed with the aim of recovering information about the

Web Application that is not obtainable by static analysis of the code. For instance, dynamic analysis is necessary to retrieve the actual content of dynamically built client pages (cfr. the class Client Built Page in Figure 1), since this content can be precisely defined only by executing the code. In addition, dynamic analysis may be indispensable for deducing links between pages, such as the ones defined at run-time by script blocks included in server pages, or by active Web objects.

The dynamic analysis phase is based on, and uses, static analysis results. The Web Application is executed and dynamic interactions among the entities described in the class diagram are recorded. Dynamic analysis is performed by observing the execution of the Web Application, and tracing any observed event or action to the corresponding source code instructions (and, consequently, to the classes represented in the class diagram).

Analysis of the execution is a task that can be carried out either automatically, on the basis of the application code, or manually, by observing the page execution by a browser and recording the observed events (i.e., results of an execution including visualization of pages/frames/forms, submission of forms, processing of data, a link traversal, or a database query, etc.) All the events must be traced to the code and all the entities responsible for these events must be identified.

The dynamically recovered information can also be used to verify, validate and, if necessary, complete the information obtained by static analysis.

4.4.3 Automatic Clustering of the Web Application

In the third step of the Reverse Engineering process the problem to group together set of components collaborating to the realization of a functionality of the Web Application is addressed.

An automatic algorithm partitioning the components of the Web Application in a set of clusters, on the basis of the information extracted during the first two steps of the Reverse Engineering process, has been defined and is described in the following chapter.

The obtained clusters are analysed by a human expert in order to identify the functionalities that they realize. This human intensive task can be partially automated. Methodology to recover, automatically, valuable information supporting this task is described in chapter 9, 10 and 11.

4.4.4 Abstraction of UML diagrams

In the final step of this reverse engineering process, UML diagrams are abstracted on the basis of the information retrieved in the previous steps.

The Class Diagram depicting the structure of the Web Application is obtained by analysing the information about the Web Application entities and relationships retrieved by static and dynamic analysis. This Class diagram is drawn as an instantiation of the conceptual model presented in the previous chapter and depicted in Figure 3.6, where each Web page and each inner page entity are represented as a class, with a stereotype describing the type of entity (e.g., static client pages will correspond to the stereotype <<Static Page>> classes, while the name of the class will correspond to the name of the page in the application). Relationships among these stereotype classes are represented, with names conforming to the ones presented in the previous chapter (such as link, build, redirect, include, etc.). Moreover, according to Conallen's notation, each class is characterised by attributes corresponding to the variables it references, and by methods corresponding to the functions and scripts included in it. Examples of the diagram recovered can be found in Chapter 6.

Sequence and collaboration diagram are also abstracted, on the basis of the static information extracted and of the information recovered by observing the execution of the application.

In Chapter 7 some examples of diagrams that have been abstracted with the described process are reported.

A more complete comprehension of the Web Applications needs the recover of UML diagrams at a greater level of abstraction, such as diagrams at business level (class diagrams, sequence diagrams and use case diagrams). These diagrams are also abstracted by means of methodologies and processes that are described in Chapter 8.

Chapter 5: Web Application Clustering

In this chapter a method to cluster the components of a Web Application in subsets realizing a specific functionality of the application is presented. This method is based on the analysis of the connections between the components of the pages. A heuristic algorithm that is described in this chapter supports the method. This algorithm has been implemented as part of the tool WARE. An example of the application of the algorithm is also presented in this chapter.

5.1 Introduction

The abstractions obtained with the Reverse Engineering approach described in the Chapter 4 are structural representations that are very useful as detailed views of the Web Application under analysis. To address global maintenance intervention, such as reengineering interventions or migrations, more abstract representations are needed. Further elaborations of the extracted information are needed to recover these diagrams.

In this chapter a method to group together components collaborating to the realization of the same functionality of the Web Application is proposed and described. In the following chapters the obtained partition of the components of the Web Application in clusters is used to abstract business level diagrams (cfr. Chapter 8). The problem of the identification of the functionality realized by a cluster needs human intervention to be solved but it can be supported by the automatic methodologies presented in Chapters 9, 10 and 11.

Clustering approaches for factoring Web Applications have been suggested in the literature. Some of them collapse the graphical representations of an application around notable graph components, such as dominators, strongly connected components, and shortest paths [Ric01]. Other approaches exploit the directory structure of the application to recover logically related components [Mar01].

A rich literature on software clustering has been produced in the past decades in the field of traditional software systems ([Bas85], [Sch91], [Man98], [Man99], [Bal01], [Tze00a], [Tze00b]). A valuable overview of cluster analysis and system remodularization is presented by Wiggerts in [Wig97]. However, explorative studies, aiming to assess the portability of the proposed approaches towards the web applications area, have not been conducted or described yet. Explorative studies should preliminarily address the following issues: the choice of a model describing the web application components adequately, the definition of a criterion establishing when a pair of components should be clustered into a cohesive unit, and the definition of a clustering algorithm to be applied.

5.2 Background on clustering approaches for software systems

A number of clustering approaches exploit source code analysis techniques, trying to cluster together files conceptually related. Anquetil et al. propose a different approach that exploits file names analysis to extract concepts about an existing application [Anq98]. This approach is grounded on the hypothesis that file names encapsulate domain knowledge, but this may not happen with software applications, or web applications, whose source code is generated by automatic tools, or by inexperienced developers that don't follow any coherent convention on file names.

Other clustering criteria are based on the assumption that logically related components are localized in the same file system directories, and analyse the physical paths of the files to discover cohesive clusters. The effectiveness of this method depends on the approach the developer used for distributing the application files in the file system. Unfortunately, web applications are not often planned with a directory organization that mirrors the functional one. Besides, many tools supporting web applications production (e.g.: Microsoft Front Page or Macromedia Dreamweaver) encourage the designer to structure it according to the nature of the files, or to their access properties (for instance, Front Page always creates a directory "images" and a directory "_private").

More promising clustering approaches seem to be those based on the analysis of graphs representing some kind of dependence between the application components. Some approaches have proven useful for comprehending or factoring traditional software systems, but a tailoring activity is needed in order to make them suitable for comprehending web applications. For instance, some approaches need to be applied to acyclic graphs (cfr. the approaches based on the dominance relationships [Hec77], [Cim95]), while the graph modelling the interconnections between web application pages is often a strongly connected graph. To make the graph acyclic, all the backward links due to hypertextual references from a page to the home page, or any index page, should be identified and removed from the graph. This may be an expensive and difficult task, since it requires analysing the semantic of every hypertextual link of the application.

A "mixed" clustering approach is that proposed in [Tze00b], which combines both pattern-driven techniques based on the identification of library modules and omnipresent modules, and an incremental clustering technique, named Orphan Adoption, for assigning the non-clustered files to some sub-system. Of course, since the pattern-driven approach focuses on common structures that frequently appear in manual decomposition of industrial software system, it should be adapted according to the common structures of web applications.

Finally, a heuristic method and a tool that treats clustering as an optimisation problem are proposed in [Man98], [Man99]. The method exploits a global measure of quality of a clustering. The space of possible partitions of a graph is explored (using genetic algorithms, or local optimum search algorithms) looking for the clustering that maximizes the quality measure.

The method has been defined with respect to a dependence graph of a traditional software application that models source code dependencies among the application files. This approach may be ported in the field of web applications, provided that an adequate model of the dependencies among Web Application components is defined.

5.3. A Clustering Methodology for Web Applications

The goal of the proposed clustering method is to group software components of a Web Application into meaningful (highly cohesive) and independent (loosely coupled) clusters. According to Anquetil et al. [Anq99], three issues must be considered to do clustering. The first issue is to build a model in which the components to be clustered are adequately described. The second one consists of defining when a set of components should be clustered into a cohesive unit, and the third issue consists of selecting the clustering algorithm to be applied.

In the approach that is proposed, as far as the description of the components is concerned, the reference Web Application model is the one described in section 3.3. As to the second choice, the coupling between components is quantified on the basis of the direct relationships between them. The more links between components, the stronger their coupling. Besides, for a finer tuning, a strategy has been established to weight the links, assuming that the coupling depends on the type of the link too.

Finally, the third choice is that of a clustering algorithm. There are many different clustering algorithms in the literature [Wig97]. Some of them have been exploited in the field of software remodularization [Sch91, Bas85], to support reverse engineering [Mul88], [Won94] or program comprehension [Tze00a, Tze00b]. A possible taxonomy distinguishes between hierarchical and non-hierarchical ones. Anquetil et al. [Anq99] experimented with several clustering algorithms, and their results show that hierarchical clustering provides as good results as other ones. A hierarchical clustering has been adopted in this approach, since it can be used to obtain different partitioning of a system at different levels of abstraction.

5.3.1 Web Application Connection Graph

The model of a Web Application provided in Chapter 3 can be analysed at a coarser degree of granularity, such as that of the web pages, or at a finer one, such as that of the inner components of the web pages.

The clustering approach proposed in this section considers the following components and relationships of a Web Application: components include web pages, server pages, client pages, framed client pages, client modules, and web objects (such as script blocks, images, applets, etc.). Relationships comprise *link*, *submit*, *redirect*, *build*, *load_in_frames*, and include ones. The model focusing these components and their relationships is provided in Figure 5.1 as a UML class diagram.

Each Web Application can be represented by an instance of this conceptual model, that is called the Web Application Connection Graph WAG = (N, E), where N is the set of the Web Application components and E is the set of edges among components. Each graph can be obtained with the support of reverse engineering tools, such as the WARE tool described in the previous chapter, that extract the needed information from the source code of the application.

5.3.2 Defining the coupling between Web Application components

The choice of a metric for expressing the degree of coupling of a pair of components is strategic for the success of a clustering algorithm. The proposed definition of coupling takes into account some intuitive criteria deriving from the knowledge and expertise in Web Application development and maintenance.

This expertise suggests that both the *typology* and the *topology* of the connections need to be taken into account for expressing the degree of coupling between components. Therefore, it is assumed that the considered relationships produce a different coupling between the connected components. In particular, some specific assumptions concerning *build*, *redirect*, *link*, and *submit* relationships are made. A *build* relationship between a server page and the built client page it produces is assumed to produce the stronger degree of coupling among the Web Application components, since the existence of the client page depends on the server page. A *redirect* relationship between two pages produces a higher degree of coupling than a *link* relationship, since a 'redirect' statement usually implicates the execution of an elaboration, by moving the control-flow from the former to the latter page. Besides, a *submit* relationship between a client page and a server page produces a higher degree of coupling than a *link* relationship, since a '*submit*' statement usually implicates the request for an elaboration and a data-flow between the pages. Moreover, due to the data-flow, a *submit* relationship are associated with a higher coupling than a *redirect* one.

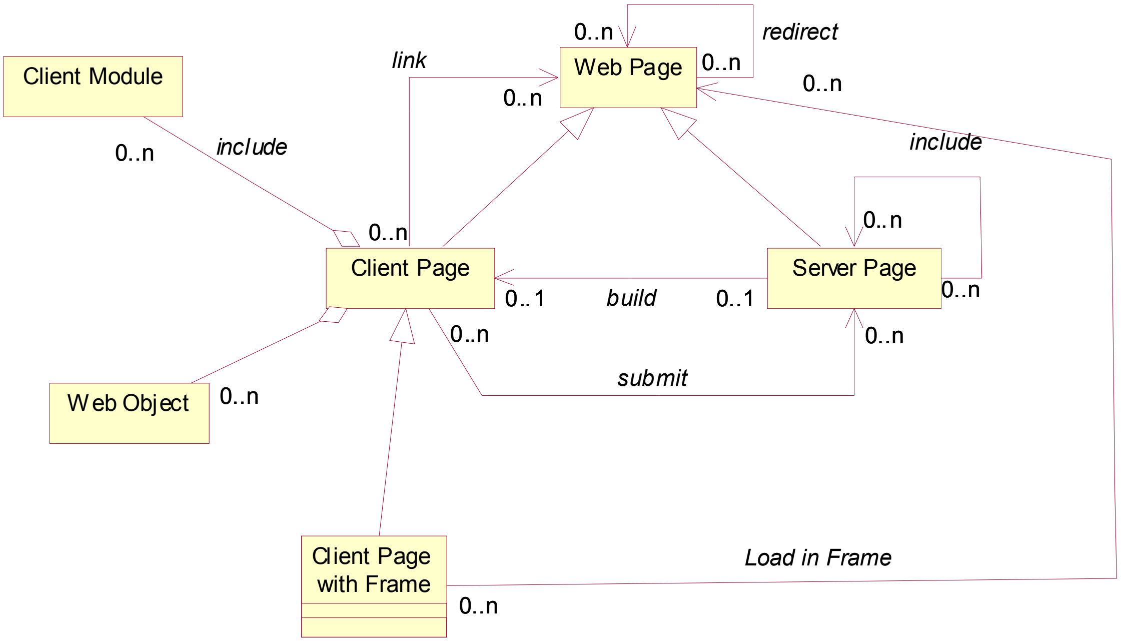


Figure 5.1: The focused Web Application conceptual model

These hypotheses on the connection typologies are taken into account by assigning *link*, *redirect* and *submit* relationships with the positive weights w_L, w_R and w_S respectively, and by assuming the following relations:

$$w_{RL} = w_R / w_L$$

$$w_{SL} = w_S / w_L$$

$$1 < w_{RL} < w_{SL}$$

The ratios w_{RL} and w_{SL} are empirically assigned on the basis of the expertise. For instance, in the experiments that have been carried out, good results have been obtained with w_L=1, w_{RL} =2.4 and w_{SL}= 3.

The degree of coupling $C_{A,B}$ of two components, namely A and B, is therefore expressed by the following metric:

$$C_{A,B} = C_{AB} + C_{BA}$$

where C_{AB} is a measure of the coupling produced by edges from A to B, and vice-versa C_{BA} is a measure of the coupling produced by edges from B to A.

The simplest way to measure C_{AB} is that of counting the weighted edges outgoing from A and going into B and, analogously, the simplest way to measure C_{BA} is of counting the weighted edges outgoing from B and reaching A. However, the coupling between the nodes A and B should be considered intuitively stronger when A uniquely reaches the node B (or B is uniquely reached from A), rather than when A reaches both B and other nodes (or B is reached both from A and from other nodes). In order to take into account these different topologies differently, an additional weighting strategy is adopted, assigning a weight w_x^{OUT} to exiting edges, and a weight w_x^{IN} to incoming edges. The w_x^{OUT} (assigned with each edge of type x exiting from the node) is defined according to the *fan-out* of that node, while the w_x^{IN} (assigned with each edge of type x coming into the node) is defined according to the *fan-in*[1] of the node.

This strategy assumes that each non-terminal node has a constant Outgoing Connection Potential set to 1. Analogously, each node with incoming edges has a constant Incoming Connection Potential (set to 1). Outgoing Potential (Incoming Potential) is distributed among outgoing (incoming) edges proportionally to the fan-out (fan-in) of the node, and proportionally to the edge weights w_L, w_R and w_S.

Given a node A, the weights $w_{LINK}^{OUT}(A)$ of each *link* edge, w_{SUBMIT}^{OUT} (A) of each *submit* edge, and $w_{REDIRECT}^{OUT}(A)$ of each *redirect* edge exiting from A can be obtained by solving the linear system shown below where N_{LINK} (A) is the number of connections of *link* type outgoing the node, $N_{SUBMIT}(A)$ is the number of connections of *submit* type outgoing the node, and $N_{REDIRECT}$ (A) is the number of connections of *redirect* type outgoing the node.

$$\begin{cases} N_{LINK}(A)\cdot w_{LINK}^{OUT}(A) + N_{SUBMIT}(A)\cdot w_{SUBMIT}^{OUT}(A) + N_{REDIRECT}(A)\cdot w_{REDIRECT}^{OUT}(A) = 1 \\ \dfrac{w_{SUBMIT}^{OUT}(A)}{w_{LINK}^{OUT}(A)} = w_{SL} \\ \dfrac{w_{REDIRECT}^{OUT}(A)}{w_{LINK}^{OUT}(A)} = w_{RL} \end{cases}$$

[1] *Fan-in* of a node is the number of edges entering the node, and *Fan-out* of a node is the number of edges leaving the node.

The weights w_{LINK}^{IN} (B) of each *link* edge, w_{SUBMIT}^{IN} (B) of each *submit* edge, and $w_{REDIRECT}^{IN}$ (B) of each *redirect* edge coming into a given node B can be obtained by solving a similar linear system.

Finally, the degree of coupling $C_{A,B}$ of two components are expressed as follows:

$$C_{A,B}= C_{AB} + C_{BA}= p_{A \rightarrow B} * p_{B \leftarrow A} + p_{B \rightarrow A} * p_{A \leftarrow B}$$

The first product $p_{A \rightarrow B} * p_{B \leftarrow A}$ is an indicator of the cumulative strength of the connections from A to B, while the second one is an indicator of the cumulative strength of the connections from B to A. In general, given two nodes, namely X and Y, the term $p_{X \rightarrow Y}$ is an indicator of the strength of the interconnections due to weighted outgoing edges from X to Y, while $p_{Y \leftarrow X}$ indicates the strength of the interconnections due to weighted incoming edges reaching Y from X.

The term $p_{X \rightarrow Y}$ is expressed as follows:

$$p_{X \rightarrow Y} = N_{LINK}(X \rightarrow Y) \cdot w_{LINK}^{OUT}(X) + N_{SUBMIT}(X \rightarrow Y) \cdot w_{SUBMIT}^{OUT}(X) + N_{REDIRECT}(X \rightarrow Y) \cdot w_{REDIRECT}^{OUT}(X)$$

where $N_{LINK}\ (X \rightarrow Y)$, $N_{SUBMIT}\ (X \rightarrow Y)$ and $N_{REDIRECT}(X \rightarrow Y)$ are the number of connections of *link*, *submit* and *redirect* type from X to Y, respectively.

Analogously, the term $p_{B \leftarrow A}$ is expressed as the sum of weighted *incoming edges* reaching B from A.

If the value of the product $p_{A \rightarrow B} * p_{B \leftarrow A}$ is one, all edges outgoing from A reach B, and there is no edge going into B that does not come from A. If the symmetrical condition for the product $p_{B \rightarrow A} * p_{A \leftarrow B}$ is also true, the degree of coupling $C_{A,B}$ will assume the maximum value, that is equal to two. The minimum value of $C_{A,B}$ is zero, when the nodes are not directly connected.

5.3.3 The clustering algorithm

Agglomerative hierarchical clustering algorithms start from the individual items, gather them into small clusters, which are in turn gathered into larger clusters up to one final cluster containing everything. The result is a hierarchy of clusters.

The proposed hierarchical algorithm is iterative, starts from a clustering with n clusters, each one containing a single Web Application component, and produces new clusters based on four clustering rules:

- *R1*: the cluster containing a *built client page* is merged with the cluster containing the *server page* building the former;

- *R2*: *if and only if* all the pages referenced by the frame tags of a *client page with frame* belong to the same cluster, the cluster including the latter page is merged with the former cluster;

- *R3*: *if and only if* all the pages including a *client module* or a *server page* belong to the same cluster, the cluster including the former pages is merged with the latter cluster;

- *R4*: the pair of clusters whose coupling value is the maximum one is gathered into a new cluster.

The algorithm applies to the connection graph WAG=(N,E) of a web application. The description of the algorithm is provided in Fig. 5.2, where *n* is the cardinality of the node set N, *c* indicates a generic cluster in a given clustering, and *x* indicates any of *link*, *redirect*, and *submit* relationships.

```
BEGIN with n clusters each containing one Web Application component;
DEFINE the wL, wRL and wSL values;
FOR EACH cluster containing a built client page component, APPLY rule R1;
WHILE (there is at least a couple of connected clusters) DO
  FOR EACH cluster containing a client page with frame component, APPLY rule R2;
  FOR EACH cluster containing a client module component, APPLY rule R3;
  FOR EACH cluster c, and for each x, COMPUTE wxOUT (c) and wxIN (c);
  FOR EACH pair of clusters, COMPUTE the couplings between them;
    APPLY rule R4;
OD
```

Figure 5.2: The clustering algorithm

This algorithm results in a hierarchy of clustering, each one containing a set of clusters. However, in order to obtain a partition of the application components rather than a hierarchy, the hierarchy can be pruned at an appropriate height and considering only the upper clusters. The choice of the appropriate cut-height can be based on specific quality metrics. Possible metrics are those expressing the quality of a given clustering. Of course, the quality of a clustering is as good as it supports the comprehension of the web application.

According to [Man98], [Man99], a good clustering includes clusters with high intra-connectivity and low inter-connectivity. The intra-connectivity expresses the degree of cohesion among entities of a web application, and the inter-connectivity can be interpreted as a coupling measure among entities of a web application. Therefore, the *Quality of a Clustering* metric QoC is introduced, that can be expressed as the difference between IntraConnectivity and InterConnectivity of a clustering:

QoC= IntraConnectivity – InterConnectivity.

where:

$$IntraConnectivity = \frac{1}{NC}\sum_{\substack{k\in CLUSTER\\ Card(k)>1}} \frac{\sum_{i,j\in CLUSTER_k} p^0_{i\to j}\cdot EOUT^0_i}{Card(k)\cdot(Card(k)-1)}$$

$$InterConnectivity = \begin{cases} \dfrac{1}{\dfrac{NC\cdot(NC-1)}{2}} \displaystyle\sum_{\substack{h,k\in CLUSTER\\ h\neq h}} \frac{\sum_{i\in CLUSTER_h}\sum_{j\in CLUSTER_k} p^0_{i\to j}\cdot EOUT^0_i}{2\cdot Card(h)\cdot Card(k)} & \text{if } NC>0 \\ 0 & \text{if } NC=0 \end{cases}$$

In these expressions, *NC* is the number of clusters in the considered clustering configuration, $p^0_{i\to j}$ is the $p_{i\to j}$ between two generic nodes *i* and *j* in the original Connection Graph of the web application, $CLUSTER_k$ is the k-th cluster in the considered clustering, $EOUT^0_i$ is the number of edges leaving the i-th node in the original Connection Graph of the web application, Card(x) is the cardinality of the x-th cluster of the considered clustering.

The *IntraConnectivity* is a weighted mean of the cluster inner edges. Its values vary between 0 (when no cluster has got inner edges) and 1 (all clusters inner nodes are completely connected). The *InterConnectivity* is a weighted mean of the edges among clusters. Its values vary between 0 (clusters not connected by edges at all) and 1 (every node in the clusters is connected to every other node from the other clusters).

The QoC of a given clustering assumes values ranging from –1 to +1. The minimum value is obtained with a clustering having each cluster with a single node and inter-connected with all the other ones. Besides, the maximum value is assumed either by a clustering with only one cluster including all the nodes, or by a clustering with only isolated clusters that are completely intra-connected.

The clustering obtained at each iteration of the proposed algorithm is characterized by a given value of the QoC quality metric. The clustering exhibiting the maximum value of QoC is a candidate to implement the best partition of the web application components. Therefore, the hierarchy of clustering may be cut at the cut-height that is associated with the maximum QoC. The clusters from this configuration are submitted to a validation process, with the aim of establish if the cluster completely realizes a functionality of the system.

The problem of comprehending a Web Application can be addressed according to the following structured approach:

1) Reverse engineering of the Web Application and production of the WAG;
2) Do clustering according to the proposed algorithm;
3) Find the C_{max} clustering with the maximum QoC value;
4) Submit the C_{max} clustering to a validation process.

The validation process is a human intensive process and it is needed to determine if each cluster realizes a functionality of the Web Application. This process will be partially supported by the Reverse Engineering techniques that are described in the Chapters 9, 10 and 11.

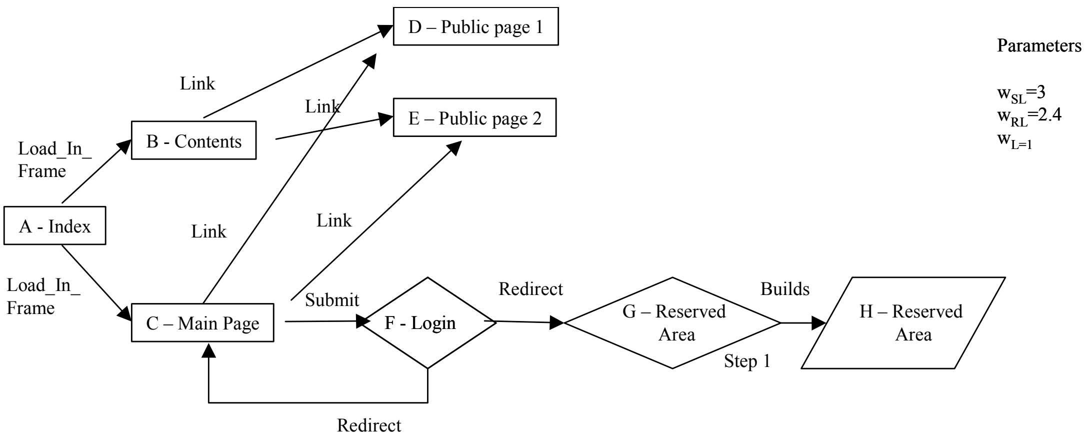
During the clustering validation process, clusters can be modified splitting one cluster in more ones, or merging more clusters into one, or moving some components from a cluster to another one to have valid clusters.

5.4 A clustering example

In order to illustrate how the proposed clustering algorithm works, in this section a small exemplar Web Application is analysed in conformance with the clustering method.

The pages of the application are divided into two areas: a public area is accessible by all users and a reserved area whose access is limited to registered users.

The application is composed of eight items, including five pages (labelled from A to E), two server pages (labelled as F and G), and a built client page (namely H). The Web Application representation is provided in Fig. 5.3-a, according to the notation presented in the previous chapter.



- a -

Step 2

$p_{B\to D}=p_{B\to E}=1/2$ $p_{C\leftarrow F}=1$ $C_{B,D}=C_{B,E}=1/2*1/2=1/4$

$p_{C\to D}=p_{C\to E}=1/5$ $p_{F\leftarrow C}=1$ $C_{C,D}=C_{C,E}=1/5*1/2=1/10$

$p_{C\to F}=3/5$ $p_{GH\leftarrow F}=1$ $\mathbf{C_{C,F}=3/5*1+1/2*1=11/10}$

$p_{F\to GH}=1$ $p_{D\leftarrow B}=p_{E\leftarrow B}=1/2$ $C_{F,GH}=1*1=1$

$p_{F\to C}=1/2$ $p_{D\leftarrow C}=p_{E\leftarrow C}=1/2$ **QoC=0.132143**

- b -

Step 3

$p_{B\to D}=p_{B\to E}=1/2$ $p_{GH\leftarrow CF}=1$ $C_{B,D}=C_{B,E}=1/2*1/2=1/4$

$p_{CF\to D}=p_{CF\to E}=1/4.4$ $p_{D\leftarrow B}=p_{E\leftarrow B}=1/2$ $C_{CF,D}=C_{CF,E}=1/4*1/2=1/8$

$p_{CF\to GH}=0.545$ $p_{D\leftarrow CF}=p_{E\leftarrow CF}=1/2$ $\mathbf{C_{CF,GH}=0.545*1=0.545}$

QoC=-0.023889

- c -

Step	***QoC***	***C***
1	***-0,*11607**	
2	**0,132143**	**1,1**
3	**-0,02389**	**0,545**
4	**0.127083**	**0,25**
5	**0.022778**	**0,5**
6	**0,055556**	**1**

The best QoC is for C=1.1

- d -

Fig. 5.3: An example

The home page of the application (labelled A) includes two frames, the first one (B) providing the access to pages of the public area (D and E, respectively), and the second one (C) containing a form for accessing the reserved area (composed of G and H pages) through an authentication page (F).

The clustering algorithm produced a hierarchy of clusters. At the first step, G and H pages were merged into a cluster named GH, according to the *Build* rule. At the second step, C and F pages were merged into a new cluster CF since their degree of coupling $C_{C,F}$ assumed the maximum value (cfr. the coupling values listed in Fig. 5.3-b). The new cluster CF-GH composed of clusters CF and GH was analogously obtained at the third step (cfr. the coupling values listed in Fig. 5.3-c). At the final iteration of the algorithm, the sixth step, one cluster including all the application items was obtained. The QoC values obtained at each iteration are reported in Fig. 5.3-d. The maximum QoC value was obtained at the second step.

The clusters obtained at this step were analysed in order to assess their validity. Valid clusters have been recognized, since each of them implemented a specific function. The GH cluster implements the visualization of reserved information to authenticated users, while the CF cluster implements the user authentication function. The remaining clusters A and B implement coordination functions, while D and E provide the visualization of two distinct groups of information.

This first experiment provides encouraging results about the effectiveness of the methods, but many other experiments have been carried out. The results of these experiments, based on real world Web Applications, are presented in Chapter 7, while a tool supporting the execution of the clustering algorithm is presented in the next chapter.

Further detail and examples about the proposed clustering method can be also found in [Dil02c].

Chapter 6: The tool WARE

In this chapter a tool called WARE is presented. This tool supports the Reverse Engineering process described in Chapter 4: it supports the extraction of information from the source code of an existing Web Application and from the analysis of its execution, the abstraction of detailed UML diagrams depicting the structure of a Web Application and the execution of the clustering algorithm described in the previous chapter. This tool also supports metrics evaluation. Architecture, functionalities and examples of the use of this tool are presented.

6.1 Introduction

The most part of the tasks of the Reverse Engineering process described in the previous chapters may be carried out automatically: examples are the extraction of information from the source code of Web Applications, the clustering algorithm and the abstraction of class diagrams depicting the structural view of the Web Application. The automatization of these tasks reduces drastically the effort related to the execution of the WARE Reverse Engineering process (experiments described in the following chapter have confirmed this hypothesis). A tool, that is also named WARE, has been designed and developed with the aim to support the WARE Reverse Engineering process.

In this chapter the architecture and the functionality of the tool WARE are described and an example of how the tool can support the Reverse Engineering is reported. The results of a further experimentation, based on a number of different Web Applications is reported in the following chapter.

6.2 Architecture of tool WARE

The tool WARE has been designed as an integrated environment including several components arranged in the software architecture shown in Figure 6.1. As the figure illustrates, the WARE tool architecture have three layers: the Interface Layer, the Service Layer, and the Repository Layer. The *Interface Layer* implements the 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 *Extractors* and the *Abstractors*. The *Repository* stores the information extracted and abstracted about the Web Application using intermediate format files and a relational database. Additional details about WARE's layers are provided in the following.

6.2.1 WARE Service Layer.

The Service layer of WARE includes both Extractor and Abstractor components: Extractors directly retrieve relevant information from the source code of an application and store it in intermediate format files, while Abstractors are able to abstract further information and documents from the directly retrieved information.

Source code extractors included in WARE are implemented in C++ language, and the set of analysable source code languages is constantly in evolution. The current version of the tool includes extractors that analyse the HTML language (version 4.0) and some scripting languages, such as the Javascript and VBScript languages usable at the client side of a Web Application, and ASP and PHP scripts from the server side. These extractors do not recover an Abstract Syntax Tree from the analysed code, but they just recognise and identify in the code the information needed to re-build the required diagrams, by means of lexical and syntactical analysis. The difficulties involved into the static analysis phase are due to the peculiarities of the scripting technologies. Often, the most part of the grammar recognition problems are based on the fundamental hypothesis of correctness of the code under analysis.

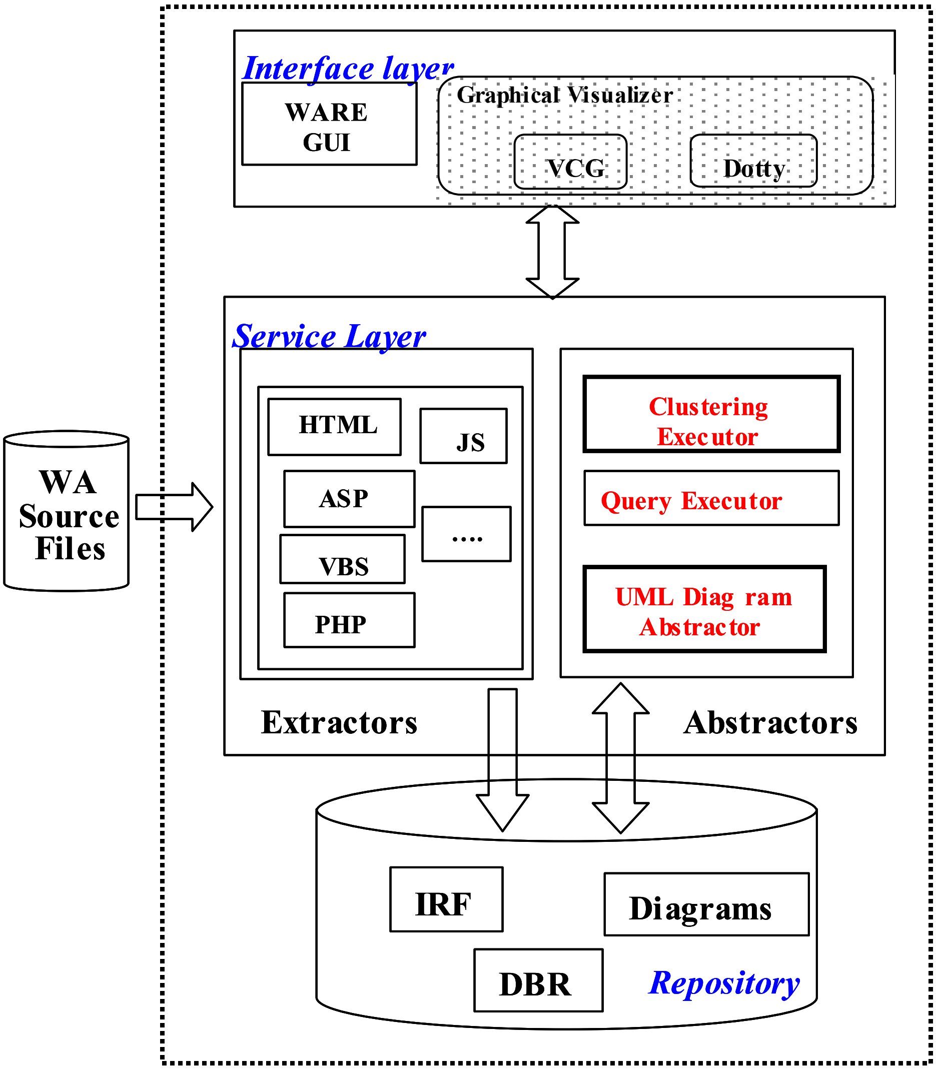


Figure 6.1: Architecture of tool WARE

For Web Application domain, this hypothesis can be considered for server scripting code but it cannot be guaranteed for client scripting code. In fact, it is possible that server scripting code or client scripting code generate client code dynamically during execution. Moreover, a common characteristic of the browser is the fault tolerance: for this reason the existence of Web pages containing incorrect HTML code is quite common. So, an ad-hoc fault tolerant approach based on authoms has been adopted to analyse the source code of Web Applications. This technique has been detailed described in [Tra01]. Extractors are also able to recognise some lexical and syntactical errors contained in the source files. Finally, parsers store the extracted information in an Intermediate Representation Form (IRF) that is implemented as a tagged XML-like file (see [Dil02] for more details); this file is then parsed by an *IRF translator* component that populates the relational Database included in the Repository with the information produced by the Extractors.

As to the Abstractors, a first one is the *Clustering Executor,* supporting the clustering algorithm that is described in the next chapter. The *Query Executor* is the Abstractor that implements predefined SQL queries over the database and retrieves data about the Web Application that may aid Concept Assignment processes and dynamic analysis execution. Possible information provided by the Query Executor includes the list of a Web page links, Web page inner items (such as Scripts or other Objects), Form fields, Client/Server Functions activated in a Web page, and so on. However, the tool supports customized queries, too. Finally, the *UML Diagram Abstractors* implement several abstraction tasks supporting the recovery of UML diagrams, including the class diagrams at different degrees of detail (*e.g.*, providing only client pages, or static pages, or filtering out the forms, etc.).

6.2.2 WARE Interface Layer.

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

The main functions available to a user include:

- Automatic Static Analysis of a Web Application source code.
- Support to the recovery of the Dynamically Instantiated Elements, that allows a user to find source code statements producing dynamically instantiated components or relationships, which cannot be retrieved automatically with a static source code analysis. (This information can be, therefore, stored in the repository using a specific user interface offered by WARE).
- Information Browsing, by means of which a user can browse information recovered about a Web Application, such as the inventory of the Web Application components and their source code. Moreover, the reachability relationship of a given component can be computed. WARE allows each element of a page into the source code to be localized. Moreover, users may also formulate customisable queries over the database, by choosing the type of application item, relationship, or parameter to be searched for and displayed.
- Graphical Visualization. WARE is able to show the graphical representations of the following models:
  a. Class diagram of the Web Application, depicted according to the extensions defined by Jim Conallen [Con99];
  b. Web Application Connection Graph (WAG), showing the relationships between the components of a Web Application (cfr. Subsection 5.3.1);

c. Reachability Graph, showing the pages that can be reached from a given page of a Web Application.

- Clustering functions. WARE automatically executes the clustering algorithm described in the previous chapter. WARE produces different clustering configurations and therefore a user is able to analyse each of them and to select the one that maximises cohesion between Web Application components and minimises coupling between them. The user may associate a descriptive name and a colour to each cluster or he can modify the chosen configuration. WARE allows a user to know what are the relationships and the data exchanged between the components inside a cluster and between the components inside a cluster and the remaining components of the Web Application. These information may be very useful to validate a cluster and to establish what is the user function implemented by each cluster. Moreover, WARE allows the WA components to be grouped in subsets by using other simple clustering criteria. A simple criterion implemented by WARE consists of grouping together physical components of a Web Application (Client and Server pages, client modules, multimedia components, etc.) contained in the same directory on the web server. However, a user can create a subset of WA components by selecting them from the inventory list.

Currently, the graphical visualisation of the diagram is achieved using some freeware graph displayers, such as VCG [Vcg], and Dotty [Dot]: this visualisation does not support the UML notation style, but different shapes and colours are used to draw different kinds of entity and relationship. As an example, a box is used for drawing a Static Page, a trapezoid for a Built Client Page, a diamond for a Server Page and a triangle for a Form. However, export of the diagrams in XMI format has been considered and is under developing.

Of course, WARE allows the visualization of customized graphs, reporting only some node types and some edge types.

- Evaluation of Software Metrics. Some summary measures, such as the number of Web Application pages, scripts, or the LOC count of a Web Application are automatically computed by the tool and showed on demand to the user. An ad-hoc tool using information extracted and abstracted by WARE calculates more complex metrics. These metrics are used to estimate the maintainability of a Web Application in Chapter 13.

WARE is also provided with some wizards that guide users during the tool configuration phase and creation of a new project. The user interface of WARE supports both Italian and English languages.

6.3 Analysis of a Web Application with the tool WARE

In this section the results of the submission of a real Web Application to the Reverse Engineering process is reported. The case study introduced in this chapter is extended and discussed in deep in the following chapters.

The selected Web Application supports the activities of undergraduate courses offered by a Computer Science Department. It provides students and teachers with different functions, such as accessing information about the courses, allowing a student to get registered to a course or to an examination session, allowing a professor to manage the examination and student tutoring agendas, allowing a registered student to download teaching material, and so on.

The application was developed using Microsoft ASP and VBScript languages on the server side of the application, Javascript and HTML on the client side. It runs on Microsoft IIS Web Server. Only the files constituting the source code of the Web Application are available, with no design documentation.

The WARE tool has automatically realized static analysis. The results of the analysis have been stored in text files according to the defined Intermediate Representation Form. Figure 6.2 reports an example of the transformation from the HTML code (auth.htm on the left side of the figure) to the IRF (auth.htm.irf on the right side of the figure) of a web page. The information contained in the IRF files has been extracted and stored in a relational database by WARE, according to the model presented in Chapter 3.

Inputs needed by WARE to correctly execute static analysis of a Web Application comprehend also some information about the configuration of the web server on which the Web Application is executed (e.g. what files extensions the web server associates to server pages, what tags are interpreted as starting server script tag, etc.). WARE extractors produce also a log file reporting some syntax errors recognized in source files (such as opening tags without needed closing ones, closing tags without opening ones, tags without compulsory attributes, etc.). These errors aren't reported by browsers, which try to interpret also incorrect HTML pages. Figure 6.3 shows a list of warnings reported for the Web Application under analysis. A code inspection revealed that these warnings correspond to actual syntax errors in the web pages source code.

Dynamic analysis was therefore performed in the second step of the Reverse Engineering process, with the aim of recover information about the Web Application that is not obtainable by static analysis. For instance, this analysis is needed to retrieve the actual content of dynamically built client pages, since this content can be precisely defined only executing the code.

```
Line #  File auth.htm                                     File auth.htm.irf

   1    <html>                                            <OPEN><FILENAME="\auth.htm"></OPEN>
   2    <form name=auth method=post action="auth.asp">    <OPEN FORM> <LINE=2>
   3    Login:<input name=login type=text>                  <NAME="auth"> <METHOD="post">
   4    Password:<input name=pwd type=text>                 <ACTION="auth.asp">
   5    <input type=submit>                               </OPEN FORM>
   6    <input type=reset>                                <INPUT><LINE=3>
   7    </form>                                             <NAME="login"> <TYPE="text">
   8    </html>                                           </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 6.2: An excerpt of the source code of a web page and its corresponding Intermediate Representation Form

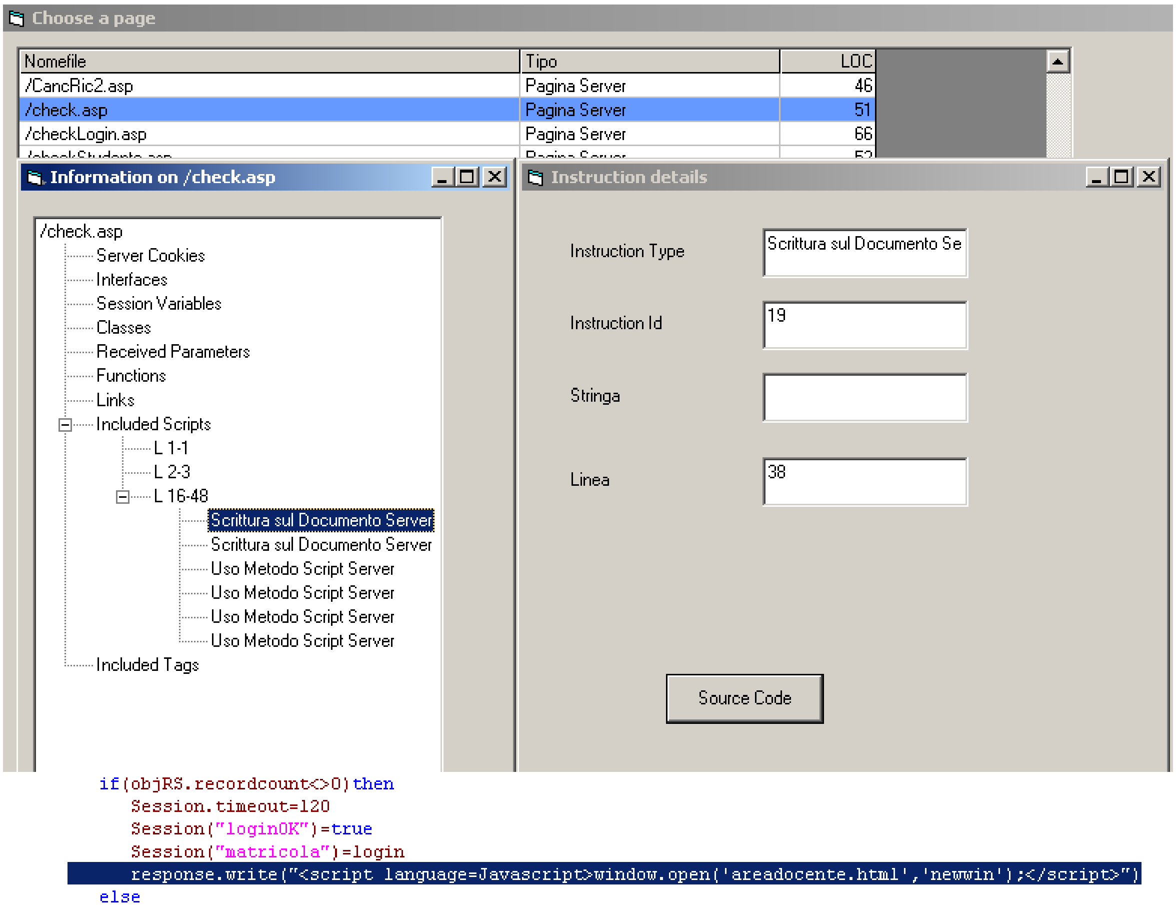
```
Error in /areastudenti.html Line:17  Warning: attribute PATH needed
Error in /main.html Line:21          Warning : attribute PATH needed
Error in /maindoc.html Line:21       Warning : attribute PATH needed
Error in /menudoc.html Line:17       Warning : attribute PATH needed
Error in /FormPreRicev2.asp Line:417 Warning: tag </A> needed
Error in /VisListaRic.asp Line:327   Warning: tag </A> needed
Error in /VisPreRic.asp Line:302     Warning: tag </A> needed
```

Figure 6.3: Warning reported as result of the static analysis of the Web Application

An example is reported in Figure 6.4, where the Information Browsing functionality of WARE has been used to individuate an output statement in the server page check.asp (the response.write instruction reported in the lower part of the figure). Analysing the semantic of this statement, it is possible to establish that a Redirect operation from a client page built by check.asp and the client page areadocente.html could be instantiated (by means of the javascript method window.open).

Some summary information about the components and relationships of the Web Application under analysis can be evaluated automatically by WARE. Figure 6.5 reports both the count of statically retrieved information, such as the number of server pages, client pages, scripts, forms, functions, connections composing the Web Application, both dynamically obtained information.

Finally, the capability of WARE of producing automatically graphical views of the Web Application was exploited during the working session. The current version of WARE produces graphs that can be visualized with the tools VCG [Vcg] and Dotty [Dot]. As an example, in Figure 6.6 is reported the Class Diagram representing the web pages implementing the whole Web Application. In this diagram different colours and shapes have been used to characterize different types of components and connections: server pages have been depicted as diamonds, static client pages with rectangles, built client pages with trapeziums and so on.



Nomefile	Tipo	LOC
/CancRic2.asp	Pagina Server	46
/check.asp	Pagina Server	51
/checkLogin.asp	Pagina Server	66
/checkStudente.asp	Pagina Server	52

```
/check.asp
    Server Cookies
    Interfaces
    Session Variables
    Classes
    Received Parameters
    Functions
    Links
    Included Scripts
        L 1-1
        L 2-3
        L 16-48
            Scrittura sul Documento Server
            Scrittura sul Documento Server
            Uso Metodo Script Server
            Uso Metodo Script Server
            Uso Metodo Script Server
            Uso Metodo Script Server
    Included Tags
```

Instruction Type: Scrittura sul Documento Se

Instruction Id: 19

Stringa:

Linea: 38

```
if(objRS.recordcount<>0)then
    Session.timeout=120
    Session("loginOK")=true
    Session("matricola")=login
    response.write("<script language=Javascript>window.open('areadocente.html','newwin');</script>")
else
```

Figure 6.4: Use of the Information Browsing functionality of WARE to detect a Redirect operation that is dynamically instantiated

Statically retrieved information	
Server Page	75
Static Client Page	23
Built Client Page	74
Client Script	132
Client Function	48
Form	49
Server Script	562
Server Function	0
Redirect (in Server Scripts)	7
Redirect (in Client Scripts)	0
Link	45
Dynamically Instantiated Elements	
Link	0
Submit	0
Redirect (in Client Script)	0
Redirect (in Server Script)	7

Figure 6.5: Structural metrics of the Web Applications under analysis

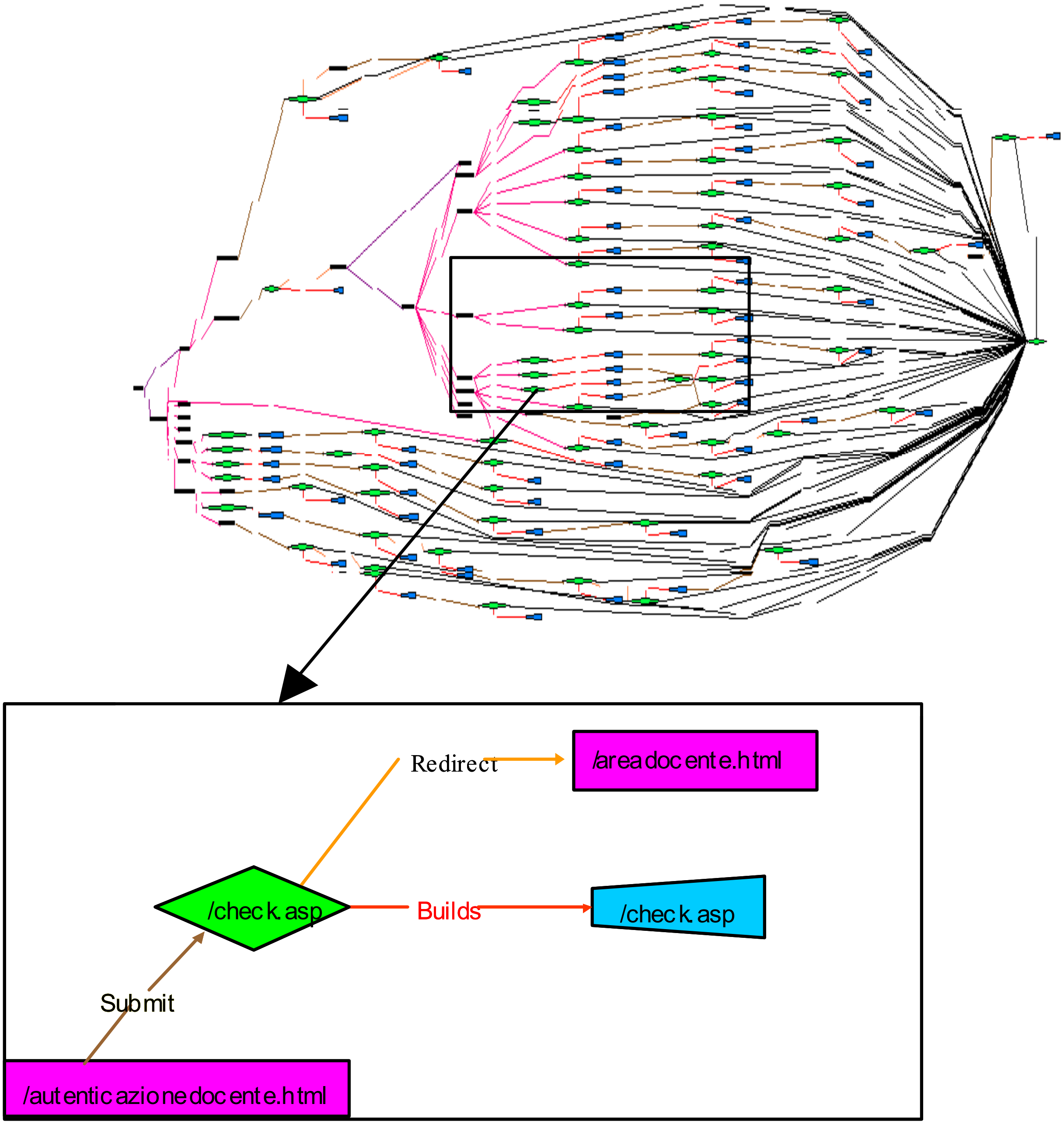


Figure 6.6: Class diagram reporting the pages of the Web Application and the relationships between them (a zoom of a part of this diagram is shown in the lower frame of the figure)

However, it is possible to modify colours and shapes of components and connections in the produced diagrams. Edges represent connections between the components and they are labelled according to the typology of that connection. Colours of the edges may be changed for a better visualization.

When the static analysis of the WA code and the recovery of dynamic information have been accomplished, the WA can be submitted to the clustering algorithm described in the previous chapter, in order to obtain a collapsed view of its structure. Clustering was therefore executed on the subject WA, producing various clustering configurations.

The optimal clustering configuration proposed by the tool was submitted to a validation process, in order to understand the functions implemented by the clusters and to validate them. The process was carried out with the support of WARE, and by inspecting the code and observing the Web Application execution.

The optimal clustering of the Web Application presented 49 Clusters, with an Average of 3.58 Pages per Cluster. Figure 6.7 shows the graph of this clustering, where each node represents a cluster. The size of this graph is sensibly smaller than the WAG one, since it includes 49 nodes, that is, less than 30% of the nodes of the original WAG (considering just the web pages).

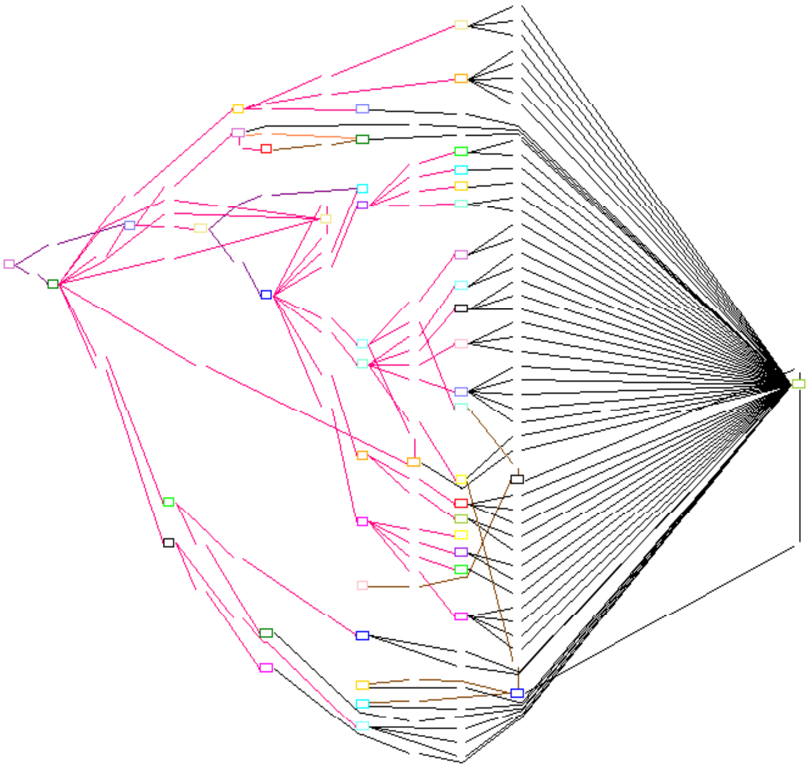
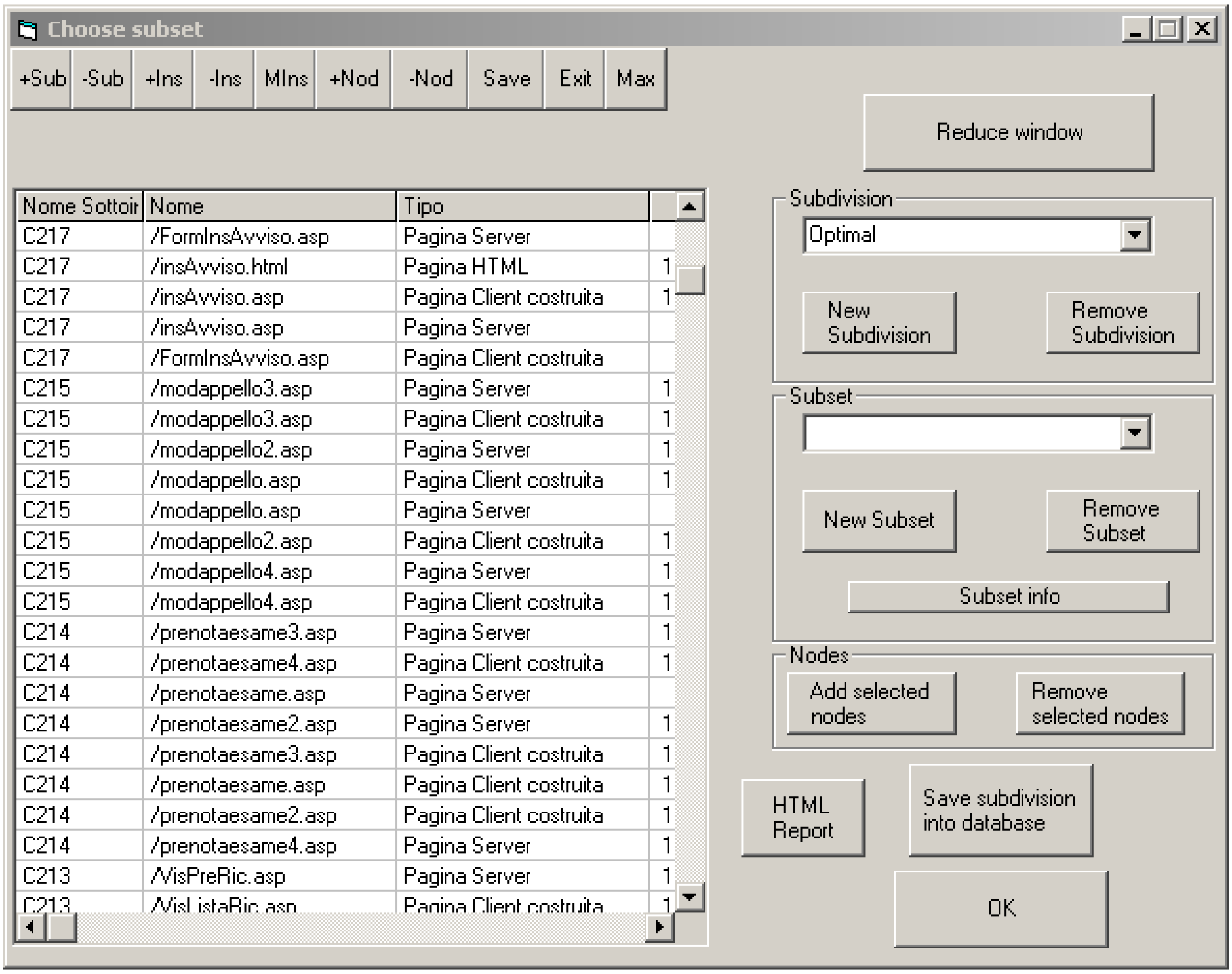


Figure 6.7: Clusterized WAG

During the clustering validation phase, some clusters were modified splitting one cluster in more ones, or merging more clusters into one, or moving some components from a cluster to another one to have valid clusters. For the analysed WA 8 were merged to produce 3 new clusters.

The WARE Cluster Management User Interface (Figure 6.8) usefully supported these tasks, allowing insertion, modification or deletion of clusters and showing tables and diagrams reporting relationships and exchanged data among the WA pages involved by each cluster. As an example, Figure 6.9 shows the form reporting in the higher part of the figure the relationships among the pages inside a cluster of the WA under analysis, while in the lower part data exchanged between these pages are reported. Figure 6.10 reports a summary after the clustering configuration validation.

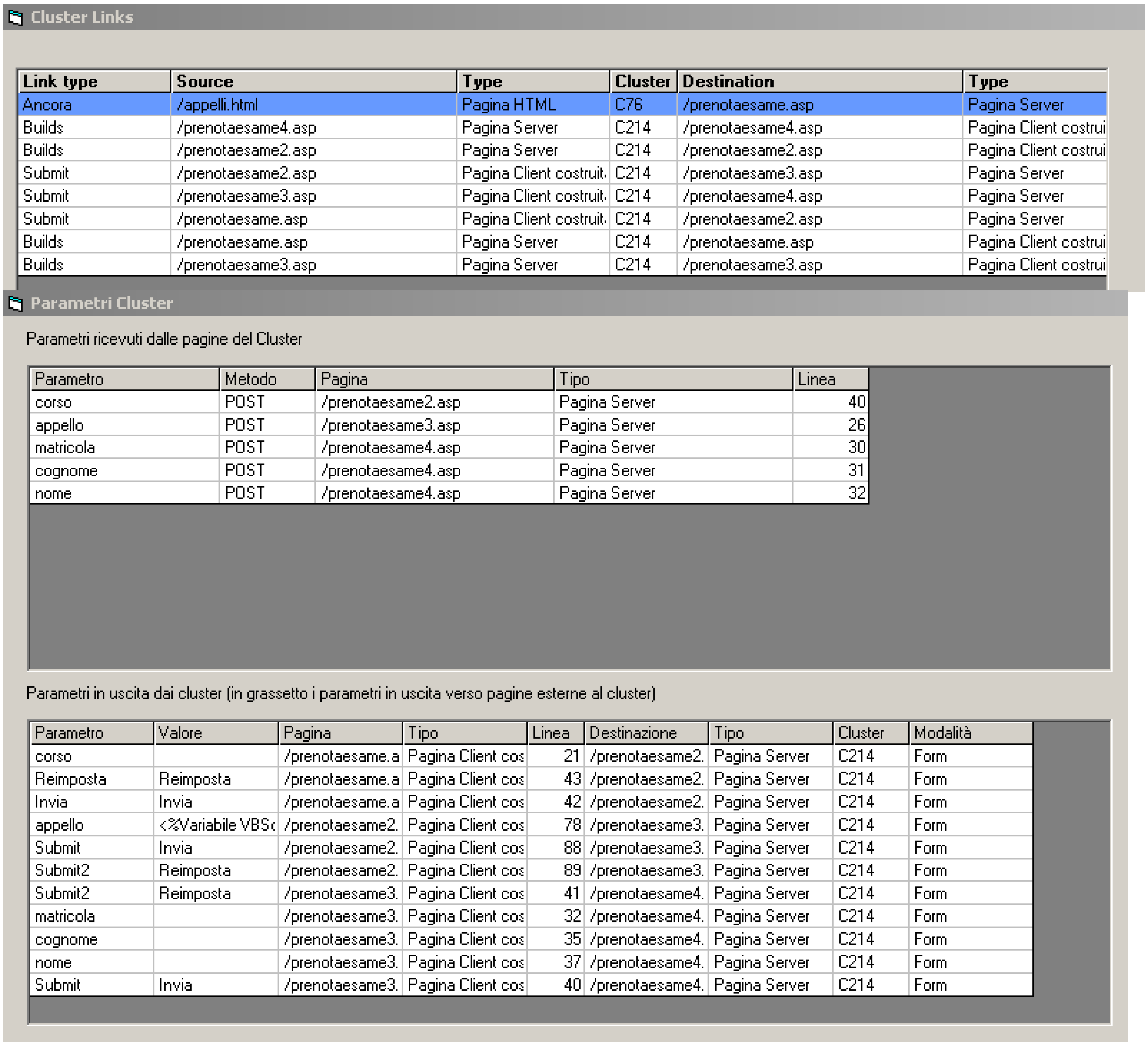
Reachability tables and diagrams were used to establish if a cluster is connected to the remaining part of the Web Application or not. Usually, isolated clusters represent obsolete parts of a Web Application. As an example, Cluster #43, constituted by the server page insappello2.asp and by its built client page cannot be reached by any other component of the Web Application.



Subdivision: Optimal

Nome Sottoir	Nome	Tipo	
C217	/FormInsAvviso.asp	Pagina Server	
C217	/insAvviso.html	Pagina HTML	1
C217	/insAvviso.asp	Pagina Client costruita	1
C217	/insAvviso.asp	Pagina Server	
C217	/FormInsAvviso.asp	Pagina Client costruita	
C215	/modappello3.asp	Pagina Server	1
C215	/modappello3.asp	Pagina Client costruita	1
C215	/modappello2.asp	Pagina Server	1
C215	/modappello.asp	Pagina Client costruita	1
C215	/modappello.asp	Pagina Server	
C215	/modappello2.asp	Pagina Client costruita	1
C215	/modappello4.asp	Pagina Server	1
C215	/modappello4.asp	Pagina Client costruita	1
C214	/prenotaesame3.asp	Pagina Server	1
C214	/prenotaesame4.asp	Pagina Client costruita	1
C214	/prenotaesame.asp	Pagina Server	
C214	/prenotaesame2.asp	Pagina Server	1
C214	/prenotaesame3.asp	Pagina Client costruita	1
C214	/prenotaesame.asp	Pagina Client costruita	1
C214	/prenotaesame2.asp	Pagina Client costruita	1
C214	/prenotaesame4.asp	Pagina Server	1
C213	/VisPreRic.asp	Pagina Server	1
C213	/VisListaRic.asp	Pagina Client costruita	1

Figure 6.8: Subset Management User Interface



Cluster Links

Link type	Source	Type	Cluster	Destination	Type
Ancora	/appelli.html	Pagina HTML	C76	/prenotaesame.asp	Pagina Server
Builds	/prenotaesame4.asp	Pagina Server	C214	/prenotaesame4.asp	Pagina Client costrui
Builds	/prenotaesame2.asp	Pagina Server	C214	/prenotaesame2.asp	Pagina Client costrui
Submit	/prenotaesame2.asp	Pagina Client costruit	C214	/prenotaesame3.asp	Pagina Server
Submit	/prenotaesame3.asp	Pagina Client costruit	C214	/prenotaesame4.asp	Pagina Server
Submit	/prenotaesame.asp	Pagina Client costruit	C214	/prenotaesame2.asp	Pagina Server
Builds	/prenotaesame.asp	Pagina Server	C214	/prenotaesame.asp	Pagina Client costrui
Builds	/prenotaesame3.asp	Pagina Server	C214	/prenotaesame3.asp	Pagina Client costrui

Parametri Cluster

Parametri ricevuti dalle pagine del Cluster

Parametro	Metodo	Pagina	Tipo	Linea
corso	POST	/prenotaesame2.asp	Pagina Server	40
appello	POST	/prenotaesame3.asp	Pagina Server	26
matricola	POST	/prenotaesame4.asp	Pagina Server	30
cognome	POST	/prenotaesame4.asp	Pagina Server	31
nome	POST	/prenotaesame4.asp	Pagina Server	32

Parametri in uscita dai cluster (in grassetto i parametri in uscita verso pagine esterne al cluster)

Parametro	Valore	Pagina	Tipo	Linea	Destinazione	Tipo	Cluster	Modalità
corso		/prenotaesame.a	Pagina Client cos	21	/prenotaesame2.	Pagina Server	C214	Form
Reimposta	Reimposta	/prenotaesame.a	Pagina Client cos	43	/prenotaesame2.	Pagina Server	C214	Form
Invia	Invia	/prenotaesame.a	Pagina Client cos	42	/prenotaesame2.	Pagina Server	C214	Form
appello	<%Variabile VBS	/prenotaesame2.	Pagina Client cos	78	/prenotaesame3.	Pagina Server	C214	Form
Submit	Invia	/prenotaesame2.	Pagina Client cos	88	/prenotaesame3.	Pagina Server	C214	Form
Submit2	Reimposta	/prenotaesame2.	Pagina Client cos	89	/prenotaesame3.	Pagina Server	C214	Form
Submit2	Reimposta	/prenotaesame3.	Pagina Client cos	41	/prenotaesame4.	Pagina Server	C214	Form
matricola		/prenotaesame3.	Pagina Client cos	32	/prenotaesame4.	Pagina Server	C214	Form
cognome		/prenotaesame3.	Pagina Client cos	35	/prenotaesame4.	Pagina Server	C214	Form
nome		/prenotaesame3.	Pagina Client cos	37	/prenotaesame4.	Pagina Server	C214	Form
Submit	Invia	/prenotaesame3.	Pagina Client cos	40	/prenotaesame4.	Pagina Server	C214	Form

Figure 6.9: Relationships and data exchanged between the components of a cluster and the remaining part of the Web Application

Number of initial clusters	49
Number of split clusters	0
Number of incomplete clusters	8
Number of valid clusters	41
Number of new clusters obtained from the subdivided ones	0
Number of new clusters obtained by merging incomplete clusters	3
Number of final valid clusters	44

Figure 6.10: Results from Clustering validation

Further details about the use of the tool WARE can be found in [Dil04f] and [Dil04g], while an extended experimentation of the Reverse Engineering process will be reported and discussed in the next chapter.

Chapter 7: Experimenting the WARE Reverse Engineering Process

In this chapter the results of the experimentation of the Reverse Engineering process presented in the previous chapters, are reported. A discussion about the programming techniques needed to improve the understandability of the application is reported.

7.1 Assessing the effectiveness of the WARE Reverse Engineering process

The Reverse Engineering approach presented in Chapter 4 has been experimented with a validation experiment. This experiment was conducted with a number of real Web Applications. Software engineers who were expert users of Web Application technologies were enrolled in the experiment. They were taught the reverse engineering process and tool facilities, and were asked to use them to analyse existing Web Applications. Software engineers were grouped in teams composed of 2 or 3 people, and each team was assigned a single application.

As to the experimental materials, six Web Applications with different characteristics and implemented using ASP, Javascript, PHP, and HTML technologies, were selected. According to Tilley and Huang's classification (cfr. Section 2.2), three of them were class 3 applications including dynamic client and server pages (hereafter these applications are called WA1, WA2, and WA3). Two class 2 applications (WA4 and WA5) with dynamic functions just on the client side were considered, along with a primarily static class 1 application (WA6).

As regards the domain of the Web Applications, WA1 supported the activities of an undergraduate course (it has been presented also in chapter 5); WA2 provided functions for the management of an Italian research network, WA3 and WA5 were two personal Web sites. Finally, WA4 was an application supporting the activities of a society for historical studies, and WA6 was a Web site providing the on line reference guide of a programming language.

While WA1 was developed without any automatic generator of HTML code, an automatic generator was used for producing some presentation-related aspects of WA2, WA3, and WA4 (such as table layout). For WA5, this tool was used to define the navigational structure, too, by automatically generating navigational bars.

The teams carried out the analysis of the Web Applications according to the prescriptions of the reverse engineering process. The results they achieved at each step of the process are described below. Further detail about this experiment can be found in [Dil02d] and [Dil04].

7.2 Carrying out Static Analysis

In the first step of the Reverse Engineering process, the teams carried out static analysis with the support of the tool WARE, with the aim of detecting the Web Applications' items and their relationships. Table 7.1 reports a summary of the data collected about the applications: in the first column, the type of the item or relationship is reported, while in the remaining columns, the count of items/relationships retrieved for each Web Application analysed is shown. In this Table, the name of the items corresponds to the name of the Web Application entities enclosed in the reference model presented in Chapter 3.

Table 7.1: Statically retrieved data from the analysed Web Applications

Item type	WA1	WA2	WA3	WA4	WA5	WA6
Server Page	75	105	21	0	0	0
Static Page	23	38	19	80	45	257
Built Client Page	74	98	20	0	0	0
Client Script	132	225	113	261	4	3
Client Function	48	32	60	68	1	4
Form	49	100	5	0	25	5
Server Script	562	2358	40	0	0	0
Server Function	0	11	0	0	0	0
Redirect (in Server Scripts)	7	0	0	0	0	0
Redirect (in Client Scripts)	0	0	41	0	0	0
Link	45	266	121	162	448	1508

Thanks to the automation of the analysis by the tool WARE, the human effort required to accomplish this step was limited just to the activation of the tool parsers and of the IRF translator that populates the Repository with the information extracted from the source code.

These tasks were automatically executed by the tool, in a number of seconds (ranging from 14 seconds to 62 seconds) that depended on the size of the application analysed (cfr. Table 7.4 for effort data).

7.3 Carrying out Dynamic Analysis

In the second step, class 2 and 3 applications were submitted to dynamic analysis, and additional information about their composition could be retrieved.

As a first result, the existence of relationships between Web Application Pages that had not been retrieved by static analysis was deduced. A first type of relationship was due to code instructions not originally included in the HTML code, but produced at run time by output instructions in server/client scripts (such as the ASP response.write, PHP print, and Javascript write) whose arguments depend on input values. These output instructions were able to produce three different types of relationship between items, such as the Link type between Web pages, the Submit type between forms and server pages, and the Redirect relationship between a Client Script and a Web Page (cfr. the Link, Submit, and Redirect associations in the Web Application model). Sometimes the dynamic Link relationships were also defined by executing a Java applet implementing a menu of hypertextual links. In this case, since the code of the applet was not included in the Web Applications (likewise the code of any Flash or ActiveX object) and could not be parsed, only by execution of the applications could the target pages of the links be identified. Moreover, server scripts including response.redirect instructions were able to produce dynamic Redirect relationships between a Server Script and a Server Page, whose destination depended on the input values.

Table 7.2 reports the count of relationships dynamically retrieved from the Web Applications analysed. In the first column, the type of relationship is reported, while the remaining columns show the count for each Web Application analysed.

Table 7.2: Dynamically retrieved relationships from the analysed Web Applications

Relationship type	WA1	WA2	WA3	WA4	WA5	WA6
Link	0	1	9	0	27	0
Submit	0	32	0	0	0	0
Redirect (in Client Script)	0	0	27	0	0	0
Redirect (in Server Script)	7	0	0	0	0	0

Unlike static analysis, dynamic analysis required greater intervention by the software engineers, who preliminarily had to define the set of input values for executing the Web Applications and, therefore, had to observe the resulting behaviour and record the obtained output. At the moment, these tasks are not supported by the WARE tool. However, the tool can be used to limit the scope of the dynamic analysis to those Web Pages including elements responsible for dynamic results: in fact, these Pages can be detected by the static analysis performed by the tool, and retrieved by querying the tool repository. The total effort (expressed in man hours) required to carry out dynamic analysis for each Web Application is reported in Table 7.4.

The additional information retrieved was, finally, added to the tool WARE Repository manually.

After accomplishing the first two steps of the reverse engineering process, a WAG representing all the retrieved items and relationships could be instantiated for each Web Application. As an example, the graph obtained for the first one, WA1, is reported in Chapter 5.

7.4 Carrying out Clustering

In this phase, the Web Application Connection Graphs (WAG, cfr. Chapter 5) were submitted to the Clustering algorithm described in the previous chapter, so that hierarchies of clusterizations of the components of the Web Applications were proposed and the best of those (according to Quality of Clusterization factor defined in the previous chapter) were chosen and submitted to a validation process.

The clusters identified by the algorithm are composed by strictly interconnected pages, such as pages that actually cooperate in the implementation of a given functionality, or pages that are strongly interconnected by many navigation links (i.e. links just used to make easier the user navigation, such as the back links to go back a previous visited page or the cross links introduced to make a short cut to a longer path, but not necessary to implement a given user functionality).

During the experiments, two applications, WA4 and WA5, that presented frequent navigation links and mechanisms, such as navigation bars, were mainly characterized by loosely cohesive clusters. These clusters were not very useful to support the comprehension of the Web Application behavior. In the case of the WA6, the recovered clustering included more cohesive clusters, since the links in the Web Application Connection Graph were mainly representative of semantic relationships among the pages (i.e. relationships actually needed to implement a user functionality).

The clustering algorithm allowed even the detection of groups of components isolated from the remaining ones. The presence of isolated pages was essentially due either to the presence of incorrectly resolved dynamic links between the pages, either to the presence of unreachable obsolete pages. However, an isolated cluster may correspond to a separate functionality, accessible by another/secondary Home page included in the cluster itself.

In the validation step, the proposed clusters of each Web Application were manually analysed in order to validate them, by distinguishing *valid* clusters from *invalid* ones.

During the validation process, both static analysis and dynamic analysis was carried out. During the static analysis, when the names of the source files of the application were meaningful, rather than generic, or the files were contained in directories with a meaningful name, the software engineer could more easily identify the concept associated with the cluster. During the dynamic analysis, the presence of explicative labels associated with the anchors was useful for understanding the interconnected pages behavior.

The results of the validation step are listed in Table 7.3, which reports, in the upper part, the number of initial clusters proposed by the tool, and the number of spurious, split, incomplete, and accepted clusters. In the lower part of the table, the number of clusters obtained by modifying the ones originally proposed by the tool is listed too.

Table 7.3: Results from clustering validation

	WA1	WA2	WA3	WA4	WA5	WA6
Number of initial clusters	49	101	27	49	31	115
Number of spurious clusters	0	0	1	1	0	0
Number of split clusters	0	0	2	0	5	12
Number of incomplete clusters	8	15	3	0	0	0
Number of accepted clusters	41	86	21	48	26	103
Number of new clusters obtained from the spurious ones	0	0	1	2	0	0
Number of new clusters obtained from the subdivided ones	0	0	5	0	13	31
Number of new clusters obtained by merging incomplete clusters	3	7	1	0	0	0
Number of final clusters	44	93	28	50	39	134

7.5 Discussion

The experiment described in this Section was carried out with the aim of assessing the feasibility and effectiveness of the proposed reverse engineering process. The feasibility of the process was proved by the experiment, since the goal of reconstructing several models of the Web Applications for documenting both their structure and their behaviour was achieved with respect to applications with different characteristics, ranging from Class 1 static applications, to Class 3 dynamic ones.

As to the effectiveness, both the adequacy of the reverse engineering results and the efficiency of the reverse engineering process were assessed.

As regards adequacy, the recovered diagrams were submitted to the judgement of software engineers who were expert in Web Application development and maintenance, in order to assess whether the diagrams described the applications correctly or not. The experts decided the models correctly described both the functional requirements and the structure of the analysed Web Applications, so it is possible to conclude that the proposed approach is adequate.

As to the efficiency of the reverse engineering process, the distribution of the effort required to carry out some steps of the process was evaluated. Table 7.4 lists the effort (in man hours) taken by the experimenters to accomplish the tasks of static analysis, dynamic analysis, automatic clustering and clustering validation, for each application analysed.

Of course, as data in Table 7.4 confirm, the most expensive steps are those requiring human intervention (e.g., dynamic analysis and clustering validation) to analyse and understand the meaning and the behaviour of Web Application items. However, in no case was the human effort required considered to be unacceptable by the experimenters, so it is possible to conclude that the process was also efficient.

Table 7.4: Effort data and computational times from the reverse engineering experiments

Task	**WA1**	**WA2**	**WA3**	**WA4**	**WA5**	**WA6**
Static analysis (seconds)	14	52	19	30	22	62
Dynamic analysis (man hours)	1	6	4	0.5	2	0.25
Automatic clustering (seconds)	2	6	2	1	3	412
Clustering validation (man hours)	10	16	6	5	5	18

In addition, by observing and analysing the activities carried out by the experimenters during the most expensive steps, it was possible identify some features that may negatively affect the analysability of a Web Application. Consequently, some desirable features were identified that programmers should include in the code to facilitate some analysis or comprehension tasks (carried out both manually or automatically).

Some of the difficulties the experimenters encountered were due to Web Application items that are dynamically produced during execution of the application. The analysability of the applications increases when links to Web Pages are explicitly declared in the source code, rather than dynamically generated, since the latter can be resolved only by an expensive dynamic analysis.

Therefore, the use of static links, rather than dynamic ones, in the code of Web Applications, each time it is possible, should be preferred.

Another difficulty that negatively affected comprehension and validation of the clusters was the task of understanding the actual role of hyperlinks in Web Applications, distinguishing navigational links from semantic links implementing a functional dependence between Web Application items. Therefore, it was recognized that some annotation in the code explicitly describing the role of the link would be useful. As an example, the HTML language provides the ‘name’ attribute for the ‘anchor’ tag implementing hyperlinks, that may be used to specify whether a link is a navigational (cross or back) link or a semantic link. Figure 7.2 shows a fragment of HTML code including examples of anchors described with the ‘name’ attribute. In the example, the ‘name’ attributes with values ‘crossA’ and ‘crossC’ show that the first two anchor tags implement navigational links for reaching the HTML pages ‘A.html’ and ‘C.html’ directly by a shortcut. The third ‘name’ attribute with the value ‘backHome’ indicates that this anchor tag implements a shortcut to the Home page. The last anchor tags do not include the ‘name’ attribute, and therefore implement a semantic link.

```
………..
<title> Argument B </title>
…
<a href=“A.html” name=crossA > Argument A </a>
<a href=“C.html” name=crossC > Argument C </a>
<a href=“index.html” name=backHome > Home Page </a>

<a href=“B1.html”> Argument B.1 </a>
<a href=“B2.html”> Argument B.2 </a>
………
```

Figure 7.2: An example of HTML code including ‘named’ links

In addition, using a suitable internal documentation standard to annotate each main entity of the Web Application would be another desirable programming practice, which would ease the comprehension tasks. As an example, a brief description of the page meaning/behaviour, its input/output data, and its interconnections with other Web Application components should be introduced as a formatted comment in the page, as shown in Figure 7.3. This information could be automatically captured by a static analyser, and provided to support any Concept Assignment Process involving that page.

Of course, the experimenters also recognised that code analysis tasks were simplified when each file of the application was associated with a self-explanatory name, instead of a cryptic and anonymous one.

```
<%@ Language=VBScript %>
<% Option Explicit
%>
<HTML>
<HEAD>
<TITLE> check </TITLE>
…………..
<META NAME="Purpose" CONTENT="This page checks Login and Password of a Teacher, then
it redirects to Teacher Home Page">
<META NAME = "Incoming Links from Pages:" CONTENT = "/autenticazionedocente.html">
<META NAME = "Outgoing Links to Pages:" CONTENT = "/autenticazionedocente.html,
/areadocente.html">
<META NAME="Input Parameters" CONTENT="login,password">
<META NAME="Output Parameters" CONTENT="">
<META NAME = "Session Variables" CONTENT = "loginOK, matricola">
<META NAME="Included Modules" CONTENT="">
<META NAME="Database" CONTENT="../basedatisito.mdb">
<META NAME="Images" CONTENT="bgmain.gif">
</HEAD>

<BODY>
……………..
```

Figure 7.3: An example of a web page with commented lines providing internal documentation

Finally, the experimental results demonstrated the feasibility and effectiveness of the process for reverse engineering Web Applications with different characteristics, including both purely static Web Applications, and Web Applications with dynamic elements. Of course, the validity of such experimental results is limited by the reduced number of applications considered, and the reduced number of process variables observed during the experiment (i.e., adequacy and efficiency). Further experimentation is, therefore, required in order to extend the validity of the experiment, by investigating further research issues.

Chapter 8: Abstraction of Business Level Diagrams

In this chapter a method to recover UML diagrams at business level of a Web Application is described. Heuristic algorithms and methodologies to recover business level class diagrams, use case diagrams and sequence diagrams are proposed. The results of the experimentation of the method on a medium sized Web Application are reported.

8.1 Introduction

In the previous chapter, methods and techniques to recover diagrams representing the structural view of a Web Application have been proposed and described and the results of a validation experiment involving them have been reported. These representations are very useful supports to carry out a maintenance intervention with a limited impact on the application, but they are less useful in the case of a maintenance intervention involving a wider set of Web Application components, such as a reengineering intervention or a migration. If these tasks must be faced, more abstract representations may be needed.

Business level UML diagrams are typical products of the high-level development phase of a software system. They describe conceptual components (business objects) from the domain of the problem addressed by the application, their mutual relationships and the basic functionalities the system must provide.

In literature, approaches extracting business level object oriented models from procedural legacy systems have been used to migrate the systems towards object-oriented platforms with the support of wrapping technologies [Del97]. Moreover, objects from an object-oriented conceptual model have proved useful to support a systematic reuse, since each validated object represents a reusable component that can be integrated in the production of new systems [Can96].

Further works in literature, addressing the recover of objects and object oriented models from traditional software are described in [Can96], [Cim99], [Gal95], [Geo96], [Liu90], [Liv94], [New95] and [Yeh95].

Defining and validating similar approaches in the context of Web Applications represent a relevant research issue. The abstraction of business level diagrams of a Web Application is a very difficult and expensive task. In particular, scripting languages used to implement Web Applications are object-based languages, instead of effective object orient languages. So, the abstraction of object oriented business level class diagrams is difficult.

In this chapter a method to abstract business level UML class diagrams, use case diagram and sequence diagrams is described. This method needs information extracted by the tool WARE (cfr. Chapter 6).

8.2 Recovering Business Level Class Diagrams

In this chapter a method that is similar to the one proposed in the context of traditional software in [Del97] and [Dil00] is described. That method is based on a Reverse Engineering process including three main steps:

- Identification of candidate classes and their attributes;
- Association of methods to candidate classes;
- Identification of relationships between classes.

In the following the terms class and object are used as synonyms. In traditional software, the identification of the candidate objects and of their attributes is made by looking for groups of logically related data making up the state of objects. This search is usually based on those language mechanisms that allow groups of related data implementing a relevant concept, either from the domain of the application or from domain of the solution, to be defined in the code. These mechanisms include those for the definition and use of data structures such as records, user data types, and table schemas in databases. Moreover, the identification of object operations, i.e., methods, is centred on suitable pieces of code (such as programs, subroutines, slices) that can be associated with the candidate objects according to specific coupling criteria. Finally, specific heuristic criteria, such as those defined in [Dil00], can be used to define the relationships between objects.

Several questions have to be addressed when trying to use these methods in the Web Application context.

The first problem regards the identification of objects and objects' attributes in a Web Application, since the selection of the mechanisms that are generally used for implementing groups of related data is not obvious for Web Applications. Most web technologies and languages (such as HTML, ASP, PHP, VBS, JSP, etc.) provide syntactic constructs for declaring data groups like RecordSets, or Collections, or Classes, but some of them are not used frequently. Moreover, since a Web Application is usually implemented as a multi-tier system, with a database server implementing one tier of the architecture, a simple Web Application code analysis may not allow the identification of the persistent data stores and of the data store schemas. In this case, indeed, such data store descriptions may be deployed on a different tier of the application, and may be inaccessible, such as in the case of a Web Application that makes the services of an existing information system available on the net, implementing the front-end of the system.

A second question with Web Applications regards the identification of the chunks of code implementing the object methods, since the pieces of code that are usually considered for traditional software (e.g., programs, subroutines, slices, etc.) may not be meaningful for Web Applications. In this case, possible functional units to be considered should include web pages, script blocks inside a page, functions in a page, depending on the requested degree of granularity. Moreover, appropriate criteria for electing these chunks of code to object methods and associating them with the correct object should be accurately defined, as well as suitable rules for defining the possible relationships between the recovered objects should be defined.

In the following subsections, a technique to overcome these problems is proposed.

8.2.1 Identifying candidate classes and their attributes

A method identifying candidate classes from a Web Application is proposed. This method includes two steps. The first step comprises the identification of relevant groups of data items from the Web Application code; in the second step, an automatic procedure is executed to define the candidate classes from the list of groups identified in the previous step.

The elements of interest for identifying the attributes of candidate classes in a Web Application are groups of data items that are involved in the same user input/output operation (such as data displayed in input/output HTML forms, or HTML tables), or in the same read/write operation on a data store (such as an ASP Recordset, or an array of heterogeneous data in PhP language), or the data set involved in a database query operation. In addition, data groups that are passed throw distinct pages or instances of Classes used in the pages are taken into account.

The rationale behind this choice is that the set of data items that a user inputs by an input form, or that are shown to a user by an output form, usually represents concepts of interest for the user in the domain of the application. Analogously, data items that are read from, or written to a persistent data store may be representative of meaningful concepts of the business domain.

In a preliminary step of the process, a static analysis of the Web Application source code is required for retrieving these data groups and their references in the code. Each data item of each group is associated with the identifier used to reference it in the code.

Therefore, the items from the groups are submitted to a refinement step aiming at solving the problems of synonyms (i.e., identifiers with different names but the same meaning) and homonyms (i.e., identifiers with the same names but different meanings). Synonym identifiers must be assigned with the same unique identifier. Homonym identifiers must be associated with distinct names. In addition, while carrying out synonyms & homonyms analysis a meaningful name, synthesizing the meaning of the data according to the rules of the business domain, is assigned to each data item. Of course, both the definition of the names, and synonyms and homonyms analysis are based on code reading and inspection of available documentation and are human intensive tasks.

At this point, each data group may be considered as the attributes of a potential class and a validation should be carried out for distinguishing valid business classes, i.e. classes actually associated with a concept from the business domain, from invalid ones. This validation is usually accomplished manually.

In order to reduce the effort required by this analysis, the method exploits an automatic procedure that analyses the data groups in order to identify the ones that are more likely to represent meaningful classes. Only the selected data groups are, therefore, submitted to the validation process. This procedure, called Produce_Candidate_Objects, is based on heuristic criteria and is illustrated in Figure 8.1, where:

- GList is the list of data groups;
- g is the generic group in the GList;
- Card(g) is the cardinality of the g group;
- Nref(g) is the number of references to the g group;
- a is a generic data item in g;
- CAND is the list of candidate objects;
- C is a generic candidate object from CAND;
- SORT (A, K) is a function for sorting the list A according to the criteria described in K;
- TOP(A) is a function for accessing the top element of a list A;
- REMOVE(A, x) is a procedure for removing an item x from the list A;
- INSERT(A, x) is a procedure for inserting the item x in the list A;
- ADD(C, h) is the procedure for adding all the item of a group h in the C group.

The procedure Produce_Candidate_Objects analyses the input list Glist of data groups and produces the output list CAND of candidate classes.

The first heuristic rule implemented by this procedure states that the more the references of a same data group in the code, the greater the likelihood that it represents a meaningful concept. The second heuristic rule establishes that groups with a small size may represent more simple and atomic concepts than larger groups, and larger groups may represent more complex concepts made up of joined smaller groups.

According to these two rules, groups from the Glist are preliminarily arranged in descending order with the number of references of each g group, and in ascending order with the cardinality of each group. This order is produced by the procedure SORT whose output consists of the ordered list ‘OrdList’ of data groups.

```
Procedure Produce_Candidate_Objects (in: GList; out: CAND);

BEGIN
OrdList = SORT (Glist, Descending on $N_{ref}(g)$ AND Ascending on Card($g$) );
CAND = $\emptyset$;
WHILE (OrdList $\neq \emptyset$ OR ($\cup_i a \in C_i \equiv \cup_i a \in g_i$ )) DO
      $h$=TOP(OrdList);
      IF ($\exists$ a$\in h$: a$\notin C$ $\forall$ $C$ $\in$CAND) THEN
            IF (!$\exists$ $C\in$CAND: $C \subseteq$ $h$ THEN
                  INSERT(CAND, $h$)
      ELSE
                  $\forall$ $C_i$ $\in$ CAND: $C_i \subseteq$ $h$ DO
                        BEGIN
                        $k = h - \cup_i C_i$;
                        ADD($C$, $k$);
                        END
            END IF
      END IF
      REMOVE (OrdList, $g$);
END WHILE
END
```

Figure 8.1: The procedure generating the list of candidate objects

Starting from the top group in OrdList, the procedure analyses each group and, if a group comprises at least a new data item not yet included in any other group in CAND, it is inserted in the CAND list of candidate objects. OrdList is examined until it includes at least a group, or until the union set of all the data items of the candidate objects in CAND and the union set of all the data items of the groups in Glist are equal.

When a group h from OrdList includes all the data items making up one or more groups C_i in CAND, only the k data items in h that are not yet included in any group of CAND are added to the C_i groups whose elements are all included in h. The reason is that the group h is likely to represent a composite concept produced by a logical link among the C_i groups. The attributes that are added to the C_i groups are necessary to record this link, which is used to deduce relationships between objects, according to the method proposed in Section 8.2.3.

As an example of this case consists in a data group associated with a report showing information about some distinct objects of the application domain.

At the end of the procedure, the CAND list will include the set of data groups that have been selected as candidate objects. Each group in the CAND list will have to be assigned with a meaningful name describing the concept it represents. The data items of each group will make up the attributes of the object, e.g., its state. The attributes that appear in more than one candidate object are analysed in a successive step according to the method presented in the next Section 8.2.3.

For the sake of precision, CAND is the set of candidate classes from which business objects can be instantiated.

As an example of the application of the proposed method, let's consider the following OrdList:

OrdList = {G1=(a,b,c), G2=(d,e,f), G3=(a,b,c,d,e,f,g), G4=(a,b,c,m,n), G5=(a,b,q,r), G6=(d,f,s,t,u)}

The final CAND list will contain the following groups of candidate business objects:

CAND = {(G1,m,n,g), (G2,g), G5, G6}.

8.2.2 Associating methods to classes

The identification of methods to be associated with classes essentially depends on two factors: the degree of granularity of the chunks of code to be considered as potential methods, and the definition of a criterion for assigning a potential method to a class.

In a Web Application the search for class methods can be centred on chunks of code with fine granularity levels, such as inner page components like scripts or modules included in pages, or on coarse-grained components, like pages or groups of pages. Finer the granularity level, greater is the effort required for extracting the component from the code, and reengineering it as an object method.

The technique proposes to consider physical Web pages (e.g., Server pages, Client pages) as potential methods to be associated with the candidate objects of the Web Application, rather than inner page components. This choice allows a reduction of the effort required for the object oriented reengineering of the Web Application.

As to the problem of defining a criterion for associating methods to objects, a technique that aims to minimize the coupling between distinct objects is adopted.

Measures of coupling between pages and objects are computed based on the accesses of pages to the candidate object. A page accesses a candidate object when it includes instructions that define or use the value of some object attribute. It is assumed that accesses made to define the value of some object attributes produce a greater coupling than accesses made to use some object attributes.

Therefore, minimization of the coupling between objects is achieved by associating each page to the object it is most highly coupled with. In particular, if a page accesses exclusively one object, it is assigned as a method of that object. If a page accesses more objects, it is assigned to the object it accesses prevalently. In this second case, the accesses of the page to the other objects can be considered as messages exchanged from the object the page has been assigned with, to the other objects.

To implement the criterion for associating pages to objects, the following definitions are introduced:

- M={mi}, is the set of pages (e.g., potential object methods - in the following the terms page and method are used as synonyms);
- c is the generic element from the CAND list of candidate objects;
- αdef is a positive number expressing a weight associated with each define access made to an object;
- a_{use} is a positive number expressing a weight associated with each use access made to an object;
- $\alpha_{def} > \alpha_{use}$;
- $N_{def,m,c}$ is the number of accesses of type define made by a page m to the object c;
- $N_{use,m,c}$ is the number of accesses of type use made by a page m to the object c.

Moreover, the function Acc(m, c) that expresses the weighted number of accesses made by m to c is defined as:

$$\text{Acc (m, c)}= \alpha_{def} * N_{def,m,c} + \alpha_{use} * N_{use,m,c} \qquad (1)$$

Said MAX [X] a function that returns the maximum value from a set X of values, the following criterion is used to assign a method m to a class c:

$$\text{m is assigned to } c \in \text{CAND} \Leftrightarrow \text{Acc}(m , c) = \text{MAX}_{cj \in \text{CAND}} [\text{Acc}(m , c_j)] \qquad (2)$$

that is, the page m is assigned, as a method, to the class c iff c is the class such that the number of weighted accesses made by the page m to c is greater than all the other weighted accesses m makes to other classes.

When the criterion (2) is satisfied by two or more classes, the intervention of a software engineer is required to establish the correct assignment of m with one of the classes.

Pages that do not make access to any objects are considered as coordinating modules controlling the executions of other methods (in a Web Application this page usually corresponds to home pages, or pages that address the user navigation along the Web Application).

8.2.3 Identifying relationships between classes

The candidate classes in the CAND list may include common attributes: these attributes will indicate potential relationships among the involved classes. These relationships are depicted as UML association relationships.

For each set of classes having common attributes, a UML association is established between them. Each common attribute is assigned just to one class of the set, and all these classes are linked by an association relationship. The software engineer intervention is required to establish the correct assignment of the attributes to the object.

The case of pages accessing more than one class will originate additional relationships. More precisely, the accessing page is assigned with the object it is most highly coupled with, according to the criterion (2), while a relationship is defined between the class the page is assigned to, and each remaining class the page accesses. Also this kind of relationships is depicted as UML association relationships.

In a successive refinement step, the recovered association relationships may be analysed in order to assess whether a class with specialization or any aggregation/composition relationship can substitute them.

Finally, a UML Class diagram will represent the recovered classes, their attributes and methods, and the relationships among them.

8.3. Recovering UML Use Case and Sequence diagrams

The Reverse Engineering methods proposed for recovering use cases and sequence diagrams from the code of a Web Application are presented in the following.

8.3.1 Recovering Use Case Diagrams

The problem of recovering use cases from the source code of a Web Application is solved on the basis of the results obtained with the clustering approach presented in chapter 5. That approach provided a set of clusters validated by an expert. The expert has to identify the user functionality each cluster is responsible for. As introduced in chapter 5, this task is a human intensive task, because it needs a comprehension of the semantic of the components of the Web Application under analysis. However, approaches to recover, automatically, information about the semantic of these components has been proposed and described in the following chapters.

Groups of validated clusters can be associated with potential use cases of the Web Application, and a use case model can be reconstructed for describing the external behaviour offered by the application to the end users.

In order to obtain a Use Case Diagram, relationships among use cases may be deduced analysing the links between corresponding clusters (this functionality is also supported by the tool WARE: cfr. Figure 6.9). As an example, if a cluster associated with a use case 'A' is linked to just another cluster associated with the use case 'B', a candidate <<include>> relationship from use case 'A' to use case 'B' may be proposed; if a cluster is linked to more other clusters, a possible <<extend>> relationship among the use case corresponding to the former cluster and the remaining ones may have to be considered. However, these indications provide the reverse engineer just with simple suggestions about how use case diagrams can be drawn. Finally, actors in the Use Case diagram are linked to use cases associated to clusters including any page requiring data input from a user (e.g., Web pages including forms), or database connections.

8.3.2 Recovering Sequence Diagrams

As to the UML Sequence diagrams abstraction, for each use case (i.e., validated cluster) it is possible to produce a Sequence Diagram whose objects will derive from classes associated with cluster's pages, while interactions among objects are deduced from the accesses that a page assigned as a method to a given Class makes to other Classes.

A process, based on heuristic criteria, obtains the recovering of a sequence diagram. This process includes the following steps:

- Draw a sequence diagram for each identified use case;
- Identify the pages composing the cluster (or group of clusters) associated to a use case;
- For each page in the cluster, identify the class which the page was assigned to as a method: for each identified class put an object in the diagram;
- If a page assigned to an object of the Class A makes accesses to other objects of the Classes B_i, draw an interaction between the object of A and each object of B_i;
- Assign each interaction with a meaningful name (corresponding to the name of the invoked method) and define the list of parameters exchanged between the objects (deduced by analysing the data flow between the Web Application interconnected items);
- Draw an interaction between an object and an actor if that object was assigned to a page including input or output forms;

- If there is a relation between pages assigned to the same class, it is considered as a call between methods of the same Class, and therefore a 'Message to self' is drawn on the object life-line.

According to the UML notation, in a Sequence Diagram the correct temporal sequence of the events (e.g., interactions between objects) can be deduced by reading the flow of interactions from upside to downside in the diagram. In the recovered Sequence Diagrams, this temporal sequence may be reconstructed by statically analysing the control flow in the source code. As a preliminary result, the temporal sequence may correspond to the lexicographic statement sequence. Therefore, dynamic analysis can be used to refine the statically defined temporal sequence.

8.4. A case study

In order to assess the effectiveness of the proposed techniques, several controlled experiments were carried out involving some real-world Web Applications. An example of these experiments is shown in this section, where the results obtained in a case study are presented and discussed.

In the case study, the Web Application, designed to support the activities of undergraduate courses offered by a Computer Science Department analysed in Chapter 6, has been considered.

During the first step of the recovery process, the code was analysed for identifying data groups, according to the method presented in Section 8.2.1. Groups of data items involved in I/O forms, in read/write operations on persistent data stores, and so on, were looked for, and the number of their occurrences in the code was evaluated. The WARE tool (cfr, Chapter 6), that statically analyses the source code of Web Applications, was used to support this task. The analysis retrieved 128 references to data groups including a total of 485 data items.

Therefore, synonyms/homonyms analysis was carried out and each data item was assigned with a meaningful name. This task was accomplished by reading and inspecting the code, and analysing the Web Application during its execution. At the end of synonyms/homonyms analysis, just 43 different data groups including a total of 26 different data items were defined.

These data groups were submitted to the candidature procedure Produce_Candidate_Objects that automatically selected 8 candidate classes, including a total of 38 attributes (of course just 26 were different attributes). The candidate classes are listed in Table 8.1, where the set of attributes of each class is reported. Some candidate classes presented common attributes that were exploited to identify the relationships among the classes in the third step of the process.

In the second step of the process, the accesses each page made to the candidate classes were analysed. Twenty pages that did not reference any data group were detected, since they had only presentational or navigational purposes.

The remaining pages were assigned as methods of the candidate classes according to the method described in section 8.2.2. More precisely, for each page the function Acc(m,c) was computed with respect to all the accesses to all the classes the page made, and each page was assigned to a class according to the criterion (2) in section 8.2.2.

For evaluating the function Acc(m,c), different values had to be defined for the α_{DEF} and α_{USE} weights. During the experiments carried out with different Web Applications, several values for α_{DEF} and α_{USE} were tried in order to detect the ones that produced the best results (i.e., the best assignment of methods to classes, according to the judgment of an expert). The best results were achieved for $\alpha_{DEF} = 1$ and $\alpha_{USE} = 0.6$, and this couple of values were used in the case study too.

Table 8.1: Candidate classes produced by the Candidature Procedure

Candidate classes and corresponding attributes	
Student	(Student name, Student surname, Student code, Student email, Student phone number, Student password)
Teacher	(Teacher name, Teacher surname, Teacher email, Teacher phone number, Teacher password, Teacher code)
Exam Session	(Exam date, Exam time, Exam classroom)
Tutoring	(Tutoring date, Tutoring start time, Tutoring end time, Course code, Course name)
Course	(Course code, Course name, Course academic year)
Tutoring Request	(Student name, Student surname, Student code, Tutoring request date)
News	(Course code, News text, News number, News date, Teacher code)
Exam Reservation	(Student code, Student name, Student surname, Course code, Exam date, Exam reservation date)

In the third step of the process, associations between classes had to be defined. In a preliminary phase, candidate classes were examined in order to find classes with common attributes. This task was carried out with the support of an automatic procedure.

For each set of classes including a same attribute, an association between these classes was established, and each common attribute was assigned to the class that was better characterized by that attribute, depending on the experimenter's judgment.

Other relationships between classes were defined on the basis of accesses to the attributes of other classes that were made by a method assigned to a given class. A relationship was established between the class the method was assigned to, and the remaining classes whose attributes were referenced by this method.

Figure 8.2 shows the UML class diagram representing the resulting business object model of the Web Application, while Table 8.2 reports the list of the Web Application pages assigned as methods to each identified class.

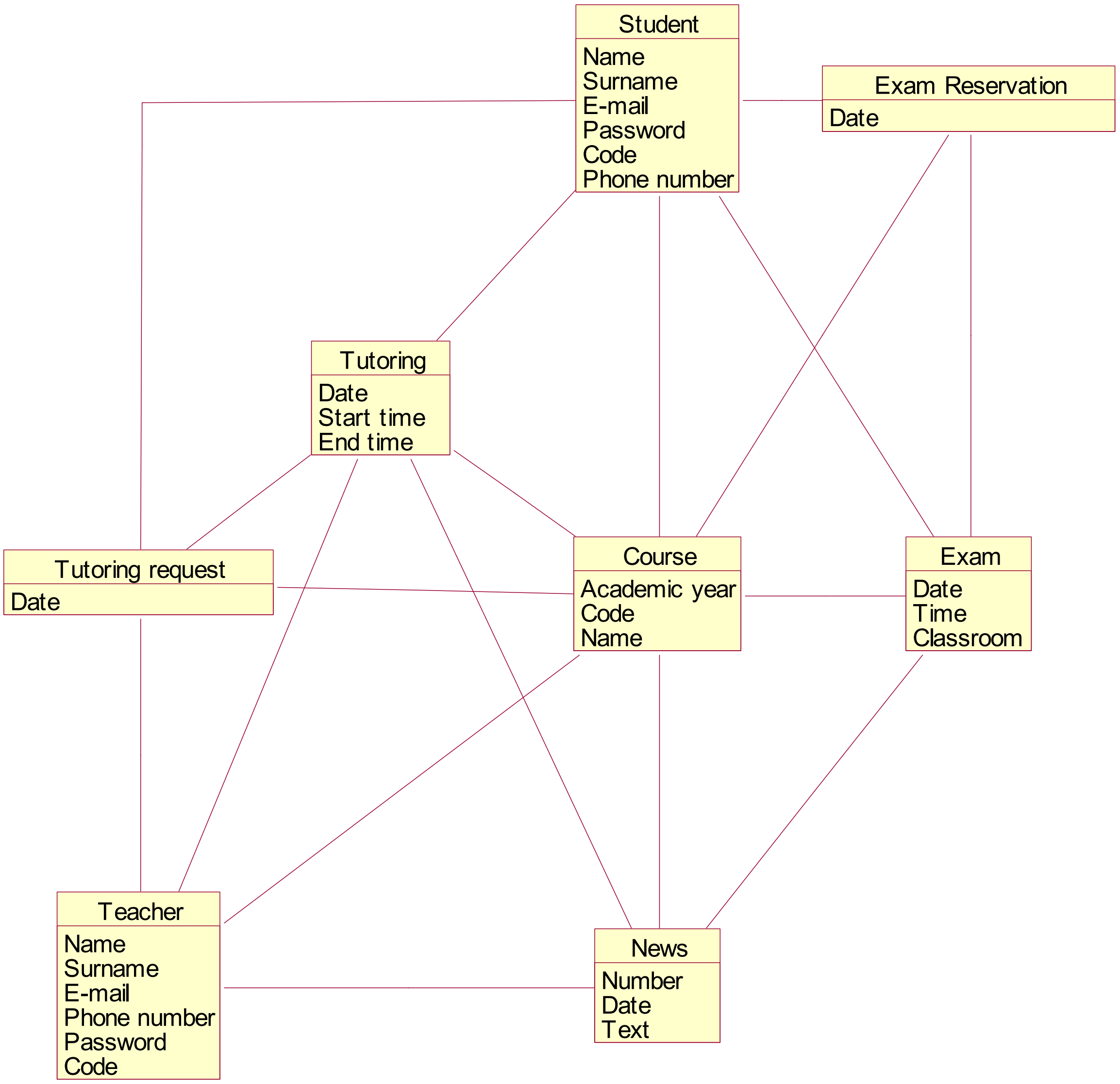


Figure 8.2: UML class diagram representing the resulting business object model of the Web Application

Table 8.2: Web Pages implementing Class methods

Class	Web Application pages assigned as Class methods
Student	chiediPass.html, cancellareg1.asp, cancellareg2.asp, visualizzareg2.asp, prenotatiapp3.asp, FormCancPreRicevimento.asp, PreRicev3.asp, iscrivicorsi.asp, regstudente.asp, regstudente.html, autenticazionestudente.asp, checkStudente.asp, modificaiscriz2.asp, modificastud.asp, modificastud2.asp
Teacher	autenticazionedocente.html, check.asp, registradoccorso.html, registradocente.asp, eliminadoc.asp, eliminadoc2.asp, FormPreRicevimento.asp, modificadoc.asp, modificadoc2.asp
Exam Session	listaappelli2.asp, listaappellistud.asp, cancellaapp2.asp, cancellaapp3.asp, prenotatiapp2.asp, modappello2.asp, modappello3.asp, modappello4.asp, prenotaesame2.asp, prenotaesame3.asp, prenotaesame4.asp, insappello.asp, insappello3.asp
Tutoring	CancPreRic.asp, CancRic.asp, CancRic2.asp, FormCancRicevimento.asp, FormPreRicev2.asp, FormModRicevimento.asp, ModRic.asp, ModRic2.asp, VisListaRic.asp, VisPreRic.asp, FormInsRicevimento.asp, insRicevimento.asp
Course	aggiungicorso.asp, aggiungicorso.html, listaappelli.asp, visualizzaapp.asp, FormVisBacheca.asp, FormCancAvviso.asp, cancellaapp.asp, cancellacorso.asp, cancellacorso2.asp, cancellareg.asp, visualizzareg.asp, prenotatiapp.asp, FormVisPreRicev.asp, sceltacorsi.asp, modappello.asp, prenotaesame.asp, modificadcnz.asp, modificadcnz2.asp, modificaiscriz.asp, insAvviso.html, regcorso.asp, registracorso.html.
Tutoring request	-
News	VisBacheca.asp, CancAvviso.asp, DelAvvisi.asp, modappmsg.html, modappmsg1.asp, FormInsAvviso.asp, insAvviso.asp
Exam reserv.	-

In order to abstract the use cases of the application, the clustering method proposed in Chapter 5 was applied. As a result of the automatic clustering, 44 valid clusters were recovered that were submitted to a validation step. Table 8.3 reports the list of validated clusters.

The validated clusters were initially associated to use cases and a top use case diagram was produced. Figure 8.3 reports an excerpt of this diagram, showing the <<extend>> and <<include>> relationships between use cases that have been deduced using the criteria proposed in Section 8.3.1.

For each of these use cases, a Sequence Diagram was drawn.

As an example of these diagrams, Figure 8.4 reports the one derived for the use case ‘Course insertion’, that inserts in the database a new Course taught by a given teacher. The cluster corresponding to this use case includes two pages: the client page ‘aggiungicorso.html’ and the server page ‘aggiungicorso.asp’, both assigned to the object Course. The former page includes a form requiring input of data by a user, therefore an interaction of this page with an actor was drawn. The page ‘aggiungicorso.html’ does not refer to any other object, but is linked to the page ‘aggiungicorso.asp’ by a submit operation: this was modeled in the sequence diagram by the self-interactions on the Course object. The page ‘aggiungicorso.asp’ makes a reference to some attributes of the object Teacher, then an interaction between the Course and Teacher object was also drawn.

Table 8.3: Validated clusters with their descriptions

Cluster Id.	Description
1	Home page
2	Teacher login
3	Teachers function menu
4	Teacher area entry point
5	Teacher area frameset
6	Tutoring date insertion (available to teachers)
7	Tutoring date deletion (available to teachers)
8	Tutoring area entry point (available to teachers)
9	List of registered students for tutoring (available to teachers)
10	Teacher registration
11	Assign a course to a teacher
12	Teacher data deletion
13	Undergraduate course insertion
14	Teacher and course data update
15	Teacher and course management area entry point (available to teachers)
16	Course deletion (available to teachers)
17	Student data deletion (available to teachers)
18	Students enrollment area entry point (available to teachers)
19	List of enrolled students display
20	Exam schedule modification
21	Insertion of a new date in the exam schedule
22	Deletion of a date from the exam schedule
23	List of students registered to an exam (available to teachers)
24	Exam schedule area entry point (available to teachers)
25	Exam schedule list display (available to teacher)
26	Bulletin board area entry point
27	List the news in the bulletin board
28	News insertion in the bulletin board
29	News deletion from bulletin board
30	Students area entry point
31	Lost password request (available to students)
32	Student Enrollment Area entry point
33	Student data update (available to students)
34	Student course enrollment
35	Exam area entry point (available to students)
36	Exam schedule listing (available to students)
37	Register a student to an exam
38	Tutoring request area entry point (available to students)
39	Tutoring requests insertion (available to students)
40	Tutoring date deletion (available to students)
41	Tutoring date update (available to teachers)
42	Utility module 'adovbs.inc'
43	Obsolete functionality (old version of Insertion of a new date in the exam schedule)
44	Work in progress page for not yet implemented functionalities

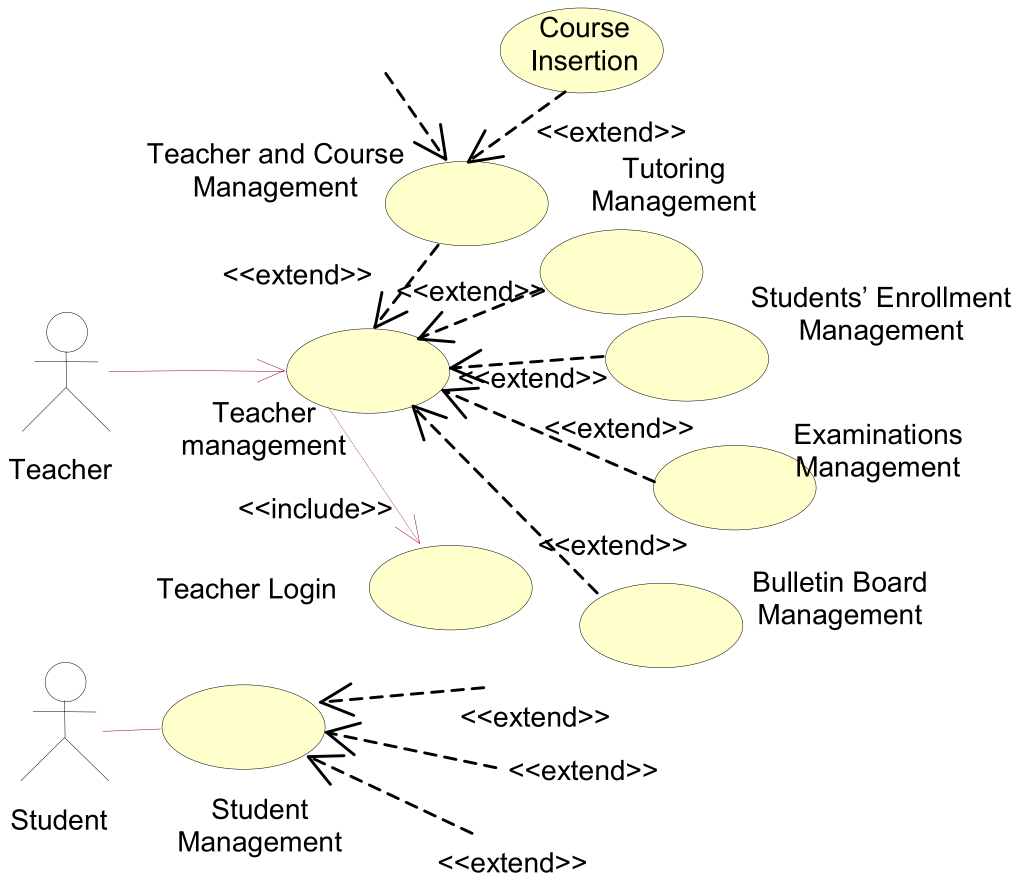


Figure 8.3: An excerpt of the Use Case diagram recovered from the Web Application

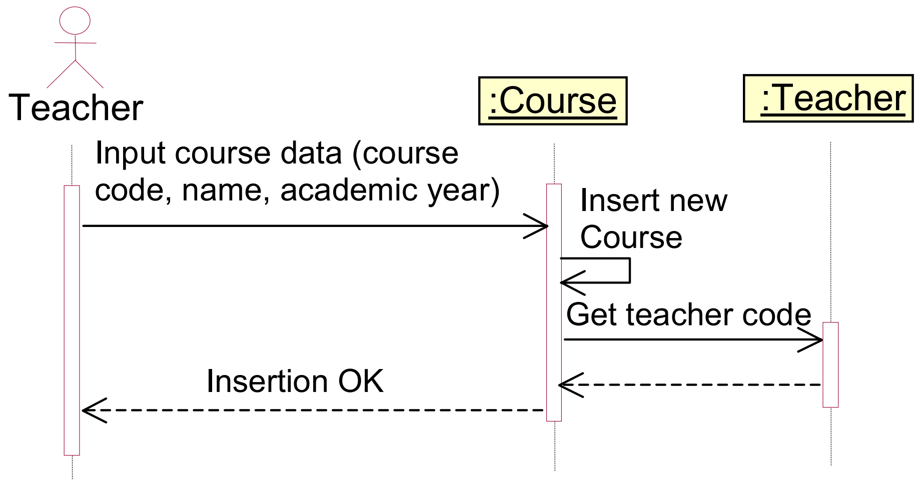


Figure 8.4: The UML sequence diagram representing the interactions for the use case 'Course insertion'

All the recovered diagrams were validated by submitting them to the judgment of the software engineers that had developed the Web Application. These diagrams were compared against the original diagrams designed by the software engineers: no substantial differences were found between the recovered diagrams and the original ones.

It was concluded that the recovered diagrams represented the same concepts and the differences with the original ones were mainly due to implementation details. Similar results were obtained in the other experiments that have been carried out: these results showed us the effectiveness of the method.

Further details about the methods and the techniques presented in this chapter can be found in [Dil03] and [Dil03b].

8.5 Future Works

In future work, the definition of criteria for a further automation of the model reconstruction will be addressed, as well as the investigation on possible approaches for identifying UML aggregation, composition, or generalization-specialization relationships between classes will be carried out. A wider experimentation involving more complex Web Applications, implemented with different technologies, will be moreover carried out, in order to extend the validity of the proposed approaches.

Chapter 9: Concept Assignment for client pages

In this chapter, a method providing automatic support in the assignment of concepts to Web documents is presented. This method is based on Information Retrieval principles. Descriptions of the heuristic algorithms defined and of the tool realized to support the method are provided in this chapter with the results of some experiments carried out to validate it.

9.1 Introduction

The recovering of semantic information about the functionality realized by the components of a Web Application cannot be addressed only analysing the structural information of its components (i.e. Web pages, server and client script modules, etc.). In the experiment presented in the previous chapters, human expert, without any automatic support, has addressed this task, and the effort of clustering validation and concept assignment was the most expensive step of the Reverse Engineering process (cfr. Table 7.4). So, in the following chapters techniques to recover, automatically, information about the semantic of the components of a Web Application are proposed.

In this chapter the textual information contained in these artefacts are analysed, with the aim to propose a concept describing the contents of a client page.

The term *concept assignment* was introduced by Biggerstaff et al. [Big93] to describe the problem of assigning a synthetic description regarding the computational intent of segments of source code. A *concept* was defined as a description at a higher level of abstraction than the source code. Several approaches to provide automatic support to the concept assignment process have been proposed for traditional (i.e. not web based) applications, such as the one described in [Gol01] where a hypothesis-based concept assignment method for COBOL programs is proposed.

Differently from traditional software applications, web based applications are characterised by a large amount of textual information contained in the pages making up them; usually, this information is displayed in the pages forming the user interface to improve the usability of the application by describing possible uses of the application, available user functions, arguments discussed in the page and so on. Moreover, special edit formatting (as bold, italics, underlined characters) is usually used to highlight some piece of the displayed text related to the most relevant information contained in the page.

Both the text contained in the web pages, and the editing format used to display it, are a relevant source of information that can be used in a concept assignment process involving a Web Application. This method is exploited to automatically support the identification of concepts to be associated with artifacts recovered by reverse engineering the Web Application. The motivation of this choice is based on the fundamental hypothesis that the concept to assign to a web page displayed to a user is contained inside the text of the page itself.

There are several solutions described in the literature to the problem of obtaining a synthetic description of a given text document. In particular, Information Retrieval (IR) methods are based on the analysis and elaboration of the textual content of the document in order to be able to extract few relevant terms allowing the classification (i.e. a synthetic description) of the document. These IR methods may be classified with respect to the granularity of the analysed piece of text. Some methods consist of the analysis of each word of the document, determining an array of terms [Har92] on which some appropriate elaborations are made to identify the terms usable to classify the document. Some other methods consist of the analysis of single words and bi-grams (sequence of two consecutive words in the text) [Ton03]. These methods may be useful to find index words for a document, or to evaluate the statistic similitude between two documents.

9.2 A specialized conceptual model describing a Web Application

A conceptual model describing the components of a Web Application and the relationships among them has already been reported in Chapter 3. In that model the attention was focused on the structural aspects of a Web Application. In this chapter the part of the conceptual model describing the textual content of the Web Application is focused.

Web pages contain the information to be shown/provided to/from a user, being the information made up by text, images, multimedia objects, etc., embedded in the page itself, or retrieved/stored from/in a file or database. Thus, in a Web page, it is possible to distinguish a *control component* (i.e., the set of items - such as the HTML code, scripts and applets - determining the page layout, business rule processing, and event management) and a *data component* (i.e., the set of items - such as text, images, multimedia objects - determining the information to be read/displayed from/to a user).

In particular, just the HTML code is considered as the control component while for the data component are considered the pieces of text, i.e. any sequence of words included by two HTML tags, that are displayed to a user using the text formatting defined by the HTML tags.

Of course, a HTML tag in the Control Component may be *nested* into another one, as well as a tag may have a list of attributes. Moreover, a word may have synonyms and may derive from a stem (e.g. the masculine form of a word is the stem of a feminine one, the singular form of a word is the stem of a plural word, the infinite tense of a verb is the stem of any other tense, and so on); in addition a word may be a stopword, i.e. a word (such as articles, conjunctions, prepositions, exclamations, pronouns) that does not provide any significant contribution to the meaning of the sentence it belongs to.

Each piece of text from a page is characterised by its meaning and, therefore, can be associated with a concept. Analogously, a Web page can be associated with a concept describing its meaning: this concept can be selected from all the concepts associated with the pieces of text included in the page.

These aspects of a Web page are represented by the UML class diagram shown in Figure 9.1, providing the conceptual model of a Web Application considered in the remaining part of this chapter.

9.3 Identifying the Concepts

The identification of the concept to assign to a Web page is based on the following hypotheses:

- Client Pages are used to allow the interaction of the users with the Web Application;
- Client Pages contain textual information to communicate the aim/scope/topic of the page to the user;
- some special editing formats (e.g. bold, italics, underlined, character size, etc,) are used in the page to highlight some pieces of the text (i.e. some words) that would summarise the aim/scope/topic of the page;
- the pieces of text declaring the main aim/scope/topic of the page are, usually, at the top of the page;
- the text format is defined by appropriate HTML tags and attributes.

Thus, the particular editing format and position of the text in a page represent information that can be exploited to identify automatically those words that may describe conceptually a page, i.e. the Concept to assign to a page.

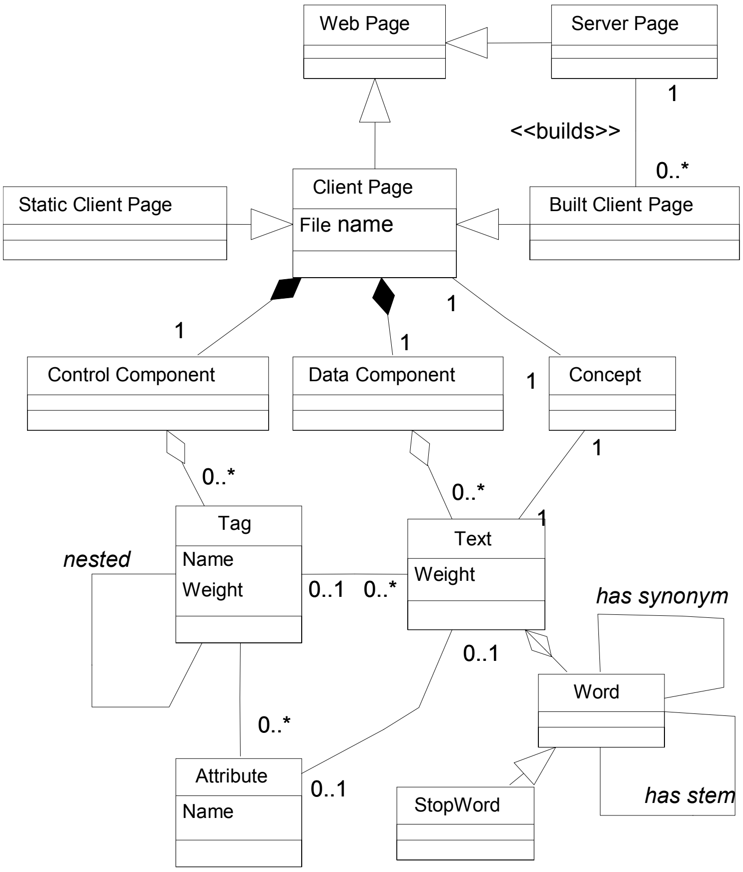


Figure 9.1: The Web Application's reference conceptual model

A stepped process has been defined for accomplishing the task of assigning a concept to each Web page. The input data of the process are the client pages, and for each client page the following steps are executed:

1) separation of the control component from the data component of the page;
2) normalisation of the text making up the data component; each piece of normalised text is a candidate concept;
3) computation of a relevance weight for each candidate concept;
4) clustering of similar candidate concepts and computation of a weight for each cluster of concepts;
5) selection, among the candidate concepts, of the concept to assign to the page.

9.3.1 Separating the Control Component from the Data Component

In the first step of the process, the code of each client page is parsed with the aim of separating the HTML tags making up the control component from the pieces of text making up the data component .

Let's consider the following example:

```
<html>
<b>Hello World
    <h6>I am here</h6>
 </b>
</html>
```

The corresponding sequence of tags is the following:

```
<html> <b> <h6> </h6> </b> </html>
```

will form the control component, while the two sentences: '`Hello World`' and '`I am here`' will form the data component.

Also the text making up the values of some tag attributes have to be considered, such as the value of the `alt` attribute of the `IMG` tag, because they could give a significant contribute to the definition of the page concept.

The control and data component of each page and the relationships among them are stored into a repository that stores all the information about a Web Application represented by the Web Application's conceptual model in Figure 9.1.

9.3.2 Text Normalisation

In this step, each piece of text belonging to a data components is analysed in order to:

- eliminate stopwords;
- substitute each word with its stem, when necessary;
- resolve synonyms, by substituting each word having one or more synonyms with a reference synonym.

At the end of this step, each normalised text will correspond to a candidate concept.

The normalisation operation is needed in order to avoid that two or more text pieces including synonym words, or using singular/plural or masculine/feminine forms are associated with different concepts, while they actually represent the same concept.

Just as an example, let us consider the following two sentences:

```
'At the party the actresses had on dark dresses'
'At the reception all the actors had on black suits'.
```

Both sentences are normalised as follows:

```
'party actor have dark suit'
```

since 'actor' is the stem for 'actors' and 'actresses', 'have' is the stem for 'had', 'suit' is the stem for 'suits' and 'dresses', and 'party', 'dark' and 'suit' are the reference synonyms for 'reception', 'black' and 'dress' respectively; moreover all the other words ('at', 'the', 'all', 'on') in the two sentences are stopwords and then they will not be included in the normalised form. In this way, the normalised forms of the two sentences are actually associated with the same concept.

The normalisation process is also needed to allow the successive step of concepts clustering to be carried out.

9.3.3 Computation of the Concept Weights

In this step, a weight is computed for each candidate concept (i.e. each normalised text) identified in a page, on the basis of both the editing format used to display the piece of text associated to that candidate concept, and the position of the text in the page.

Different editing formats can be used to give more or less emphasis to text. Usually, editing formats such as bold, italics, underlined characters, as well as a large size characters, are used to give greater emphasis to text, while small size characters are used to reduce the emphasis of the text.

In a client page, the text editing format is determined by the tags, and tag attributes, that enclose the text itself. HTML tags that provide either no emphasis, or greater emphasis, or less emphasis to text are distinguished by normal editing formatting.

For example, the tags <B> and </B> are used to display the enclosed text in bold format, while the tags <SMALL> and </SMALL> are used to display the enclosed text with a reduced size of the character. Therefore, it is possible to classify the HTML tags according to the emphasis they give to the text.

Table 9.1 shows an excerpt of the HTML tags classification that has been used in the proposed method. The tags have been classified as:

- Neutral, i.e. no emphasis is given to the text,
- Medium, just a little emphasis is given to the text;
- High, a significant emphasis is given to the text;
- Very High, the maximum emphasis is given to the text;
- Low, to reduce the emphasis of the text;
- Very Low, to give the minimum of emphasis to the text.

A weight has been associated to each class of tags; the values of the weights shown in Table 1 are the ones that produced the best results in some experiments that have been carried out.

Similarly, a classification and a set of weights was defined for the tag attributes.

A particular consideration has to be done for the tag A (i.e. the tag specifying an anchor for a hyperlink to another page): the effect of this tag consists of highlighting the hyperlink (usually by underlining it), but the text associated with this tag, typically, provide a description (i.e. a concept) of the target page of the hyperlink, rather than a description of the page containing it.

Since more tags may be applied to a text, their total effect on the text is given by their cumulative actions; as an example, the following statement:

```
<b> <i> Hello World </i> </b>
```

will display the sentence '`Hello World`' both in bold and italic format.

To take into account the cumulative effects of tags, the Total Weight (TW) is considered. It is computed as the product of the single weights of each tag involved in the cumulative set of Tags applied to a piece of text:

$$TW\ (TS(Ci)) = \Pi_{T \in TS(Ci)}\ W\ (T)$$

where TS is the set of tags cumulatively applied to the text forming the candidate concept Ci, T is the generic tag in TS and W(T) is the weight associated to the single tag T.

Table 9.1: Classification and weight of the HTML Tags

Tag Name	Class	Weight
S, DEL, A, SMALL, STRIKE, ……….	Very Low	0,25
KBD, H6, CITE, CODE, … ………	Low	0,5
COL, COLGROUP, COMMENT, DD, H5, DEN, DIR, EMBED, EM, THEAD, BUTTON, TFOOT, FIELDSET, FN, FONT, FRAME, FRAMESET, TEXTAREA, DIV, ADDRESS, TT, U, UL, VAR, WBR, XMP, SERVER, SHADOW, SIDEBAR, BODY, ACRONYM, BR, HTML, ………	Neutral	1
B, OL, DL, STRONG, MENU, MARQUEE, H3, BLINK, BIG, Q PRE, TH, TR, I, CENTER, CAPTION, FORM, ………	Medium	1,5
TITLE, H2, ………	High	2
H1	Very High	3

A similar computation is made for tag attributes, when they act in a cumulative way, and a Total Attribute Weight (TAW) is defined as the product of the weights associated to the single attributes.

As an example, let us consider the HTML web page in Figure 9.2, where the boldface letters between the HTML tags would represent any text to display to a user:

```
<title>a b c d e f</title>
<html>
<b>a b c d
     <h6>x y z</h6>
</b>
<div>w x y</div>
<h1>a b c d g h i j k l
     <code>c d e f g</code>
</h1>
</html>
```

Figure 2: An example of a HTML Page

For the sake of brevity, the text represented by the boldface letters is considered as already normalised; in this case the following six candidate Concepts, named C_1…$C_{6,}$ have been identified:

```
C1={a,b,c,d,e,f}

C2={a,b,c,d}

C3={x,y,z}

C4={w,x,y}

C5={a,b,c,d,g,h,i,j,k,l}

C6={c,d,e,f,g}
```

While the control component of the page is formed by the tag sequence:

```
<title> </title> <html> <b> <h6> </h6> </b> <div> </div> <h1>
<code> </code> </h1> </html>
```

According to the values in Table 9.1 the weights associated to each tag in the control component are:

Tag	H1	TITLE	B	DIV, HTML	H6, CODE
W(T)	3	2	1.5	1	0.5

And the cumulative weights associated with each one of the six candidate concepts C1, …, C6 are:

	C1	C2	C3	C4	C5	C6
TW	2	1.5	0.75	1	3	1.5

The Position of a candidate concept in the page displayed to the user is the other feature to consider to evaluate the emphasis given to the displayed text. Indeed, the text displayed at the top of a page is read as soon as the page is displayed in the browser window, without scrolling it, and that gives more emphasis to that piece of text. Vice-versa, the text at the bottom of the page has, usually, a less emphasis and may require the scrolling of the page in order to allow the user to read it.

The Position of a candidate concept can be evaluated by considering a text string formed by all the normalised text, making up the candidate concepts. These concepts are copied in the string with the same sequence by which they are encountered in the source code of the page (i.e. with the same sequence by which they are displayed). Let us say CS this string, LS the length of the string CS, and Pos(Ci) the position in CS of the first character of the first word of the candidate concept C_i. Position Weight (PW) must be computed to associate to each candidate concept C_i, as follows:

$$PW\ (Ci) = 1 - [\ Pos(Ci) / LS\ (CS)\ ]$$

As an example, the position weights for the six candidate concepts of the page in Figure 9.2 are:

	C1	C2	C3	C4	C5	C6
PW(Ci)	1	0.80	0.68	0.59	0.5	0.16

Finally, the Concept Weight (CW) to be associated with each candidate concept Ci of a page is computed as:

$$CW(Ci) = TW(Ci) * TAW(Ci) * PW(Ci)$$

Of course, if no attribute value is considered it is TAW(Ci)=1.

The Concept Weight CW(Ci) takes into account both the editing format used when the candidate concept Ci is displayed, and the position in the page where it is displayed.

As an example, the following table reports the CW(Ci) values computed for the six candidate concepts of the page in Figure 9.2.

	C1	C2	C3	C4	C5	C6
CW(Ci)	2	1.21	0.51	0.59	1.5	0.24

9.3.4 Concept Clustering

Some particular cases are those where different candidate concepts contain the same words, or two or more candidate concepts share all their words, or when all the words of a candidate concept are included in another concept.

These cases may indicate those concepts may share any common sub-concept, or that some concepts may be equivalent, or that a concept is composed by one, or more, other ones.

However, if a same group of words is included in more than one candidate concept, this group is likely to be more representative of the actual concept to assign to the page.

This communality of words among different candidate concepts may be exploited to identify and cluster common concepts, in order to reduce the number of candidate concepts of a page and to built more significant concepts.

In this case, the proposed clustering criterion is based on a metric of similarity SD between couples of candidate concepts, that is defined as follows:

$$SD(Ci, Cj) = |\ (FC_i \cap FC_j)\ | \ /\ Max(\ |FC_i|, |FC_j|\ )$$

where:

- C_i and C_j are two candidate concepts;
- $FC_i = \{W_1, \ldots W_n\}$ and $FC_j = \{W_1, \ldots, W_m\}$ are the sets of the words forming the candidate concepts C_i and C_j respectively;
- $|S|$ is the cardinality of the set S;

Thus, if two candidate concepts share no words, the Similarity Degree (SD) is zero, while if they share all their words SD is 1.

Two candidate concepts Ci and Cj are considered to be similar if the value of SD(Ci, Ci) is equal or greater than a predefined threshold TH, so they can be clustered together according to the following rule:

$$\text{Ci is similar Cj iff } SD(Ci, Ci) \geq TH$$

Thus, each cluster will group the candidate concepts satisfying the above condition; among all candidate concepts grouped into a cluster the Prevalent Candidate Concept (PCC) can be identified, that is the candidate concept having the maximum value of CW(Ci), being Ci a candidate concept belonging to the considered cluster. The PCC of each cluster is then considered as the concept describing the cluster itself.

Finally weight has been computed for each cluster, called ClW, defined by the sum of all the CW(Ci) of the candidate concepts in the cluster:

$$ClW\ (CLi) = \Sigma_{Cj \in CLi}\, CW(Ci)$$

Table 9.2 shows the values of SD(Ci, Cj) for the six candidate concepts of the example HTML page in Figure 9.2, while Table 9.3 reports the results of the clustering of those candidate concepts by using TH=0,65, and Figure 9.3 shows a graphical representation of the resulting clustering where the PCC of each cluster is shown by a boldface circle.

Table 9.2: The SD(Ci, Cj) values of the candidate concepts of the example HTML page in Figure 9.2

SD(Ci,Cj)	C1	C2	C3	C4	C5	C6
C1	1					
C2	0,67	1				
C3	0	0	1			
C4	0	0	0,67	1		
C5	0,4	0,4	0	0	1	
C6	0,67	0,4	0	0	0,3	1

Table 9.3: The results of the clustering with TH=0,65

	Composition	PCC	Concept Weight
CL1	{C1,C2,C6}	C1	3.45
CL3	{C5}	C5	1.50
CL2	{C3,C4}	C3	1.10

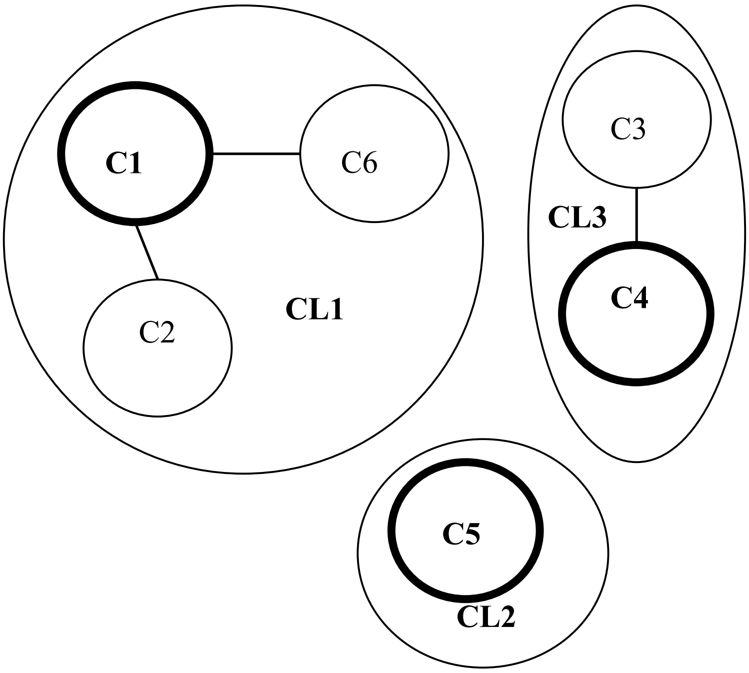


Figure 9.3: A graphical representation of the clustering results in Table 3

9.3.5 Selecting the Concepts

The identified clusters are sorted in a descending order with the values of ClW(Cli): the prevalent candidate concept of the cluster at the top of this list is proposed as the concept to assign to the page from which the candidate concept was extracted.

For the example page in Figure 9.1, C1 is proposed as the concept of the page.

9.4 The Web Concept Finder Tool

In order to validate the proposed method, a tool named Web Concept Finder (WCF) that supports the method, has been developed.

WCF tool is composed of a set of separate modules, written in Microsoft Visual C++ and Microsoft Visual Basic languages. The architecture of the tool is shown in Figure 9.4.

The description of the WCF tool modules follows.

- Control-Data Separator. This module implements a parser of HTML pages, and separates the data component from the control component of each parsed page. Parsing results are stored in the ConceptDB database.
- Word Finder. This module implements a pre-processing of the data component extracted by each page, for extracting the single words and deleting punctuation. Output of this module is a filtered version of the data component text, which is written in ConceptDB database.
- Stopword Eliminator. This module deletes stopwords from filtered Concepts.
- Stemmer. This module tries to transform words into their stems. Realization of the Stemmer module depends on the language used in the data component of the pages. Most of the algorithm solving stemming problem are base on the Porter algorithm [Por80], applicable to English language. This algorithm consists in a set of transformation rules by which some affixes and suffixes are deleted from words. This Porter algorithm cannot be used for languages such as Romance languages (French, Italian, Spanish, Portuguese, etc.), because these languages have more complex grammar rules, and many exceptions, than English language. More complex algorithms are needed to stem words in these languages, such as those developed for the Portuguese language [Ore01] and Spanish language [Hon00]). Because, the text of the analysed Web Application was written in Italian language, a stemmer for Italian language has been implemented. It exploits the grammar analysis and spell checking functionalities of Microsoft Word 2000. A set of 177 transformation rules has been considered. The most of these rules are applied to transform conjugate forms of verbs to infinitive forms. To reduce the effect of incorrect stemming due to grammar exceptions, a spell checking of each stem is performed before the substitution of the word with its stem.
- Synonym Tool. This module further reduces the size of the set of filtered text, by resolving synonyms of the words. All the synonyms of a word are looked for within the set of filtered text, and therefore they are substituted with the same word.
- Concept Weighter. This module calculates the Concept Weights, using the method described in Section 9.3.3.
- Cluster Finder. This module executes the concept clustering algorithm described in Section 9.3.4. Candidate Concept Clusters, Prevalent Candidate Concepts and Cluster Weights are identified and calculated. Results are reported in ConceptDB database.
- Page Concept Finder. This is the only module with a user interface. This module reports the ranking of the Candidate Concepts, sorted in descending order with the Cluster Weights.

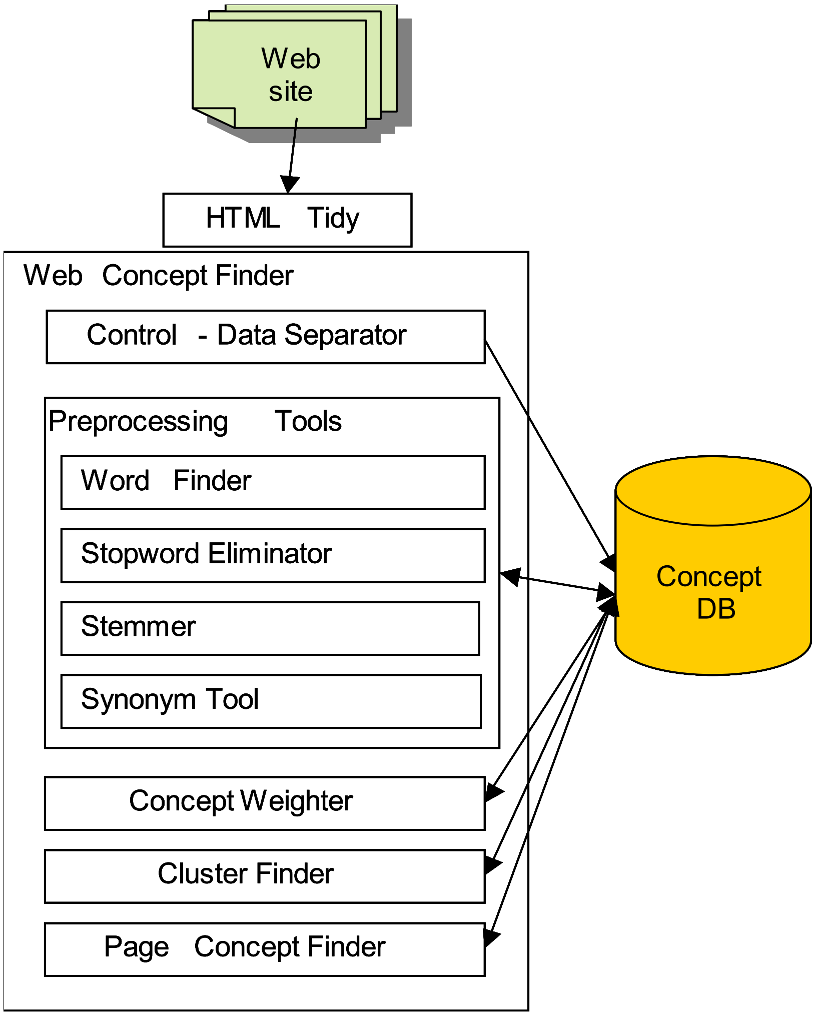


Figure 9.4: The architecture of the Web Concept Finder Tool

9.5. Experimental results

The proposed method has been experimented on a number of medium-sized Web Applications selected from real world. The aim of the experiment was to assess the effectiveness of the method to support the comprehension of the applications. In order to evaluate the effectiveness, the concepts assigned to the Web pages of the applications and proposed automatically by the method were compared with those ones obtained by a 'manual' concept assignment process carried out by software engineers who were unfamiliar with the applications.

A first experiment involving four primarily static web applications, i.e. applications whose client pages are mainly implemented by HTML and with no user-interactivity, was carried out. According to the classification proposed by Tilley and Huang (cfr. Section 2.2) they were Class 1 web applications. This type of applications has been selected since they usually have a large number of client pages with a lot of text inside describing the topics the web site deals with. The goal of this experiment was to verify that the concepts detected by the proposed method actually coincided with the main topic addressed by each analysed page.

The first considered web application (namely WA1) consisted of the web site of an Italian University, the second one (WA2) was a web site dealing with the Italian medieval history, the third one (WA3) was a web site reporting touristic information about an Italian island, and the fourth one (WA4) was the web site of an Italian research network.

Table 9.4 shows the total number of Client pages of each analysed application, together with the total number of HTML tags and Data Components extracted after the execution of the first phase of the process described in Section 3. The last row reports the average number of Data component for client page.

Table 9.4: Some data about the analysed Web Applications

	WA1	WA2	WA3	WA4
Client Page #	104	62	58	31
HTML Tag #	23783	8121	13660	13515
Data Component# (DC#)	3446	1304	2322	2992
Average DC# for page	33.14	21.03	40.04	96.52

The text making up each element of the Data Component was normalised producing the candidate concepts. A weight was computed for each candidate concept according to the formulas presented in section 9.3.3. Therefore, the Concept Clustering step was executed and similar candidate concepts were identified and grouped. Several different threshold values has been tried for concepts clustering and, for all the applications; the best results have been obtained with a value of Threshold of 0.8.

For each page, the concept clusters were sorted as described in section 9.3.4 by producing a sorted list. The Prevalent Candidate Concept of the cluster at the top of each list was taken as the Concept describing the page corresponding to that list.

The selected PCCs were compared with the ones produced 'by hand' by some software engineers. The result of this analysis was that all the selected PCCs contributed to provide the right concepts to the page but in most cases they did not provide the full page concept (according to the ones provide by the software engineers) but just partially described it. Indeed, for WA1 only about the 20% of the PCCs just provided the full page concept, while for WA2 about 43%, for WA3 31% and about 65% for WA4. However, the full page concept could be obtained by merging the selected PCC with the PCCs of the other clusters that immediately followed the first one in the cluster ordered list. In particular, just considering the first 5 clusters (i.e. the first 5 PCCs) in each list, for WA1 there are the 91% of successful cases, the 60% of successful cases for WA2, the 92% of successful cases for WA3, and 80% of successful cases for WA4. The percentage of successful cases was greater than 93% by considering the first 10 clusters of each ordered list.

One of the reasons because the selected PCC was not able to fully describe the page was that in many cases the PCC was derived from the main title of the page displayed to the user, while the complete concept of the page was formed both by that title and one or more subtitles (represented by a less weighted candidate concept due to the minor edit formatting evidence and the successive position in the page).

Thus, it is possible to conclude that to obtain better results not just the PCC of the first cluster in the list has to be considered, but also the ones that immediately follow it in the list. Indeed, in some other experiments that have been carried out, considering the first three to six clusters in the list resulted in an average percentage of successful cases greater than 70%.

These data confirmed the validity of the proposed method: in all the experiments, the selected first PCC rightly described, even if partially, the actual page concept, while the PCCs that are in the immediate successive position in the list are able to describe the page concept completely.

The method can be used to support the assignment of a meaning to software artefacts extracted applying the Reverse Engineering process previously described. As an example, it allows to reduce the human effort related to the association of a meaning to a cluster of pages.

Further details about this method can be found in [Dil04e].

9.6 Future works

Further work should be addressed in the context of Web Applications with large dynamic content, whose pages can be created on-the-fly, depending on the user interaction with the application.

The experiments highlighted some limitations of the method too, and possible solutions to these limitations had to be explored. As an example, the results produced by the clustering algorithm used in the final steps of the proposed method can be improved by considering also the relative position of two candidate concepts in the page and defining new criteria to group two successive concepts. Moreover, the combined usage of the proposed method together with 'traditional' Information Retrieval methods should be analysed in order to obtain more effective results.

Future work will have to consider also the analysis of style sheets defining the edit formatting of page content, as well as the analysis of other languages used for implementing Web Applications, such as XML.

Chapter 10: Recovering Interaction Design Patterns

In this chapter a method supporting the identification of Interaction Design Patterns in the source code of Client pages of a Web Application is presented. The method, the reference model and the supporting tool that have been developed are described in this chapter. The results of some explorative experiment are also reported.

10.1. Introduction

In the previous chapter, the problem of the concept assignment of client pages has been addressed. The proposed method recover concepts that describe the content of a client page, but not the functionality it realizes. In this chapter a method is proposed to identify if a client page contains a well-known Interaction Design Pattern. The presence of this pattern is a remarkable clue for the human expert to individuate the functionality realized by the client page.

The concept of design pattern has been introduced in software engineering to produce more flexible and reusable software systems. A design pattern provides a reusable solution to a problem, and in object oriented systems it consists of a collection of related classes and objects [Gam95]. However, the applicability of the design pattern concept is not limited to the object oriented context. Indeed, design patterns have been defined and catalogued in the context of Human Computer Interaction to design User Interfaces (UI) ([Hill], [Hcip], [Bor01], [Tid98]), where a UI pattern provides a common way for a user to interact with a software system.

Related to UI patterns are *Web Interaction Design Patterns* (WIDP) that provide a solution to the classical interaction problems of Web Applications users. As an example, typical interaction patterns in a Web Application include the ones allowing a user to search for an item on the Web, to register at a web site, to participate to a user forum, etc. Specific Web UI patterns have been proposed in the literature by [Duy02], [Gra03] and [Wel03]. In particular, van Welie in his Web site [Wel04] provides a catalogue of interaction design patterns for Web Applications.

In van Welie's catalogue, patterns are described in terms of the *Problem* addressed by the pattern, the *Context* where it should be used, the *Solution* to use for solving the Problem and an *Example* providing a screenshot and some additional text to explain the context of the particular solution. As an example, Table 10.1 shows the description of the Login pattern as it is reported in [Wel04].

Table 10.1: The 'Login' Web Interaction Design Pattern description

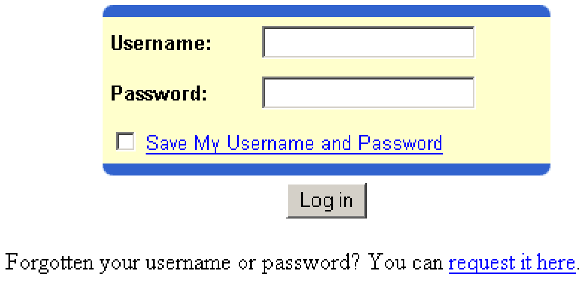
Problem The users need to identify themselves so that stored data about/of them can be used in the process they are in.

Context When users frequently return to a site that uses large amounts of data about or belonging to the users, it is convenient to have users enter that information once and use it again for future visits to the site. Usually the information that is stored is personal information and can include name, age, gender, shipping addresses, stock portfolio, bank account numbers and credit card numbers. In order to be able to access their data, users must complete their Registration first.

For many site types it can be convenient to store information of/about visitor. Often these are E-commerce Site, Community Site or Web-based Application such as electronic banking applications.

Solution When needed, ask the users to login using a combination of a username and a password

Example



Forgotten your username or password? You can request it here.

Although the necessary degree of abstraction that characterizes a pattern description, a pattern can be actually used if it is defined precisely, in order to allow a software engineer to find and apply it. As an example, an effective definition of an interaction pattern can be given in terms of the UI Model fragments it typically comprises.

Recovering the WIDPs implemented in existing Web Applications represents a viable solution to detect components to be reused for developing new applications, or for comprehending an existing application to be maintained, or evolved.

While several approaches for detecting recurrent design patterns from the code or the design of object oriented systems have been proposed in the literature ([Ase02], [Bal03], [Ton99]), the problem of reverse engineering a Web Application for detecting the WIDPs it implements has not yet been addressed.

In this chapter is proposed a method for identifying the WIDPs implicitly implemented in a Web Application that is based on the detection of the characteristic features (such as UI Model fragments, or lexical terms) they include. The method requires that fragments that are characteristic of a given pattern must be preliminary identified. To reach this aim, a metric that expresses the degree of characterization of a feature for a given pattern, with respect to a family of instantiations of the pattern, has been defined.

Therefore, an interaction design pattern is identified in a Web page by recognizing the most characteristic features of that pattern included in the page The method requires the reverse engineering of the Web Application’s code in order to identify the code components implementing the UI fragments.

10.2. Background

In the context of Web Design Interaction Patterns, although there may be many different ways of implementing a solution to the same problem (i.e., there may be a family of instances of a same pattern), some characteristic *UI fragments* in the different implementations can be found. As an example, characteristic fragments of the ‘Login’ pattern shown in Table 1 will include a form with two input fields, a submit button and ‘login’ and ‘password’ texts labeling the input fields.

Generally, an interaction pattern in a Web page is characterized both by the UI graphical items it includes (such as tables, forms, grids, input fields, etc.), both by lexical items i.e., text that is shown to the user, and that describes the interaction function (such as login, poll, site map etc.). Hereafter a *feature* may be defined as the occurrence of a UI graphical items or of a lexical items included in a page of a Web Application. The occurrence of each feature in an existing Web Application can be deduced automatically by analysing the code of the application.

These features can be used as clues for detecting interaction patterns in the code of existing Web Applications. The features that are more effective to describe a given pattern are those that are associated most *frequently* with the different instances of that pattern, and that are, at the same time, *specific* of the pattern. A specific feature is the one that can be often found in a given family of patterns, but rarely exists in other patterns.

The definition of characteristic features that can be associated with Web Interaction Design Patterns and the problem of assessing the effectiveness of a feature to recognize a given pattern are addressed in the following sub-section.

10.2.1 Characterization of Patterns’ features

Given a pattern P, in order to identify its most characteristic features, a set of client Web pages has been selected, each one containing just an instantiation of a pattern P. This set of Web pages is called *Training Set (P),* and the set of features they exhibit has been collected. Considered features include the UI model fragments reported in Figure 10.1 as a class diagram. The class diagram represents the UI fragments that are usually included in a Web page and possible relationships between them, which are considered as possible features of a WIDP. The model below is a specialization of the conceptual model described in section 3.3.

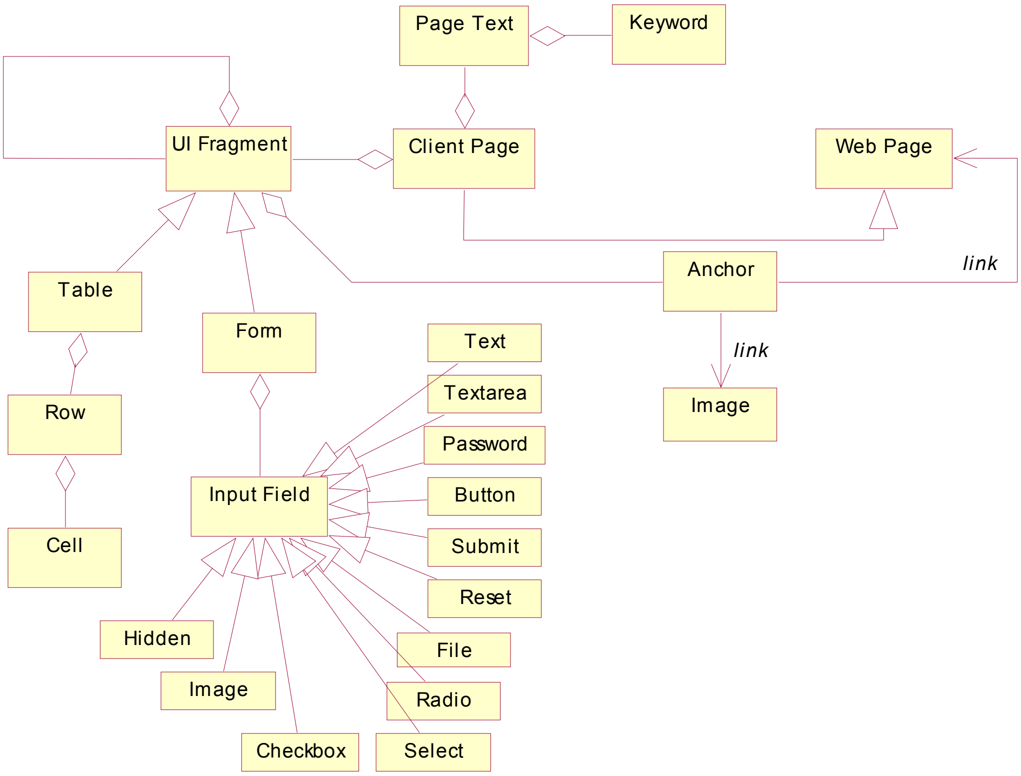


Figure 10.1: User Interface Fragments Model

For each pattern P and feature F, let's consider the frequency *Freq(P, F)* of the feature F in the *Training Set (P)*:

$$\mathit{Freq(P,F)} = \sum_{wp \in \mathrm{TrainingSet(P)}} \mathrm{Occ(wp,F)} / \mathrm{Card(TrainingSet(P))} \qquad (1)$$

Where:

- Occ(wp,F)=1 if the feature F is included in the wp Web page, elsewhere Occ(wp,F)=0
- Card(TrainingSet(P)) is the cardinality of the Training Set(P).

Of course, $0 \leq Freq(P,F) \leq 1$, where Freq(P,F)=0 means that F is never used for implementing the pattern P, while Freq(P,F)=1 means that F is used in every instantiation of the pattern P.

Besides the frequency of a feature, the specificity *Spec* of a feature for a given pattern P must be considered. It is an indicator of how much a feature F is specific of a pattern P. The specificity of a feature F for P can be evaluated with respect to a set of considered Patterns (e.g., all the patterns in Welie's catalog, or a subset of them) and by assessing frequency of the feature F in each pattern.

The set of considered patterns is called *PatternSet.*

The average value of the frequencies of the Feature F in each pattern of the *PatternSet* is called *Av(F)* and it is expressed as it follows:

$$\mathrm{Av(F)}=\sum_{\mathrm{p}\in \mathit{PatternSet}}\mathrm{Freq(p,\ F)/\ Card}(\mathit{PatternSet}) \qquad (2)$$

Therefore *Spec* can be evaluated by the following formula:

$$Spec(P,F) = \begin{cases} Freq(P,F) - Av(F) & \mathit{if}\ \mathrm{Freq(P,F) > Av(F)} \\ \\ 0 & Elsewhere \end{cases} \qquad (3)$$

Of course, *Spec(P,F)* varies in the range [0,1[.

Spec(P,F)= 0 when Freq(P,F) ≤Av(F), that is, a feature F is less frequent in P than in the complete set of considered patterns.

More frequently the feature F is encountered in the P pattern's instantiations, and less frequently in other patterns' instantiations, the greater is the specificity of the feature F for the pattern P.

Finally, in order to evaluate how much a feature is characteristic of a given pattern, the *Characterization Weight CW*(P,F) of a feature F for a pattern P is defined as it follows:

$$\mathrm{CW(P,F) = Freq\ (P,F) * Spec\ (P,F)} \qquad (4)$$

CW(P,F) varies in the range [0,1[and it represents a combined index that takes into account both Frequency and Specificity of a feature F for a pattern P. The most characteristic features for a given pattern are those showing the greater values of CW.

10.3. The approach

The approach proposed for searching patterns in existing Web Applications includes three main phases: a training phase, a candidature phase, and a validation phase. The training phase is devoted to the computation of the features' Characterization Weights on a training set of instantiations of the patterns to be searched for. The second phase exploits the computed Characterization Weights to candidate possible patterns in a set of Web pages to be analysed. The third phase is devoted to the validation of the candidate patterns.

Figure 10.2 illustrates the phases of the method and the main input and output of each phase. Additional details about each phase are presented in the following.

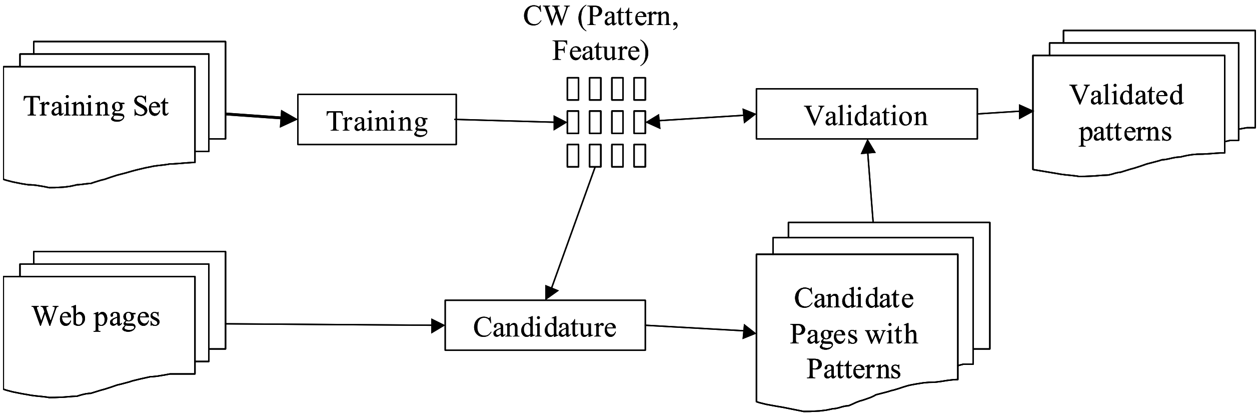


Figure 10.2: Web Interaction Design Pattern Recovering Process

In the first phase, to obtain a training set, a number of Web pages with different characteristics (such as language, domain of the application, purposes, producer, etc.) have to be taken into account. Considered features in this sample will describe the occurrence of the typical UI fragments included in a Web Application, and that are reported in the model shown in Figure 10.1. At the end of this phase, Characterisation Weights CW(P,F) are obtained.

The second phase includes two main activities:

1) *Likelihood* evaluation for each Web page and pattern;
2) Identification of candidate patterns.

To compute the *likelihood* that a pattern P is included in a Web page WP*,* let's consider the occurrence in the WP of the most characteristic features for that pattern. A *likelihood* index *L(WP,P)* is therefore provided by the following ratio:

$$\mathrm{L}\,(WP,P)=\frac{\sum_{f\in FeatureSet(P)} CW(P,f)*Occc(WP,f)}{\sum_{f\in FeatureSet(P)} CW(P,f)} \qquad (5)$$

where:

CW(P,f) is the Characterisation Weight of a feature f for a pattern P;

FeatureSet(P) is the set of features *f* having CW(P,f)>0;

Occ(WP,f) =1 if the feature *f* is included in the WP Web page, elsewhere Occ(WP,f) =0.

In formula (5), the denominator has been introduced in order to make comparable the different CW values, as the patterns vary. The values of L(WP,P) vary in the range [0, 1]. L(WP,P) is 1 when all the features f having CW(P,f)>0 are contained in the WP page. Vice-versa, L(WP,P) is 0 when no feature f having CW(P,f)>0 is contained in the WP page.

In the second step, the identification of candidate patterns in pages is obtained by comparing the L(WP,P) with respect to a threshold, and assuming that a page WP includes a pattern P if L(WP,P) is greater than the selected threshold.

The choice of the threshold of course impacts on the set of candidates: greater the threshold, smaller the set of candidates, while lower the threshold, wider the set of candidates. The best threshold values encountered in a validation experiment are discussed in Section 10.5.

The third phase requires that the candidate patterns detected in pages be submitted to a validation phase aiming to assess that each page actually includes that pattern. The validation phase is performed manually, by a software engineer with experience in developing Web Interaction Design Patterns.

After the validation, validated patterns can be added to the training set samples, allowing the CWs for the new training set to be recalculated.

10.4 The supporting environment

The identification method described in the previous section is supported by the *Web Interaction Pattern Recovery* integrated environment. This environment is composed of a number of tools realized using Microsoft Visual C++ and Microsoft Visual Basic.

The environment supports the execution of the Training, Identification, Validation and Tuning activities required by the pattern identification method and includes the following modules:

Reverse Engineering Tool. This tool extracts information from a web client page about any UI feature included in the page. The extracted information is stored in the Web Application Information Repository whose model is coherent with the information model described in section 10.2.1. This tool is an extension of the WARE tool described in Chapter 5.

CW Evaluator. This module has the responsibility to compute the Characterization Weight values according to the formula (4). The computed values are stored in the Pattern Repository.

Likelihood Evaluator. This module computes the Likelihood values for each pattern and for each page, according to formula (5), by querying the Web Application Information Repository and the Pattern Repository.

Pattern Identifier. This module selects, for each page, the candidate interactions patterns identified in that page by comparing the *L(WP,P)* values with a prefixed threshold.

Pattern Validator. The responsibility of this module is to present the candidate patterns to a human expert that will validate them. Validated patterns may be used to extend the set of pattern samples and to tune the CW values.

CW Tuner. This module recalculates CW values taking into account further pattern samples and the validated patterns.

Figure 10.3 shows the architecture of the system (the Reverse Engineering Tool is reported twice just for graphical reasons).

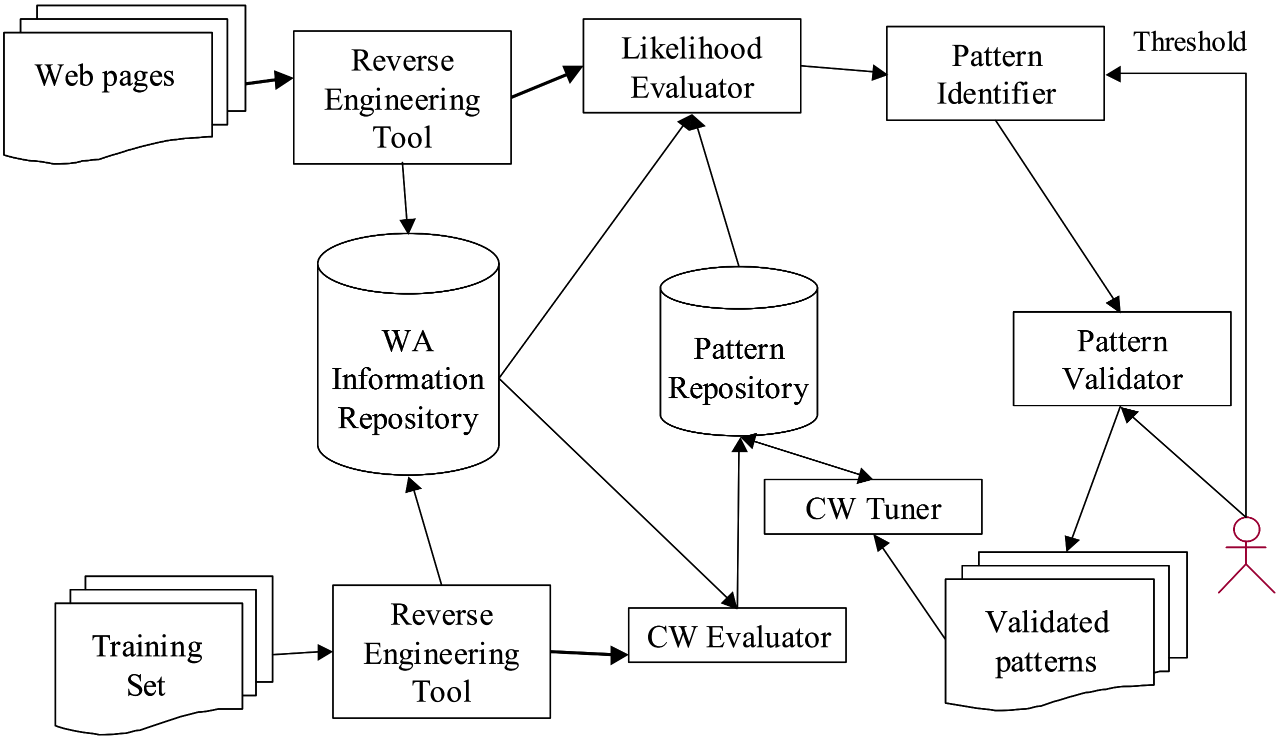


Figure 10.3: Architecture of Web Interaction Design Pattern Recovery System

10.5. An explorative experiment

To validate the proposed method, an experiment aiming at assessing the effectiveness of the method in the detection of patterns in Web pages of existing Web Applications has been carried out. In the experiment, the following six Interaction Design Patterns selected from the catalogue reported in [Wel04] have been considered:

- *Guestbook*, providing a view of the list of messages written by the visitors of a website;
- Login, consisting in an authentication module for inserting personal identification information needed to access to private services;
- Poll, consisting in a module to insert a vote for a poll;
- Registration, consisting in a module to insert personal data needed to registrate to a service;
- Search, consisting in a module to insert keywords for a search on a search engine;
- Sitemap, consisting in a view of the map of the pages of a website.

The main steps of the experiment were the following:

- Feature Definition, where a set of features that are characteristic of WIDPs are defined.
- Training Phase, where the training set is selected and CW values are calculated;

- Preliminary Identification Test, where the candidature process has been executed for the training set, obtaining a first evaluation of the effectiveness of the identification process;
- Identification Test, where the candidature process has been executed for a more significantly test set, obtaining a more reliable evaluation of the effectiveness of the process.

10.5.1 Feature Definition

This preliminary phase has been executed to define the set of features to be taken into account during the pattern identification process.

Selected features are those expressing the occurrence of a UI fragment (from the categories of fragments shown in Figure 10.1) or the occurrence of a combination of these fragments. As an example, the features that have been considered are related to the characteristics of a *form* - such as the occurrence of one or more input fields in a form- or they are related to *tables*, that take into account the occurrence of anchors, forms, input fields in cells/rows of tables, or they are *lexical features,* taking into account the occurrence of some common textual expressions labeling *form* and *table* features in web pages.

For instance, the experiment has shown that the occurrence of 2 or more *select* buttons in a form is a good feature to identify the Poll pattern, while the occurrence of 2 or more anchors in a row is a good feature to identify the Sitemap pattern (cfr. Table 3). Moreover, '*login*' and '*password*' words have been found to be good features to identify a login pattern.

The proposed method considers synonyms and translations of each textual expression as equivalent features. As an example, '*username*' is considered as a synonym of '*login*', while '*nome utente*' is considered as an Italian translation of '*username*'. If one of these features is found in the analysed Web pages, the occurrence of the equivalent feature is also added.

The set of lexical features considered in the experiment comprehends 36 textual expressions that are recurrent in the considered patterns. If more patterns are considered, more textual expressions will have to be defined.

10.5.2 Training Phase

The Training phase has been devoted to the computation of the features' Characterization Weights on a training set of instantiations of the patterns to be searched for.

The Training set was composed of 108 client pages, each of which implementing a single pattern. Table 10.2 reports the number of samples selected for each Pattern.

Table 10.2: Training Sets composition

Pattern	Samples#
Guestbook	15
Login	25
Poll	13
Registration	14
Search	20
Sitemap	21

In order to have a meaningful set of samples, they have been extracted from different websites. These websites were different as to the typology (portals, personal sites, e-commerce sites, educational sites and so on), language (English and Italian), and graphical layout.

The Characterization Weights have been calculated for each Pattern and for each Feature of the training set. Table 10.3 reports the list of features with the best values of Characterization Weight for each Pattern.

10.5.3 Preliminary Identification Test

To test the effectiveness of the method a preliminary identification test has been conducted. The input of this test is represented by the training set. This test was carried out to validate the characterization features selected by the previous step.

Likelihood values have been calculated with respect to the Characterization Weights values obtained by the training phase. The computed Likelihood values have been compared with a threshold to identify the pattern included in the pages.

To assess the correctness of the results of the candidature phase, Recall and Precision metrics have been adopted. Of course, the set of Patterns included in the training set was known, thanks to an analysis performed by an expert software engineer. Recall and Precision have been defined in the following way:

Recall: Number of correct candidate couples (web page, pattern) / Number of couples (web page, pattern) to identify

Precision: Number of correct candidate couples (web page, pattern) / Number of candidate couples (web page, pattern)

Table 10.3: Features with CW > 0.2

Best Guestbook Features	CW
Word 'guestbook'	0,8205128
Word 'comment'	0,5118095
Word 'page'	0,3888718
Word 'at least'	0,3510375
Word 'none'	0,3489927
Word 'next'	0,2917176
Word 'email'	0,2855873
1 anchor to an email in a cell	0,2334815
1 anchor to an email in a row	0,2334815
Word 'message'	0,2080847
Best Login Features	**CW**
1 password button in a table	0,7565714
1 radio button in a form	0,7565714
1 password button in a cell	0,7108571
1 password button in a row	0,7108571
Word 'password'	0,7103174
Word 'login'	0,6575238
1 input text field in a cell	0,503619
1 input text field in a row	0,503619
1 input text field in a table	0,4972698
1 image field in a form	0,422
1 submit button in a row	0,3206838
1 submit button in a table	0,3206838
1 submit button in a cell	0,3104274
1 checkbox field in a form	0,231387
Word 'username'	0,2242125
Best Poll Features	**CW**
Word 'poll'	0,5644125
2 or more select buttons in a form	0,4741561
Word 'vote'	0,4399831
Word 'results'	0,4216681
2 or more radio button in a table	0,3189687
1 radio button in a cell	0,3031277
Best Registration Features	**CW**
Word 'registration'	0,7190476
2 or more images in a form	0,6937755
2 or more input text fields in a table	0,5951021
Word 'last name'	0,5912925
Word 'email'	0,5569841
Word 'address'	0,4179251
Word 'none'	0,3489927
1 input text field in a cell	0,2752494
1 input text field in a row	0,2752494
2 or more input text fields in a cell	0,2682993
2 or more input text fields in a row	0,2682993
Word 'password'	0,2270408
Word 'all'	0,2053288
Best Search Features	**CW**
Word 'search'	0,5579731
1 text input field in a table	0,3880635
1 text input field in a row	0,2916826
1 text input field in a cell	0,2916826
Best Sitemap Features	**CW**
Word 'map'	0,775873
2 or more anchors in a row	0,3103768
Word 'search'	0,2974975
2 or more anchors in a table	0,2941863
2 or more anchors in a cell	0,2801186
2 or more anchors to image in a table	0,2350997
2 or more anchors to image in a cell	0,2090476
2 or more anchors to image in a row	0,2011111

Different candidate sets of patterns have been obtained by varying the threshold value, and Precision and Recall has been measured for each of these threshold values. Table 10.4 reports Recall and Precision values for 9 different threshold values, ranging between 0.1 and 0.9. A satisfying trade-off was obtained with a threshold value of 0.6 (82% of Recall, 79% of Precision). These results confirmed the adequacy of the choice of the set of features for the detection of the selected set of patterns.

Table 10.4: Recall and Precision value in the Preliminary Identification Test

Threshold	0,9	0,8	0,7	0,6	0,5	0,4	0,3	0,2	0,1
Recall	22/108	46/108	69/108	89/108	93/108	101/108	106/108	107/108	108/108
%	20%	43%	64%	82%	86%	94%	98%	99%	100%
Precision	22/22	46/49	69/75	89/112	93/135	101/174	106/234	107/325	108/487
%	100%	94%	92%	79%	69%	58%	45%	33%	22%

10.5.4 Identification Test

After the Preliminary Identification Test, another experiment has been executed on a wider set including 208 Web pages. These pages have been selected in different web sites, belonging to different domains. The analysis performed by an expert software engineer stated which and how many patterns were included in the set of pages. The results of this analysis was that the Web pages included:

9 Search Patterns

10 Registration Patterns

12 Login Patterns

9 Poll Patterns

8 Sitemap Patterns

7 Guestbook Patterns

Some of the web pages in the set did not include any pattern, while some other ones included more than one pattern.

The evaluation of the Likelihood was executed for each page using the Characterization Weights values obtained in the training phase. Also in this experiment the Recall and Precision metrics were computed to assess the correctness of the results. The Table 10.5 reports Recall and Precision values for 9 different threshold values, between 0.1 and 0.9. Also in this case a satisfying trade-off was obtained with a threshold value of 0.6 (80% of Recall, 66% of Precision).

Table 5: Recall and Precision value in the Identification Test

Threshold	0,9	0,8	0,7	0,6	0,5	0,4	0,3	0,2	0,1
Recall	10/55	20/55	31/55	44/55	45/55	51/55	52/55	54/55	54/55
%	18%	36%	56%	80%	82%	93%	95%	98%	98%
Precision	10/11	20/24	31/39	44/67	45/112	51/171	52/229	54/347	54/590
%	91%	83%	79%	66%	40%	30%	23%	16%	9%

The results obtained by these two experiments showed us the validity of the proposed method, as it allows a good identification of WIDP in web pages. However, further work is needed to improve the quality of the results. Further details about the method described in this chapter can be found in [Dil05].

10.6 Future works

Of course, a first improvement considers the definition of the characterising features for other patterns and, as a consequence, the updating and refining of the *Characterisation Weights* CW(P,F). A wider training set would improve the Recall and Precision degree, too.

Further improvements are obtained by considering not just the presence of single UI fragments in a page, but considering the co-presence of an adequate group of them in a page, as the implementation of a specific pattern is generally characterised by groups of features.

To reach this aim, a further analysis of a wider number of web pages is required: this goal will be addressed as future work.

Chapter 11: Identifying Cloned Components

In this chapter a method based on clone analysis techniques with the aim to identify Web Application components with identical or similar structure is presented. Four techniques to measure the degree of similarity among client pages are defined. The results of an explorative experiment are also reported. Finally, a discussion about the combined used of the information retrieved by the reverse engineering techniques based on Concept Assignment, Interaction Design Pattern Identification and Clone Analysis is reported.

11.1 Software clones and clone analysis.

Addressing the problem of concept assignment, the information about the presence of duplicated or similar components in the Web Application can reduce the effort of this expensive task in a remarkable way. In fact, the semantics associated to duplicated components may be very similar. Anyway, clone analysis can have other important applications, such as in the reengineering of Web Applications or during dynamic analysis. In this chapter clone analysis techniques identifying duplicated or similar components or portions of code in a Web Application is described.

Duplicated or similar portions of code in software artifacts are usually called clones, and clone analysis is the research area that investigates methods and techniques for automatically detecting them.

The research interest in this area was born in the '80s ([Ber84], [Gri81], [Hor90], [Jan88]) and focused on the definition of methods and techniques for identifying replicated code portions in procedural software systems.

The methods and techniques for clone analysis described in the literature focus either on the identification of clones that consist of exactly matching code portions ([Bak93], [Bak95], [Bak95b]), or on the identification of clones that consist of code portions that coincide, provided that the names of the involved variables and constants are systematically substituted (p-match or parameterised match).

The approach to clone detection proposed in [Bal99] and [Bal00] exploits the Dynamic Pattern Matching algorithm [Kon95], [Kon96], that computes the Levenshtein distance [Lev66] between fragments of code. The value of the Levenshtein distance is a measure of the similarity between two vectors. It is defined as the minimum number of insertions, deletions and replacement of items necessary to make two vectors equal. Levenshtein distance is also called edit distance if the operations are weighted in a equivalent way. A classical alternative metric to measure the distance between two vectors is the Euclidean distance. To compute Euclidean distance, the distance between two items is equal to one if the items are different, else its value is equal to zero.

Baxter [Bax98] introduced the concept of near miss clone, which is a fragment of code that partially coincides with another one. Further approaches, such as the ones proposed in ([Kon97], [Lag97], [May96], [Pat99]), exploit software metrics concerning the code control-flow or data-flow.

11.2 Clones in Web Applications.

In a Web page a control component and a data component can be distinguished (cfr. Chapter 8). Focusing on software clones characterized by the same control component, two categories of clones, with different degrees of granularity, can be taken into account. At a coarse grained level, any duplicated or replicated static page of a Web Application can be considered as a software clone. In particular, two types of replicated pages can be distinguished: static client pages having the same HTML control component, i.e., pages composed by the same set and sequence of HTML tags, and static server pages coded using a script language, such as the ASP, and including the same set of ASP objects.

At a finer degree of granularity, inner components of a page that are replicated throughout the Web Application can be also considered as clones. Examples of these components include, on the client side of the application, script blocks, script functions, HTML forms, and HTML tables that are replicated in the pages of the application. On the server side, script blocks, script functions, and subroutines can be considered.

Table 11.1 reports the classification of Web Application software clones that are considered in this chapter. The classification distinguishes the clones depending on their degree of granularity.

Table 11.1: A classification of software Clones.

Degree of granularity of a clone	**Type of clone**
Web Page	Client Page
	Server Page
Client Page inner components	Script Block
	Script Function
	HTML Form
	HTML table
Server Page inner components	Script Block
	Script Function

11.3 Identifying clones

In this section, a clone analysis based process for identifying cloned components in a Web Application is presented. The process focuses on software clones provided by artifacts showing the same control component, and includes the following phases:

- Separation of the control component from the data component within Web pages.
- Clone Detection.
- Clone Validation.

11.3.1 Separation of the control component from the data component.

In the first phase of the process, Web pages are pre-processed in order to separate their control component from the data component, and submit the control component to the next phase of the process. As an example, the pre-processing of client pages is executed with the support of language parsers that extract and store the sequence of HTML tags from HTML files, by separating them from the list of attributes and data associated with these tags and determining the information to be read/displayed from/to a user. This task is the same described in subsection 9.3.1.

11.3.2 Clone Detection.

In the second step of the process, clone analysis is used for detecting clones in Web Applications.

Most clone analysis techniques compare distinct software artifacts on the basis of relevant information extracted from them: this information may be either a sequence of symbols (from a given alphabet) representing, for instance, the artifact control structure, or a set of software metrics characterizing them. The extracted information is used to compute a similarity distance between artifacts, such as the Levenshtein or the Euclidean distance: zero-distance items provide a software clone, while quasi-zero-distance items provide near-miss- clones.

As to the identification of cloned Web pages, two classes of techniques have been used: techniques that are based on the match between sequences of symbols (using the Levenshtein distance), and techniques based on software metrics extracted from Web pages (using the Euclidean distance). A list of possible techniques (and an explanation of the acronyms) is reported in Table 11.2. STH, and CTH techniques are applicable to client pages, while SOA, and AMA ones are applicable to server pages.

In the proposed process, all the techniques listed in Table 11.2 can be used for detecting cloned Web pages, in order to combine their different output and obtain more precise final results. A complete description of STH, CTH, and SOA techniques can be found in [Dil01], while the AMA one is described synthetically in the following.

The AMA technique is a metric based technique that detects couples of cloned server pages using a set of metrics extracted from pages including ASP script blocks. Each page is characterized by an array of software metric values, and the Euclidean distance of the corresponding arrays will compare couples of pages. Experiments have shown that the AMA technique detects clones more effectively than the SOA technique [Fer02].

The set of considered metrics is shown in Table 11.3.

Table 11.2: Clone analysis Techniques.

Technique	Description
STH	Strings of HTML tags extracted from client pages are compared by the Levenshtein distance.
CTH	Counts of HTML tags extracted from client pages are compared by the Euclidean distance.
SOA	Strings of ASP built-in objects extracted from server pages are compared by the Levenshtein distance.
AMA	An array of software metrics extracted from ASP script blocks in server pages is used to compare server pages by the Euclidean distance.

Before computing the first and last metric in Table 11.3, the AMA technique requires that a preprocessing be made on the ASP pages. The preprocessing is executed for making the comparison between script blocks independent on comment lines, writing style, and choice of identifier names in the analysed code.

Table 3: Software metrics used by the AMA technique.

Software Metrics
Total Number of characters per page (NTC)
Total Number of lines of code per page (NTRC).
Total Number of ASP code blocks per page (NTBC).
Total Number of declarative statements per page (NTSD)
Total Number of conditional statements per page (NTSC)
Total Number of cyclic statements per page (NTSI)
Total Number of Functions per page (NTF).
Total Number of Subroutines per page (NTS).
Total Number of built-in ASP objects per page (NTOP).
Levenshtein distance of ASP code blocks (LDBC)

In particular, in the ‘Separation of the control component from the data component’ phase, each line of ASP block code is transformed into an intermediate format by removing blank characters and comments from the line, and substituting each data element (such as constants, or identifiers) by a dummy one. Moreover, consecutive blocks of ASP code are merged in a single comprehensive block.

As an example, the consecutive ASP blocks of code shown in the following:

```
<% If HourPart>12 then %>
<% 'It is Afternoon %>
<% Response.write "Good Afternoon!” %>
<% Else %>
<% 'It is Morning %>
<% Response.write "Good Morning!" %>
<% Endif %>
```

is transformed as follows, by removing blank characters, substituting identifiers with DUMMY names, and merging the different blocks into a single block:

```
<%IfDUMMY>DUMMYthen
Response.writeDUMMY
Else
Response.writeDUMMY
Endif%>
```

11.3.3 Techniques for detecting Inner Page Clones

On the client side of the application, candidate reusable components are provided by script blocks, script functions (implemented in Javascript, VBScript or Jscript languages), HTML forms, and HTML tables that are replicated in the pages of the application.

In order to localize cloned script blocks and script functions (potentially implementing reusable functions), the STH technique based on textual string comparisons is used. As to the identification of cloned HTML input Forms (representing reusable components implementing the function of data input and the request for their elaboration to a server page), the same STH technique based on the Levenshtein distance, comparing strings of HTML tags associated with each form, is used. Zero-distance forms represent cloned forms, with the same HTML structure. A similar technique can be used for finding cloned HTML tables, which represent a reusable output data structure.

On the server side, cloned script blocks implemented in ASP, besides script functions, and subroutines written in VBScript or Jscript language are looked for, since they potentially implement reusable functional abstractions. The AMA technique is used for analysing single script blocks written in ASP, while script functions and subroutines are analysed by comparing textual strings extracted from the code, using the Levenshtein distance.

A tool, named CloneDetector, has been implemented which provides a unified framework for executing clone analysis using different techniques. Once the user has selected the techniques of clone detection to apply, the tool computes the Distance Matrix (whose items represent the distance between each pair of artifacts from the Web Application). Clusters of clones are, therefore, obtainable from this matrix.

11.3.4 Clone Validation.

The detected set of clones need to be submitted to a validation activity aiming at assessing if they can be actually considered reusable components or not.

In the Validation phase, the detected clones are analysed as regard their meaningfulness (does the considered clone implement a meaningful abstraction in the business domain, or in the solution domain?), and their completeness (does the clone include all the lines of code necessary for implementing a valid abstraction?). Meaningless clones are discarded, while incomplete clones may have to be merged in order to obtain an actually reusable component.

Validated clones are associated with a description of the reusable abstraction they implement. Business domain knowledge and experience in building Web Applications are required for carrying out the validation activity.

11.4. An experiment.

In order to assess feasibility and effectiveness of the proposed method, several experiments involving real Web Applications were carried out; the Clone Detector tool was used to support them.

This Section provides the results of the experiments involving four Web Applications (hereafter, WA1, WA2, WA3, and WA4), which were submitted to the process proposed in Section 11.3 for detecting their clones.

The first three applications presented client pages written in HTML and script languages, and server pages coded in ASP, while the fourth one just included ASP pages. WA1 implemented a ‘juridical laboratory’ with the aim of supporting the job of professional lawyers; WA2 was a Web

Application designed to support the activities of undergraduate courses offered by a Computer Science Department (this application has been also considered as case study in the previous chapters), WA3 was an e-commerce Web Application for selling music CDs; WA4 was a Web Application designed to manage an Internet discussion forum.

Table 11.4 reports the main experimental results obtained after the clone identification and validation phases of the process: in the Table, just the exact clones that passed the validation phase have been considered. For each category of considered artifact (e.g., Web page, script block, script function, etc…), the number of artifacts in the Web Applications, the number of detected cloned artifacts, and the number of clusters of cloned artifacts are reported. In particular, each cluster gathers together the set of the same cloned items.

In the experiments, the STH textual string comparison technique was used to identify clones among the client side components, while the AMA metric-based technique was used to identify cloned items among the server side components. The total number of false positives (that is, clones detected by the process which were not actually clones) is very low (less than 7%); and all the false positive clones regarded server side items. With reference to client pages, there are about 25% of cloned pages (33% for WA1, 8% for WA2, and 35% for WA3), while the average percentage of the cloned server pages was about 9%.

As to the inner page components, there is also a higher percentage of cloned items for client pages than for server pages. In particular, for the client functions the percentage of cloned items was about 48%, while the cloning of server functions was only 2%.

This datum may indicate that exact cloning of client side items is a practice adopted more frequently than cloning server side items; however, this datum may also depend on the fact that the AMA metric-based technique, that has been used for finding clones on the server side, exploited a Levenhstein based metric (cfr. LDBC metric in Table 11.3) that produced a relevant percentage of near missing clones, composed by ASP code blocks computing the same functions but just differing for the lexicographical order of some statements. Therefore, the precision of the AMA technique is very high (i.e., all detected clones were actually validated), but the recall parameter (e.g., the number of detected clones) may be improved (despite of the precision) by omitting the LBDC metric from the metric array considered by the AMA technique.

Further details about this technique can be found in [Dil04b] and [Dil01]. In conclusion, the method showed its effectiveness in detecting a significant percentage of clones.

Table 4: Results from clone analysis

	WA1	WA2	WA3	WA4
# HTML pages	55	23	23	-
# cloned HTML pages	18	2	8	-
# clusters of cloned HTML pages	3	1	4	-
# scripts in client pages	3	12	10	-
# cloned client scripts	3	4	2	-
# clusters of cloned client scripts	1	2	1	-
# functions in client pages	12	10	10	-
# cloned client functions	12	4	2	-
# clusters of cloned client functions	4	2	1	-
# forms in client pages	1	8	10	-
# cloned client forms	0	2	9	-
# clusters of cloned client forms	0	1	3	-
# tables in client pages	7	4	3	-
# cloned client tables	0	0	0	-
# clusters of cloned client tables	0	0	0	-
# ASP pages	19	73	37	71
# cloned ASP pages	2	15	3	2
# clusters of cloned ASP pages	1	6	1	1
# scripts in ASP pages	75	576	150	5341
# cloned ASP scripts	4	0	36	0
# clusters of cloned ASP scripts	2	0	3	0
# functions/subroutines in ASP pages	5	0	3	165
# cloned ASP functions/subroutines	0	0	0	9
# clusters of cloned ASP functions/subroutines	0	0	0	3

11.5 Future Works

The adoption of the methods presented in chapters 9, 10 and 11 allow a reduction of the effort related to the comprehension of the semantic of Web Application components.

The proposed techniques have been presented separately, but they can be also used in a collaborative way. As an example, the rank of concepts proposed by the concept assignment process may be considered as a further, good feature to the identification of Interaction Design Patterns. Moreover, frequent cloned structures identified by clone analysis technique may be considered as possible Interaction Design Patterns. The results of concept assignment method and clone analysis method may be used to classify similar Web pages in three categories:

- Pages with similar (or cloned) structure but different textual content. For these pages, a common template could be extracted and they could be reengineered according to this template;

- Pages with similar textual content but different structure. These pages can be clustered together according to the keyword clustering method proposed in [Ric04] and [Ton03];

- Pages with similar textual content and structure. They are probably cloned pages, obtained with a 'copy and paste' operation. A reengineering of these components will surely improve the maintainability of the Web Application.

Chapter 12: Identifying Cross Site Scripting Vulnerabilities

In this chapter a method is presented to assess the vulnerability of the Web pages of a Web Application with respect to Cross Site Scripting attacks. The method, based on some secure programming rules, exploits source code analysis to verify that those rules are actually present in the code. Moreover, a strategy to test the effectiveness of Cross Site Scripting attacks on vulnerable pages is presented. A case study, based on a real world Web Application is discussed.

12.1 Introduction

In the last years, the problem of ICT security has been devoted an increasing interest from industry, users and Public Administrations; a study conducted by Datamonitor [Dat03] shows that the global ICT market revenue estimated in 2006 is 13.5 billion USD, while in 2002 it was 7 billion USD.

Approaches to ICT security [Mai04] are mostly focused on:

- building perimeter defences around application;
- putting up reactive defence software or intrusion detection systems.

Therefore, the effort in implementation of security is mainly concentrated in buying specific security systems (such as firewalls, or IDS) and software (such as antivirus or encryption software), and taking organisational changes to business processes finalised to improve security.

However, as tools become more sophisticated and as corporate networks and applications are more interconnected, open, and distributed, just defining and defending the perimeter may be insufficient [Mai04]. More effective approaches require that the application security is achieved building it inside the application's code, that is using secure programming practices. However, the application security is frequently overlooked. Main reasons for overlooking it are that the teams involved in the development do not communicate with one another; also, developers do not build security into their applications, based upon the false assumption that another area of security will cover it (database, network, firewalls, etc.) [Sto03].

The problem of security is particularly relevant for Web Applications [Off02]. A possible classification of security attacks affecting a Web Application distinguishes six basic categories [Sto03]:

1. Parameter tampering
2. Hidden field manipulation

3. SQL injection
4. Application design/business logic
5. Debug options
6. Common vulnerabilities

In the category of common vulnerabilities, Cross Site Scripting (XSS) is well known in the Internet community from several years ([Cert00], [Itsd]). XSS is a vulnerability of a Web Application caused by the failure of the application in checking up on user input before returning it to client's web browsers. Without an adequate validation, user input may include malicious scripting code that may be sent to other clients and unexpectedly executed by their browsers, thus causing a security exploit (cfr. subsection 12.2.2). XSS vulnerability is a relevant issue for the following reasons:

- many web sites are vulnerable to this attack (e.g., Ohmaki says that 80% of Japanese e-commerce sites are vulnerable to XSS [Ohm02]);
- the exploits are very simple to carry out, and no particular application knowledge or skill are required;
- the attacks may bypass perimeter defences (e.g. Firewalls), cryptography, digital signatures and site trusting;
- it may be very difficult for the victim to know which web application allowed the XSS attack;
- it may be very difficult for the developer to know which element of the web application allowed the XSS attack;
- evolution of hypertextual language characteristics and browser capabilities may make it possible new attack strategies and make vulnerable a web application which was considered invulnerable.

Several solutions have been proposed to defend an application from XSS attacks. A first recommended solution for the users of Web Applications consists of disabling scripting languages in their browsers, and avoiding promiscuous browsing on untrustworthy web sites. However, rather than by users, the problem should be addressed by web page developers who should check up on the input received by a page, and encode or filter the output returned to the user. More precisely, a possible remedial action to avoid XSS vulnerability, would consist of introducing an input validation function immediately after every input statement contained in a Web page. However, only the input data that will affect output data will cause a XSS vulnerability. Therefore, an effective method for detecting XSS vulnerabilities in a Web Application requires that just input data affecting output data must be analysed.

12.2. Background

12.2.1 Related works

The problem of vulnerability of software systems, and in particular, of their user interfaces, has always been considered very important for traditional applications, but the importance is growing quickly with the diffusion of web applications, in which a large amount of users may use the application in nearly anonymous way. Several approaches have been proposed in the literature to detect vulnerabilities in existing applications. In particular, several vulnerabilities are due to a lack of validation of the user input of an application, like XSS. As an example, buffer overflow vulnerability has been deeply studied, and different approaches have been proposed to detect and fix it ([Lar01], [Sha01], [Flaw], [RATS], [ITS4], [Mops], [Cow98], [Cow01]).

Recently, [DaC03] addresses buffer overflow in C and C++ software, proposing an approach based on static analysis to individuate the so-called FLF (Front Line Functions), that are functions in which a validation test of input and/or output data is needed.

A general approach to the problem of detecting vulnerabilities in Web Applications can be found in [Hua03]: this approach exploits fault injection and is based on a dynamic analysis of the application in order to deduce the presence of vulnerabilities in the application.

Another detection technique is based on the installation of a software proxy between the web application server and the network, which intercepts malicious strings in input and/or output [Str02]. According to this approach, a commercial product has been realised, where the security policy is written in an XML-based language, called SPML (Security Policy Markup Language).

Moreover, [Sco02a] and [Sco02b] describe a proxy-based approach to prevent XSS attacks, which intercepts the http queries in input to a server page and the html response in output from a server page. In this way, it is possible to simulate XSS attacks in order to verify if a web application is vulnerable or not. This approach is promising, but requires knowledge of all the elements potentially vulnerable of the application, and for this reason has been used only for web application implemented with a specific language [Big01].

12.2.2 Cross site scripting (XSS)

According to [Cgi], there is a Cross-Site Scripting attack when a client executes a page containing script code that has been injected from other sources.

In [Idef03], details are provided about how XSS attacks may be carried out bypassing the “traditional” defensive barriers coded by programmers.

A complete review on XSS attacks and defence techniques may be found in [Mic00], where also a complete list of possible effects resulting after an XSS attack are described.

Some programming tricks that may be used during the development of Web Applications to avoid XSS vulnerability are described in [Whe02].

In [Cert00], solutions that may be carried out directly by users are proposed. One of the proposed solutions recommends to disable script execution by browsers. However, this solution is often inapplicable because it reduces the functionality executable by the client side of the application. Rather, an effective attack prevention must be implemented by the server side of the application.

There are two basic techniques to accomplish a XSS attack. The first one requires that malicious code is stored into a database, and therefore, once retrieved, it will finish to be executed by a client browser.

The first technique [Cert00] is exemplified by the web application shown in Fig. 12.1. The application implements a guestbook, in which a visitor may write a message (using the page sign.html), which is stored into the database by the page sign.asp. The message may be displayed by other users (using the guestbook.asp page), retrieving it from the database.

```
Sign.html
<form method="post" action="sign.asp">
    <textarea name="txtMessage"></textarea>
    <input type="submit" value="Sign!">
</form>
```

```
Sign.asp
<% Message=request.form("txtMessage")
   conn=OpenDBConnection
   set rs=server.createobject("Adodb.recordset")
   rs.open "Guestbook",conn,1,2,2
   rs.Addnew
   rs("Message")=Message
   rs.update
%>
```

```
Guestbook.asp
<% conn=OpenDBConnection
   rs=server.createobject("Adodb.recordset")
   rs.open "SELECT Message FROM GuestBook" ,conn,3,3
%>
<table>
<% rs.movefirst
   while not(rs.eof)
     response.write (rs.fields("Message"))
     rs.movenext
   wend
%>
</table>
<% rs.close
   set rs=nothing
   conn.close
   set conn=nothing
%>
```

Figure 1: a Web Application implementing a guestbook

A XSS attack can be performed if an attacker submits a string like the one shown in Fig.12.2 to sign.asp page: when a user will open guestbook.asp web page, values of its cookies are sent to the attacker.

```
<script>location.URL= ‘http://www.attackersite.com/attacker.cgi?’ +
document.cookie </script>
```

Figure 12.2: A XSS attack string

The second technique [Cert00] requires that the victim unconsciously executes a link containing itself malicious code (there are sophisticated techniques to hide the real content of a link to an inexpert user). As an example, let us consider the link shown in Fig.12.3, where a user would query the database to search for a word:

```
<A HREF=http://www.site.com/search.asp?Word= <script> malicious code </script> >
```

Figure 12.3: A link containing malicious code

The server page ‘search.asp’ (Figure 12.4), will send to the client the response to the query but also the malicious code, which is executed by the user browser.

```
<% word=request.querystring(“Word”)
    // find if Word is in db
   else
     response.write (“The word “+Word+” isn’t in the DB”)
   end if
%>
```

Figure 12.4: Server page search.asp

Since XSS vulnerability is due to a lack of validation of user input, preventing vulnerability of an existing application would require that each input is submitted to a validation function. However, for an effective approach just input data affecting output data should be analysed.

12.3. Detection of XSS vulnerabilities

In this section, a technique for detecting vulnerabilities in a Web Application that exploits both static and dynamic analysis of the source code is presented. Static analysis is used to determine whether a server page is vulnerable to a XSS attack. Dynamic analysis is exploited to verify whether a Web Application with vulnerable server pages identified by static analysis is actually vulnerable.

12.3.1 Assessing a server page XSS vulnerability by static analysis

Although a Web Application is composed both by client and server pages, its vulnerability is due to server pages, since they are responsible for receiving the input data and outputting them.

In order to detect the vulnerabilities of a server page, let’s consider its Control Flow Graph, CFG, and associate a label with the CFG nodes corresponding to statements performing input, output, definition or use of data variables.

In particular, a CFG node is labelled as Input(*v*) if the corresponding statement assigns a value to a variable *v* that depends on a user input, a cookie, a database field value, or a data value read from a file.

In the same way, a node is labelled as Output(*v*) if the corresponding statement outputs the value of the variable *v* in a file, a database, a cookie or a built client page.

As an example, possible Input nodes are those associated with statements performing input of data from a HTML form, or reading the value of a query string, of a cookie, a database field, or any data from a file. Output nodes are associated with statements writing a database field, a file, a cookie, or the built client page.

The server page is *potentially* XSS vulnerable if there are a variable *v* and two Input(*v*) and Output(*v*) nodes that are connected by a CFG path. More precisely, a server page is *vulnerable* if there are a variable *v*, and two Input(*v*) and Output(*v*) nodes, such that all the CFG paths leaving the Input(*v*) node reach the Output(*v*) node, being def-clear with respect to *v*. Finally, a server page including an input data item that does not affect any output is certainly *invulnerable* with respect to that input.

Figure 12.5-a shows an example of a server page vulnerable with respect to the variable ‘Message’. Fig. 5-e represents the case of an invulnerable server page with respect to the variable ‘Message’.

Figures 12.5-b, 12.5-c and 12.5-d show three examples of potentially vulnerable pages, whose actual vulnerability can be assessed by a further analysis of the source code. In particular, in Figure 12.5-b, an input (i.e. the variable ‘Message’) becomes an output with some intermediate redefinition of the input; this case requires that the semantic of the statement redefining the input is analysed in order to verify if it corresponds to an input validation.

Figure 12.5-c shows the case of a CFG with no intermediate redefinition of the input, but where an input may affect an output or not, depending on a selection statement; also in this case, it’s needed to verify if the selection statement implements an input validation.

Figure 12.5-d illustrates the case of a CFG where either an input affects an output (eventually being redefined), or the input does not affects an output, depending on a selection statement; also in this case a semantic analysis is needed to verify if any input validation is done.

Given a variable *v* of a server page P, and two CFG nodes I and O, where I is labelled as Input(*v*) and O as Output(*v*), the following predicates are introduced in order to define some rules for assessing the vulnerability of a server page:

A(*v*): There exists a path on the CFG between I and O nodes.

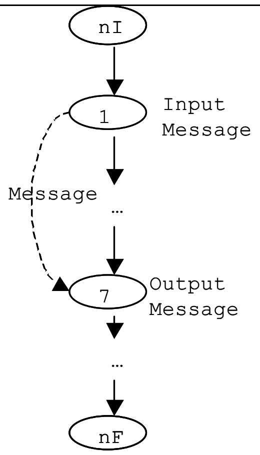
B(*v*): The O node postdominates I node.

C(*v*): Each path between I node and O node is a def-clear-path.

The following implications are verified, for each variable *v*:

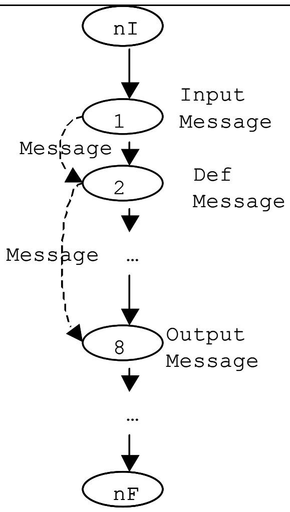
B(*v*) => A(*v*)

C(*v*) => A(*v*)



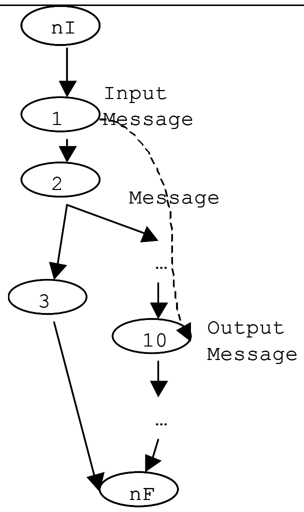
```
nI  <%
    ‘Read Message from input form
1   Message=request.form("txtMessage")
    . . . open DB connection . . .
5   rs.open "Guestbook",conn,1,2,2
    ‘Store Message into the DB
6   rs.Addnew
7   rs("Message")=Message
8   rs.update
nF  %>
```

Figure 12.5-a: Server Page 1



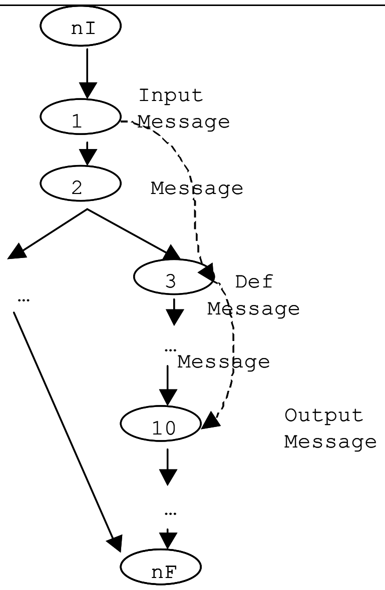
```
nI  <%
    ‘Read Message from input form
1   Message=request.form("txtMessage")
2   ‘Encode Message value
    Message=server.HTMLEncode(Message)
    . . . open DB connection . . .
6   rs.open "Guestbook",conn,1,2,2
    ‘Store Message into the DB
7   rs.Addnew
8   rs("Message")=Message
9   rs.update
nF  %>
```

Figure 12.5-b: Server Page 2



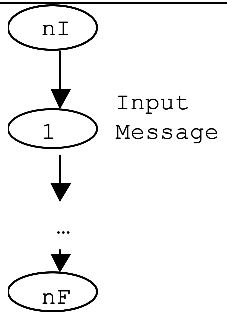
```
nI   <%
     ‘Read Message from input form
1    Message=request.form("txtMessage")
     ‘Check for string “script” in Message
2    if instr(1,Message,“script")>0 then
3      response.write(“Forbidden”)
4    else
       . . . open DB connection . . .
8      rs.open "Guestbook",conn,1,2,2
       ‘Store Message into the DB
9      rs.Addnew
10     rs("Message")=Message
11     rs.update
nF   end if
     %>
```

Figure 12.5-c: Server Page 3



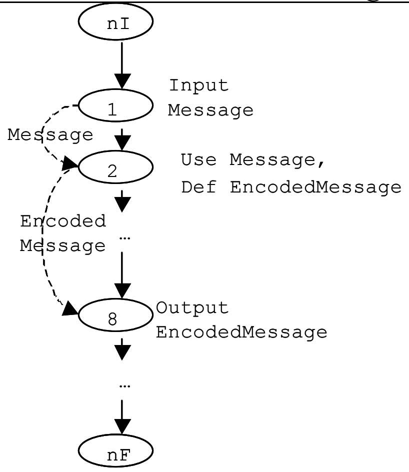
```
nI  <%
    'Read Message from input form
1   Message=request.form("txtMessage")
    'Check for string "script" in Message
2   if instr(1,Message,"script")>0 then
       'Encode Message value
3      Message=server.HTMLEncode(Message)
       . . . open DB connection . . .
7      rs.open "Guestbook",conn,1,2,2
       'Store Message into the DB
8      rs.Addnew
9      rs("Message")=Message
10     rs.update
     else
11     response.write("Malicious Request!")
nF  %>
```

Figure 12.5-d: Server Page 4



```
nI  <%
    'Read Message from input form
1   Message=request.form("txtMessage")
    . . . open DB connection . . .
5   rs.open "Guestbook",conn,1,2,2
    'Store a constant string into the DB
6   rs.Addnew
7   rs("Message")="One message received"
8   rs.update
nF  %>
```

Figure 12.5-e: Server Page 5



```
nI  <%
    'Read Message from input form
1   Message=request.form("txtMessage")
    'Encode Message value in EncodedMessage variable
2   EncodedMessage=server.HTMLEncode(Message)
…   . . . open DB connection . . .
7   rs.open "Guestbook",conn,1,2,2
    'Store EncodedMessage into the DB
8   rs.Addnew
9   rs("Message")=EncodedMessage
10  rs.update
nF  %>
```

Figure 12.5-f: Server Page 6

Figure 12.5-a,b,c,d,e,f – Examples of server pages with labelled Control Flow Graphs

These predicates can be used to characterize the vulnerability of the server page P by the following conditions PV, V and NV:

PV) $\exists v \in$ P: A(v) => P is potentially vulnerable with respect to v => P is potentially vulnerable

V) $\exists v \in$ P: B(v) AND C(v) => P is vulnerable with respect to v => P is vulnerable

NV) $\exists v \in$ P: NOT(A(v)) => P is not vulnerable with respect to v

With respect to the first five server pages reported in Figure 12.5, Table 12.1 shows the truth values of the predicates A, B, and C, and of conditions PV, V, and NV with respect to the variable 'Message'. These examples cover all feasible combinations of the predicates A, B, and C.

Table 12.1 : Predicates and Conditions values for Server Pages shown in Figure 12.5-a,b,c,d,e

Server Page	Predicate values			Condition			Input variable
	A	B	C	V	PV	NV	
1	T	T	T	T	T	F	Message
2	T	T	F	F	T	F	Message
3	T	F	T	F	T	F	Message
4	T	F	F	F	T	F	Message
5	F	F	F	F	F	T	Message

The server page in Figure 12.5-f is an example of a page that is potentially vulnerable with respect to the input variable *Message,* since the output variable *EncodedMessage* directly depends on the input variable *Message* which has not been redefined before being assigned to the output variable.

A five-step process allowing the vulnerability of a server page P to be assessed can be, therefore, proposed:

- Identify the input and output nodes of the CFG of the page P;
- Identify all paths leaving the input nodes on the CFG;
- For each path leaving an input(v) node and reaching an output(v) node, verify if the path is def-clear with respect to v;
- Evaluates A, B, and C predicates' values with respect to v;
- Evaluate the vulnerability of page P, by the PV, NV, and V conditions.

With reference to the second step of this process, in order to cope with the complexity of identifying all paths leaving the input nodes on the CFG, the analysis can be limited to a set of linearly independent paths extracted from the CFG.

12.3.2 Vulnerability of a Web Application

The vulnerability of a server page is a necessary condition for the vulnerability of a Web Application, but it isn't a sufficient condition.

For instance, a server page may send its output not directly to a client browser, but to another server page or to a persistent data store, such as a file, or a database. In these cases, if the components receiving the page's output will validate it, the Web Application is protected from a possible XSS attack. In this case, the vulnerability of the server page does not imply the vulnerability of the Web Application.

As an example, let's consider the XSS attack carried out according to the first technique described in section 12.2.2. Vulnerability of that Web Application depends on: the vulnerability of server page sign.asp, which stores user input messages in a database; on the vulnerability of server page guestbook.asp, which sends messages retrieved in the database to a user; on the vulnerability of the database. In fact, if database has a sanitization mechanism (e.g. an encoding method), Web Application would be not vulnerable to a XSS attack carried out according to the first technique.

In these cases, assessing the vulnerability of a Web Application entails that not only the single server pages, but all the software components that are interconnected with the pages are taken into account. An effective method to detect the Web Application vulnerability may involve dynamic analysis. In the next section, such a method is proposed.

12.3.3 Using dynamic analysis for assessing a Web Application vulnerability

Even if static analysis is able to detect vulnerable, or potentially vulnerable server pages, it is not able to establish whether the Web Application is actually vulnerable to a XSS attack. Indeed, some security mechanisms implemented either by the web server of the application, or by other software components, such as a software gateway, may make the Web Application invulnerable.

Dynamic analysis based on the design and execution of XSS attacks may be used to determine if a Web Application is actually vulnerable. However, dynamic analysis cannot establish the invulnerability of a Web Application, since there may be any XSS attack strategy revealing vulnerability, which might not be taken into account in the test-case design.

Two different dynamic analysis strategies are proposed to reveal vulnerability of Web Applications with respect to the two different XSS attacks exemplified in section 12.2.2. Both strategies exploit static analysis results in order to detect vulnerable or potentially vulnerable pages, and input variables causing the vulnerabilities. Therefore, only these pages are submitted to a XSS attack for each variable causing the vulnerability. A set of XSS attack strings, such as the ones proposed in [Idef03], [Mic00], [Cert00], [Ohm02], [cgi], or published in bugtraq repositories, such as [BugT1] or [BugT2] should be designed for each input variable and submitted to the Web Application during dynamic analysis. Therefore, the results of the analysis are checked in order to assess whether the attack is successful or not. A possible successful attack accomplishes the theft of the values of the set of client cookies, which are sent to the attacker's server. Another common attack will cause the forced loading in the client browser of an attacker web page, which reproduces a trustworthy web page, and where the user might insert private data that are submitted to the attacker's server.

The following algorithm describes the proposed testing strategy:

```
FOR EACH vulnerable or potentially vulnerable page P of the Web Application
     FOR EACH input field I of page P causing
     vulnerability
       Define a set S of XSS attack strings
       FOR EACH s ∈ S
          EXECUTE server page P with
          input field I=s
          Check for attack consequences
```

Vulnerabilities of the first type are more difficult to be detected, because a XSS attack will have to inject malicious data that will not be directly provided to the user, but are stored in a persistent data store. Therefore, effects of a XSS attack may be observed only if another functionality of the Web Application will send the injected data to the final user. Consequently, after an attack has been accomplished, all the Web Application functions that read data from a persistent data store should be exercised in order to discover attack's consequences. A test suite covering these functions may be executed to reach this aim.

The following algorithm describes the proposed testing strategy:

```
FOR EACH vulnerable or potentially vulnerable page P of the Web Application
     FOR EACH input field I of page P causing
     vulnerability
    Define a set S of XSS attack strings
      FOR EACH s ∈ S
         EXECUTE server page P with
         input field I= s
        FOR EACH test case T from the test suite
        EXECUTE test case T
        Check for attack consequences
```

This strategy may be supported by a Web Application testing tool that automatically executes the test cases. Some tools have been proposed in the literature to support testing of Web Applications, such as VeryWeb [Ben02], and TestWeb [Ric01]. The testing tool WATT (Web Application Testing Tool) [Dil02b] has been used to carry out the vulnerability testing according to the proposed strategy. This tool has been interfaced with a XSS Test Case Generator module, which generates automatically XSS attack test cases, and stores them in the WATT Test Case Repository. WATT has been used to execute the attacks, and therefore to exercise the Web Application with a suitable test suite. The results of the test execution were checked in order to assess the success of the attack.

Figure 12.6 shows how the XSS test case generator, and the WATT tool can be used to automatically support dynamic analysis.

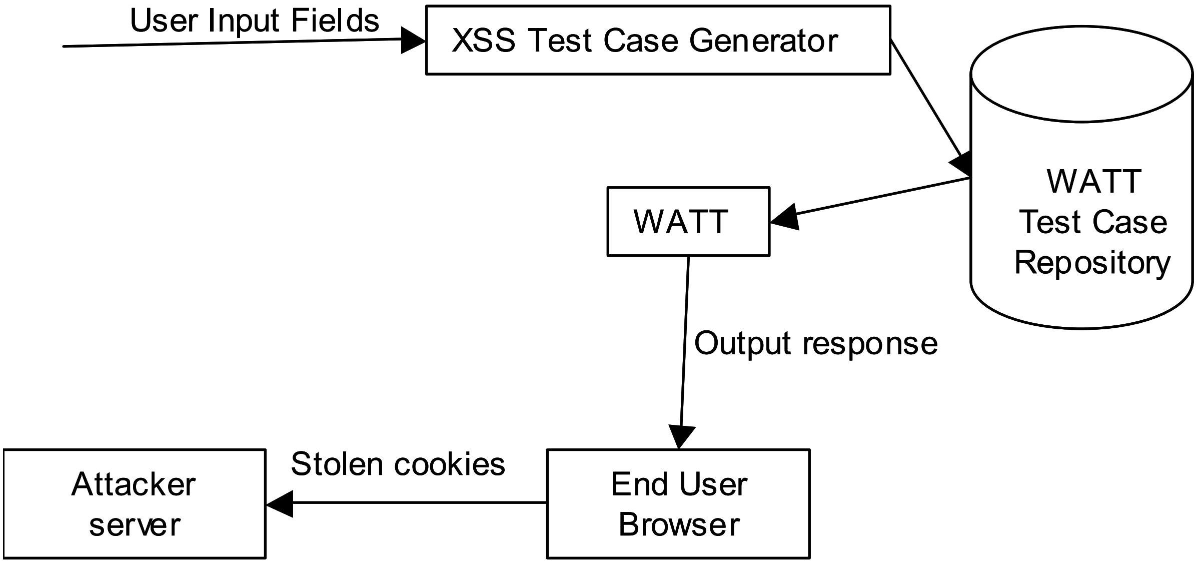


Figure 12.6: XSS testing strategy

12.4 A case study

In order to assess the effectiveness of the proposed method, a number of web applications implemented with server scripting languages, such as ASP and PHP, have been submitted to the vulnerability analysis. Open source web applications with declared XSS vulnerabilities have been searched for, in order to verify if the method is able to discover the declared vulnerability. Bugtraq web sites, such as [BugT1] and [BugT2], have been queried to select suitable applications. As an example, the 3.4.03 release of the ‘Snitz Forum’ [SnF] application is declared as vulnerable to XSS attacks of the first type, and therefore it has been considered for the experiment.

Snitz Forum implements a discussion forum providing several user functions. The search.asp page, that interfaces the user and accepts his requests of searching for a word or a sentence among the messages stored in the forum, was analysed. This page was vulnerable to the second type of XSS attacks.

During the experiment, this page has been submitted to the static analysis process described in Section 12.3.1. The page resulted vulnerable with respect to the input variable Search that is read and written by the statement reported in Fig. 12.7. Then, this page satisfies predicates A, B, and C with respect to the variable Search.

```
Response.Write  “<input  type=""text""  name=""Search""  size=""40""  value="""  &
Request.QueryString("Search") & """><br />" & vbNewLine
```

Figure 12.7: a fragment of server page search.asp (Snitz Forum 3.4.03)

The dynamic analysis confirmed that the Web Application was actually vulnerable with respect to this variable. Table 12.2 reports the test case used to carry out the exploit: the input string including the malicious script that showed the vulnerability is reported.

Table 12.2

Test case	Input Variable	Expected Output Action
	Search	
1	“><script>location.URL= ‘http://www.attackersite.com/attacker.cgi?’ + document.cookie) </script>	Client Cookie redirected to a page of attacker’s server

Static analysis revealed also two data variables causing potential vulnerability. Figure 8 reports an excerpt of the page source code, and figure 12.9 shows the CFG of the page. The Input variable causing potential vulnerability was the ‘rs’ recordset variable, which is assigned a value by the statement at line 7. This variable includes two fields that affect indirectly, through the allMemberData variable (assigned in line 11), the MembersMemberID variable that is defined at line 20 and outputted in line 22, and the MembersMemberName variable, defined at line 21 and outputted in line 22. On the CFG in figure 9 the data dependences between these variables have been represented by dashed lines.

This page is potentially vulnerable with respect to the input variable ‘rs’, since ‘rs’ satisfies the A condition (i.e., there is a path between nodes 7 and 22). On the other side, B and C conditions are not true (B is false since there are more paths from the input node 7 that do not reach the output node 22, while C is false because ‘rs’ is redefined at line 11). The semantic analysis of the source code shows that no effective validation of variables ‘rs’ and ‘MembersMemberID’ is implemented in the code, so any malicious data from input will reach the output node. On the other hand, the variable MembersMemberName is checked by the ChkString user function (cfr. line 22), that implements an effective output validation, according to the semantic analysis of the code that has been carried out.

Dynamic analysis was, therefore, carried out in order to assess the actual vulnerability of the Web Application with respect to these two variables. Although the potential vulnerability, no attack string succeeded, due to a security mechanism embedded in the DBMS, that accept only integer values for the field whose value is assigned to the variable ‘MembersMemberID’.

```
1  ,## Forum_SQL
2  strSql = “SELECT MEMBER_ID, M_NAME “
3  strSql = strSql & “ FROM “ & strMemberTablePrefix & “MEMBERS”
4  strSql = strSql & “ WHERE M_STATUS = “ & 1
5  strSql = strSql & “ ORDER BY M_NAME ASC;”
6  set rs = Server.CreateObject (“ADODB.Recordset”)
7  rs.open strSql, my_Conn, adOpenForwardOnly, adLockReadOnly, adCmdText
8  if rs.EOF then
9    recMemberCount = “”
10 else
11   allMemberData = rs.GetRows(adGetRowsRest)
12   recMemberCount = Ubound(allMemberData,2)
13   meMEMBER_ID = 0
14   meM_NAME = 1
15 end if
16 rs.close
17 set rs = nothing
18 if recMemberCount <> “” then
19   for iMember = 0 to recMemberCount
20     MembersMemberID = allMemberData(meMEMBER_ID, iMember)
21     MembersMemberName = allMemberData(meM_NAME, iMember)
22 Response.Write    “<option   value=”””   &   MembersMemberID   &   “””>   “   &
   ChkString(MembersMemberName,”display”) & “</option>” & vbNewline
23   next
24 end if
```

Figure 12.8: a fragment of server page search.asp (Snitz Forum 3.4.03)

This second case of vulnerability was not declared by the Bugtraq sites (since it reports only actually exploited vulnerabilities), while the proposed method was able to detect it. This was a valuable result, since the potential vulnerability might become an actual one, if the type of the involved database field was changed from integer to string.

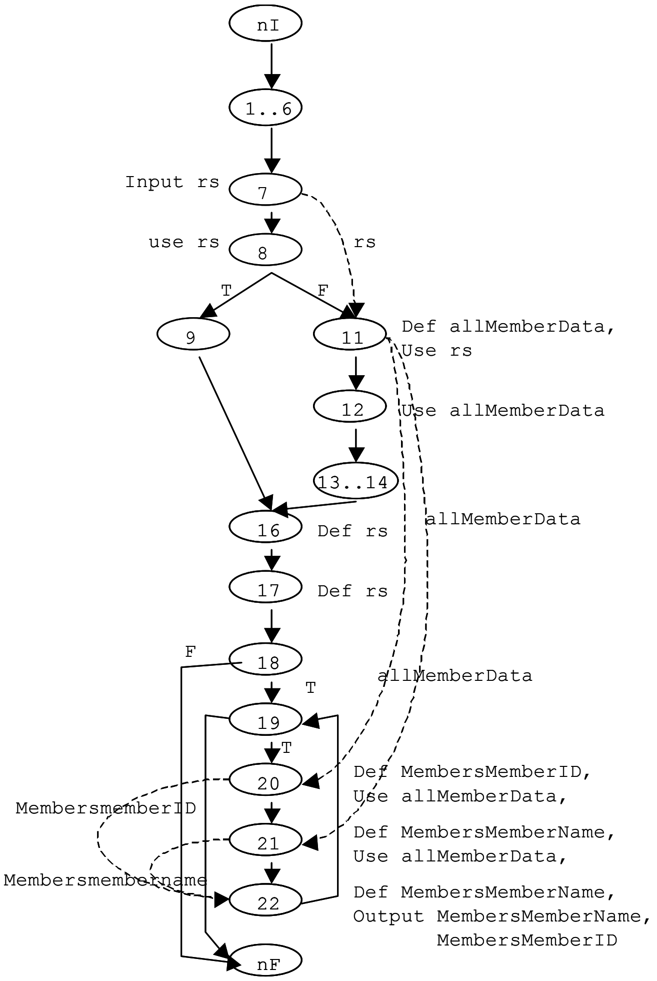


Figure 12.9: Labelled CFG of the fragment of search.asp in figure ff

As a further validation, the code of the search.asp page from the 3.4.04 release of the Snitz Forum Web Application has been analysed. In this release, the code showed in Fig. 12.7 has been corrected as it is reported in Figure 12.10. Static analysis revealed that predicates A and B were true with respect to variable Search, while predicate C was false because the value of the output variable was modified by functions trim and ChkString. The page was therefore potentially vulnerable, according to this method, and the vulnerability depended on the ChkString validation function. Dynamic analysis was not able to exploit the application.

```
Response.Write ″      "<td   bgColor="""   &   strPopUpTableColor   &   """
align=""left""   valign=""middle"">   <input   type=""text""   name=""Search""
size=""40"" value=""" & trim(ChkString(Request.QueryString("Search"),"display"))
& """><br />" & vbNewLine & _
```

Figure 12.10: a fragment of server page search.asp (Snitz Forum 3.4.04)

Further details about this technique can be found in [Dil04d].

Chapter 13: A maintainability model for Web Applications

In this chapter a maintainability model of a Web Application is presented. This model is an adaptation of the one proposed by Oman and Hagemeister [Oma92] for traditional software systems. New metrics have been defined while existing metrics are adapted to Web Applications. These metrics can be automatically evaluated from the information recovered by the Reverse Engineering approach described in the previous chapters. A case study reporting the value of these metrics for some Web Applications under analysis is discussed, with the aim to compare the maintainability of these Web Applications.

13.1 Introduction

As already discussed in this Thesis, technologies and development techniques that are used to realize a Web Application make them harder to maintain (cfr. Chapter 7). To avoid a software crisis for the Web Applications there is a strong need to urgently address the definition and the experimentation of methodological approaches, techniques and tools supporting an effective maintenance of existing Web Applications. Analogously, there is a strong need also for methods and models to assess the maintainability of existing Web Applications in order to have a valuable support to successfully estimate the effort of a maintenance intervention.

While dealing with Web Application's maintainability assessment, the first step to achieve is the definition of software attributes affecting maintainability; the related model for such Web Applications is consequently carried out.

Web Applications are substantially different from traditional software systems; thus, models and metrics defined for traditional applications cannot be ever applied to Web Applications, because they could not fit well in describing those new features (such as the hyper-textual based structure, or the usage of several technologies and programming languages to code web pages, or the dynamic generation of HTML pages) characterizing Web Applications' and affecting Web Applications' maintainability

Since a few years, Software Engineering Research Community has been dealing with problems about Web Application metrics in order to estimate Web Application developing efforts and evaluate some Web Application quality attributes.

In [Men01] some web metrics are defined and used to generate models for estimating, in the early phases of development, the effort to design a Web Application. The problem to estimate web based software development and duration is discussed also in [Rei00], where new sizing metrics, derived by metrics used for traditional software and adapted to the new Web Applications context, are defined and used in an estimation model called WebMo, derived as an adaptation of the CoCoMo II model [Boe00].

Jeff Offutt, in [Off02] discusses about the differences among traditional applications and Web Applications and indicates the main quality attributes to be considered for Web Applications and which are the main Web Application features that can affect them.

The maintainability is one of the critical aspects of a Web Application: Web Applications have to be modified and evolve in a very fast way, then those features affecting it should be defined, identified and evaluated in order to improve/reduce the ones that have a positive/negative impact on the maintainability both during the development and maintenance process of a Web Application.

Unfortunately, there are a very few works in the literature addressing the problem of assessing the Web Application maintainability, even if in [Bre98] web based applications were considered as the 'next maintenance mountain'.

In this chapter a maintainability model for Web Applications is proposed. The model is derived from the maintainability model proposed in [Oma92] by Oman and Hagemeister for traditional systems. This model is adapted to Web Applications, considering architectural and structural peculiarities that make Web Applications different from traditional systems. Proper metrics are defined in order to carry out an estimation of Web Application maintainability that can be expressed as a function of those metrics.

The current state of this research does let to compare maintainability of different Web Applications, but evaluating the maintainability level of a Web Application from an absolute viewpoint is not yet possible. Collecting and analysing data from appropriate experiments is needed in order to defining exactly coefficients of each metric to compose the maintainability function.

This model doesn’t take into account all the aspects of maintenance, in terms of phases and documentation. This is due, mainly, to the following reasons.

The major aim is to support decision-making: the model is just a prediction tool whose required effort is neglectable.

Model is focused on the source code: in the most critical cases, it is the only available documentation; in the usual web application lifecycle, the source code is the most handled document.

13.2 A maintainability model for Web Applications

Many papers discussing maintainability models for traditional software systems are present in literature. Among these, the one proposed by Oman and Hagemeister seems to us the most exhaustive and complete. It has been chosen as reference model for deriving a suitable one to define the maintainability of a Web Application.

13.2.1 The Oman and Hagemeister maintainability model

In [Oma92], Oman and Hagemeister presented a maintainability model based on a hierarchical tree structure comprehending 92 attributes affecting the maintainability of a software system. The leaf nodes in the hierarchy represent an identified maintainability attribute and, for each of these, attribute metrics are defined to evaluate that maintainability characteristic.

In Figure 13.1 the top level of the OHMM hierarchy is showed. At this level, three main categories of factors are pointed out:

- Management: practices of management employed, and facts related with them;
- Operational environment: environment, in terms of hardware and software, involved in the operation of the system under examination;
- Target Software System: the examined software system under maintenance, including the source code and support documentation.

Oman's work focuses mainly on the Target Software System; Figure 13.2 shows a detail of the sub-tree concerning this category.

Three major categories can be identified in this sub-tree:

- Maturity Attributes: maintainability characteristics referring to the maturity degree of the system under evaluation, relying on the aging, stability, reliability, number of defects; and number of maintenance interventions, techniques of development used;
- Source Code: maintainability characteristics due to those ones of the source code;
- Supporting Documentation: maintainability characteristics due to the supporting documentation; they are divided in two categories:
- Documentation Abstraction: characteristics related with content (completeness, correctness, and descriptiveness) of supporting documents set;
- Physical Attributes: characteristics related with the form (readiness, modifiability) of supporting documents set.

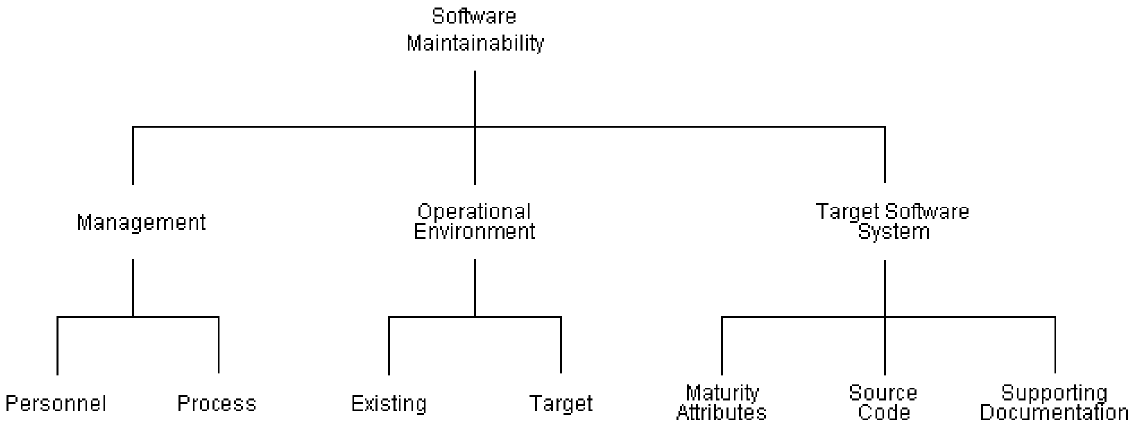


Figure 13.1 The Oman and Hagemeister top level Software Maintainability hierarchy (extracted from [Oma92])

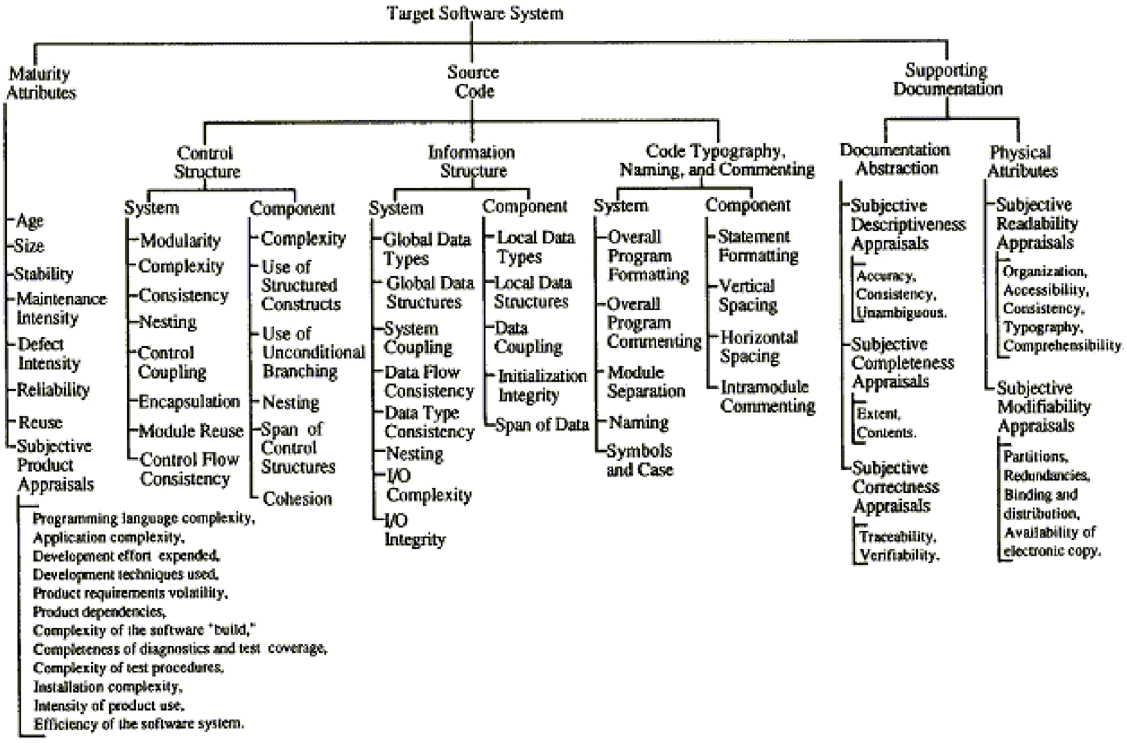

Target Software System
Maturity Attributes
Age
Size
Stability
Maintenance Intensity
Defect Intensity
Reliability
Reuse
Subjective Product Appraisals
Programming language complexity,
Application complexity,
Development effort expended,
Development techniques used,
Product requirements volatility,
Product dependencies,
Complexity of the software "build,"
Completeness of diagnostics and test coverage,
Complexity of test procedures,
Installation complexity,
Intensity of product use,
Efficiency of the software system.
Source Code
Control Structure
System
Modularity
Complexity
Consistency
Nesting
Control Coupling
Encapsulation
Module Reuse
Control Flow Consistency
Component
Complexity
Use of Structured Constructs
Use of Unconditional Branching
Nesting
Span of Control Structures
Cohesion
Information Structure
System
Global Data Types
Global Data Structures
System Coupling
Data Flow Consistency
Data Type Consistency
Nesting
I/O Complexity
I/O Integrity
Component
Local Data Types
Local Data Structures
Data Coupling
Initialization Integrity
Span of Data
Code Typography, Naming, and Commenting
System
Overall Program Formatting
Overall Program Commenting
Module Separation
Naming
Symbols and Case
Component
Statement Formatting
Vertical Spacing
Horizontal Spacing
Intramodule Commenting
Supporting Documentation
Documentation Abstraction
Subjective Descriptiveness Appraisals
Accuracy, Consistency, Unambiguous.
Subjective Completeness Appraisals
Extent, Contents.
Subjective Correctness Appraisals
Traceability, Verifiability.
Physical Attributes
Subjective Readability Appraisals
Organization, Accessibility, Consistency, Typography, Comprehensibility
Subjective Modifiability Appraisals
Partitions, Redundancies, Binding and distribution, Availability of electronic copy.

Figure 13.2 The Target Software System subtree of the Oman and Hagemeister Software Maintainability hierarchy (extracted from [Oma92])

The Source Code category is divided in three sub-categories, each divided in two sub-categories (i.e. System and Component) in order to distinguish attributes characterizing either the overall system or single components forming it.

- Control Structure: it includes characteristics concerning the way either the system or the program is decomposed (characteristics of the implementation, or code)

- System: characteristics concerning inter-modular control attributes, specifically referring to the way system is decomposed in modules, the way modules are coupled, and the way in which algorithms are implemented.

- Component: characteristics concerning attributes about intra-modular control flow and module execution.

- Information Structure: includes characteristics concerning choice and use of data structures and inter- and intra-modular data flow:

- System: characteristics of information regarding the memorization and flow of data in the system, as global data types, global data structures, nesting of data structures;

- Component: characteristics of information, at the level of a single module, concerning the memorization and manipulation of data within a system module, such as local data types, local data structure, coupling.

- Code typography, naming and commenting: including characteristics about the typographical layout, names and comments of the code:

- System: Characteristics related to typographical layout of source code and to comments related to the overall system as general formatting of programs, conventions for the names, and module separation.

- Component: characteristics concerning the typographical layout of source code and comments, at level of a single module, such as vertical spacing, comments in the module, indenting of statements.

The current work focuses on the category of the Source Code and, more specifically, on the sub-categories of Control and Information Structure. Adaptation of attributes and metrics concerning these categories to the case of Web Applications is discussed in the following.

These adaptations are influenced by and refer to components typically forming a Web Application. Components forming a Web Application, and the main relationships among them, are introduced and discussed in the next section.

13.2.2 Traditional Systems and Web Applications

The OHMM was developed with reference to traditional (Legacy) systems. These systems are usually composed of a set of programs (or modules), linked by calling relationships and different kinds of data coupling, and running just on one centralized computer. Moreover, the programs are usually coded just using one programming language (or at least very few different programming languages) and they are executed in a static way (i.e. no program or code component is created at run-time).

On the contrary, Web Applications are usually composed of different kinds of items coded with different programming/scripting languages; Web Applications' components may be executed on different computers in a distributed architecture, and some components can be generated at run-time; among Web Applications' components may exist different kinds of relationships connecting them.

13.3 Adapting Oman and Hagemeister maintainability model to Web Applications

The differences between traditional systems and Web Applications have to be considered to apply the OHMM to Web Applications: the original model has to be modified to be efficiently and effectively used with Web Applications.

In the following the proposed modifications to apply to the OHMM for estimating the maintainability of a Web Application are reported.

Table 13.1 Web Application Metrics at System Level

Metric Name	Description
TotalWebPage#	Total number of Web Application pages TotalWebPage#= TotalServerPage# + TotalStaticClientPage#
TotalLOC#	Total number of Web Application LOCs TotalLOC# = TotalServerPageLOC# + TotalStaticClientLOC#
ServerScript#	Total number of Web Application server scripts
ClientScript#	Total number of Web Application client scripts
WebObject#	Total number of Web Application web objects
InterfaceObject#	Total number of Web Application Interface Object
TotalData#	Total number of different data identifiers
I/OField#	Total number of Web Application form fields + number of I/O data from/to mass storage devices
TotalConnectivity#	Total number of relationships among web pages TotalConnectivity# = Link# + Redirect# + Submit# + Build# +Include#
TotalLanguages#	Total number of programming/scripting languages used to implement the Web Application

Firstly, a set of simple metrics is defined. These metrics characterize a Web Application either at system and component level, and then these metrics are used to evaluate the attributes. Table 13.1 and Table 13.2 report these metrics.

The metrics in the Table 13.1 aim to provide a structural size of the whole Web Application by counting the total number of components it consists of. In the following some remarks about these metrics are provided. When computing TotalWebPage# the Built Client Pages are not considered because there is not a physical source file corresponding to them; the reason is because a maintainer will operate on the source generating a built page. A problem exists about what has to be considered a LOC in Web Application: is it correct to consider as a LOC the text data that is displayed by a client page? In this work these lines of text have been counted too, because the idea is that they contribute to make more complex the execution of a maintenance operation.

Table 13.2 Web Application Metrics at Component Level

Metric Name	Description
WebPageTag#	Number of tags in the page
WebPageScript#	Number of scripts in the page
PageWeb Object#	Number of web objects in the page
WebPageI/O#	Number of form fields in the page + number of I/O data from/to mass storage devices
WebPageRelationships#	Number of relationships that page has with the other pages WebPageRelationships#= PageLink# + PageRedirect# + PageSubmit# + PageBuild#+PageInclude#
PageCodeSize	Number of source LOCs forming the page
PageInterface#	Number of Interface Objects referred in the page
WebObjectSize	Number of source LOCs forming the web object Note: This metric is used just only for those web objects whose source code is available
WebPageData#	The number of data different data identifiers in the Page
WebPageDataCoupling#	The number of data exchanged with other Web Pages
InnerComponents#	The number of inner components composing the page: InnerComponents# = PageForms# + PageWebObjects# + PageScripts# + PageFrames#
WebPageComplexity#	The cyclomatic complexity of the page
WebPageControlStructures#	The number of Control Flow Structures
PageLanguage#	Number of programming/scripting languages used to implement the page
ScriptSize	Number of source LOCs forming the Script

Moreover the web page count, the total number of scripts blocks have been considered (both server and client side ones), web objects, and interface object because their number provides an index of the Web Application general structural complexity (e.g. a greater number of scripts would mean a greater algorithmic complexity). The TotalData# and the InputField# would provide the Web Application complexity due to the data involved in the Web Application and to the user interactions. The TotalConnectivity# would indicate the overall Web Application complexity due to the control coupling among the Web Application pages. Finally the TotalLanguages# give us an index of the Web Application complexity due to the usage of several different programming/scripting languages.

The metrics in Table 13.2 aim to provide information about the structural complexity of a web page both by its internal composition and when connected to other pages. In particular the WebPageDataCoupling# and WebPageRelationships# metrics can provide an index of ripple effects among the pages.

In the OHMM the main system basic unit is the module or program, that is mainly characterized by its size in KLOC, the data it refers to, the number of control flow structures used to implement it, the control and data coupling it has with the other modules or programs.

In a Web Application the basic unit is the Web Page that is mainly characterized by its inner components (forms, scripts, web objects, and so on - these components are referred as page sub-components in the following), its size in LOC, the tags and the control flow structures used to implement it, the data it refers, the connections it has with other pages.

The common elements, the differences and the analogies existing between traditional systems and Web Applications are used to adapt the OHMM to Web Applications. The present work has been focused just on the Source Code Characteristics and in particular on the Control and Information sub-characteristics.

The Tables 13.3 to 13.6 reports the considered attributes and the metrics needed to evaluate them, for the Web Application Maintainability Model (WAMM).

In the Tables the new attributes have been highlighted writing (NEW) after the attribute name, while the word 'SAME' in the description field means that the same definition of the OHMM is used for the WAMM.

While adapting the OHMM to WAMM, 12 new attributes have been introduced, 2 ones have been deleted (i.e. Encapsulation and Span of Control Structures) and all the others have been adapted to the Web Applications by defining them using the simple basic Web Application metrics in Tables 13.1 and 13.2.

Table 13.3 Web Application Control Structure

Maintainability Attributes and Metrics at System Level

System Attribute	Web Application System Metrics
Size (NEW)	[TotalWebPage#, TotalLOC#, WebObject#, ServerScripts#, ClientScripts#, TotalLanguages#]
Modularity	[TotalWebPage#, Average PageCodeSize]
Complexity	Total Cyclomatic Complexity
Consistency	[Standard deviation of PageCodeSize, Standard deviation of WebPageComplexity#
Nesting	[Max number of InnerComponents#, Max depth of sub-components nesting, Average of nested sub-components per page, % of nested components]
Control Coupling	Total WebPageRelationships# / TotalWebPage#
Data Coupling (NEW)	Total WebPageDataCoupling# / TotalWebPage#
Module Reuse	Include# / TotalWebPage#
Control Flow Consistency	Total number of dead relationships interconnecting Web pages

Table 13.4 Web Application Control Structure

Maintainability Attributes and Metrics at Component Level

Component Attribute	Web Application Component Metrics
Size (NEW)	PageCodeSize
Modularity	[PageCodeSize, TotalWebPage#]
Complexity	Web Page Cyclomatic Complexity
Use of Structured Constructs	WebPageControlStructures#
Use of Unconditional Branching	SAME
Control Structure Nesting	SAME
Web Page InnerComponents (NEW)	InnerComponents#
PageLanguages	PageLanguage#
Density of Control Structures (NEW)	WebPageControlStructures# / PageCodeSize
Control Coupling (NEW)	WebPageRelationships#
Data Coupling (NEW)	WebPageDataCoupling#
Cohesion	SAME

Table 13.5 Web Application Information Structure

Maintainability Attributes and Metrics at System Level

System Attribute	Web Application System Metrics
Data Size (NEW)	[TotalData#, Total number of data references]
Global Data	Total number of global data/ TotalData#
Global Data Structures	Total number of global data structures/ TotalData#
System Coupling	Total WebPageDataCoupling#/ TotalData#
Data Flow Consistency	Total number of anomalous data usage/ TotalData#
Data Type Consistency	SAME
Nesting	SAME
I/O Data (NEW)	I/OField#
Density of Data (NEW)	TotalData# / TotalWebPage#
I/O Complexity	SAME
I/O Integrity	SAME

Table 13.6 Web Application Information Structure

Maintainability Attribute and Metrics at Component Level

System Attribute	Web Application System Metrics
Web Page Data Size (NEW)	[WebPageData#, Total number of data references in the Web Page]
Local Data Structures	Total number of data structures in a Web Page
Data Coupling	WebPageDataCoupling#
I/O Data (NEW)	WebPageI/O#
Initialization Integrity	SAME
Span of Data	WebPageData# / PageCodeSize

The Maintainability of a Web Application, with reference to the Source Code Control and Information Structure characteristics, may be expressed as a function of the 39 attributes described in tables 13.3 to 13.6:

$$\text{Web Application Maintainability} = F(\gamma_i A_i), \text{ i=1 .. 39}$$

where A_i is the value of i-th maintainability attribute and γ_i is the weight to assign to that attribute according to how much the attribute affects the maintainability. The definition of the values of such weights requires the availability and the analysis of (historical) data from several maintenance operations on different Web Applications. At the current state of the work there is not yet a meaningful amount of such data to be able to define the γ_i values, this task will be a future work for this research. However an initial analysis have been conducted for some Web Applications from real world. For these applications the attributes of WAMM have been computed and some simple experiments for a first validation of the effectiveness of the model have been carried out. The initial results are presented and discussed in the next section.

13.4 Case Study

Four web applications with different features, and implemented using ASP, Javascript, and HTML technologies, were selected for the case study. From here on, for sake of brevity, these applications are named WA1, WA2, WA3, and WA4. Some of these applications have been also considered in the case studies reported in the previous chapters.

WA1 is a source freeware application implementing a flexible discussion forum. WA2 is a freeware application supporting the realization of a customisable portal. WA3 is an application managing the on-line services provision for an undergraduate course. WA4 is a prototype of an e-commerce application supporting the buying of mp3 files.

The metrics described in the previous tables have been computed and used for providing a qualitative analysis of the Web Applications maintainability.

The most part of the metrics has been measured automatically by using the tool WARE and another tool developed just to support the computation of some Web Application metrics. The tools statically analyse the Web Application source code in order to compute the metrics. The remaining part of the metrics was achieved in semi-automatic way based on the results provided by the two tools: the cost for this kind of measuring was low, in terms of effort required and experience needed for the measurer. Such analysis is mainly localized on the single lines rather than on the overall structure of the application.

For sake of brevity in Tables 13.7 and 13.8 just some of the computed metrics are reported. This is the reason why Component Level Metrics are neglected. However, notice that System Level Metrics represent a synthetic expression of Component Level Metrics.

In this first case study then attention has been focused on some attributes, and related metrics, that are supposed to affect the maintainability harder than other ones. In particular, the Size, Complexity, Control and Data Coupling attributes have been considered.

In the following some remarks resulting by the analysis of the data in Tables 13.7 and 13.8 are reported.

When considering the System Size of a Web Application, all the items in 6-ple in Table 13.3 have to be taken into account. Indeed, the TotalWebPage# may not be significant if considered as a whole and by alone, but it is more significant when specifying its composition in term of the server and client pages, because these values provide an index on which side (server or client) the most of the Web Application complexity is. Moreover it is more and more significant when considered together other metrics. For example, let's notice that WA3 and WA1, from Table 13.7 have a similar TotalWebPage# then, from this perspective the two applications seem to be very similar. By using also the TotalLOC# metrics for those Web Applications, the TotalLOC# Ratio is 1:2.7 and the TotalServerPageLOC# ratio is 1:3.2. From this perspective, WA1 is much greater than WA3 and, moreover, harder to maintain, due to larger number of TotalPageLOC# and in particular for the number of TotalServerPageLOC#.

Then it is possible to suppose that the maintenance's effort is greater on the server side, where the most of the business logic is usually implemented, than on the client side. In this perspective, metrics concerning server side parts of Web Applications are good candidates for predictors of effort for maintenance interventions.

The Complexity attribute evaluation provides an indicator about the effort of maintenance intervention with specific regard to the business logic. From Table 13.7 it appears that WA1 is more complex than WA2 according to Total Cyclomatic Complexity metric with a ratio of 4:3, notwithstanding that the size of WA1 and Wa2 is almost the same in terms of TotalLOC#. Consequently, as well as in traditional software, in web applications size and complexity are not strictly related, but they have considered together to obtain useful information about the Web Application maintainability.

A high Coupling may negatively affect the impact of maintenance interventions [Bri99].

In a Web Application the kind of connections and thus the kind of control coupling, entails different consideration on the effort of maintenance intervention to implement. From Table 13.7 it seems that WA1 gets much more connections than WA2, WA3 and WA4, but about the 80% of WA1's connections are hypertextual links, which rarely bring data and business logic: usually, they are just used for navigation purposes. On the contrary, WA2 has a greater number of submit connections, that may be more complex to be maintained than the previous ones, because usually they are used to implement a user function and involve the exchange of data from a client page to the server page that will elaborate that data.

As well as in traditional software, variables amount and reference frequency are further indicators of the complexity of the information structure of a web application and its components. From this perspective WA1 is the most complex one.

Table 13.7 Web Application Control Structure Maintainability Measures at System Level

		WA1	WA2	WA3	WA4
Size	TotalServerPage#	71	126	75	22
	TotalClientPage#	0	0	23	19
	TotalWebPage#	71	126	98	41
	TotalBuiltClientPages#	56	53	74	21
	TotalLOC#	21994	22062	8025	2689
	TotalServerPageLoc#	21994	22062	6895	2098
	TotalStaticClientPageLoc#	0	0	1040	600
	WebObject#	0	0	0	0
	TotalLanguages#	3	3	3	3
Modularity	Average Page CodeSize	309,8	175,1	81,9	65,6
Complexity	Total CYclomatic Complexity	2936	2206	532	143
Consistency	Standard deviation of PageCodeSize	466,8	226,7	29,6	41,1
	Standard deviation of WebPageComplexity	49,5	22,4	1,5	3,8
Control Coupling	Total WebPage Relationship# / TotalWebPage#	17,6	5,4	1,1	3,6
	Link#	810	182	51	93
	Submit#	60	86	49	20
	Redirect#	117	17	8	32
Data Coupling	Total WebPage DataCoupling# / TotalWebPage#	11,3	6,1	1,4	1,4
Module Reuse	Include# / TotalWebPage#	3,7	5,4	0,6	0,4

Table 13.8 Web Application Information Structure Maintanability Measures at System Level

		WA1	WA2	WA3	WA4
Data Size	TotalData#	1147	1088	613	146
	TotalDataReferences#	17102	11059	3153	788
System Coupling	Total WebPageDataCoupling# / TotalData#	0,7	0,7	0,2	0,4
Density of Data	TotalData# / TotalWebPage#	16,2	8,6	6,3	3,6

By summarizing WA1 results the most complex application in terms of Size and Data while considering the Control Coupling WA1 is the most complex one for the number of connections, while WA2 is the most complex one for the kind of connections.

Further details about the maintainability model presented in this chapter can be found in [Dil04c].

13.5 Future Works

Empirical studies are needed to evaluate if the model is effective to assess the degree of maintainability of a Web Application. More specifically, a relevant goal is the evaluation of applying WAMM on data about a Web Application's story.

Metrics' effectiveness in predicting effort is strictly dependent on the kind of intervention to perform (i.e., intervention on user interface mainly regard client side pages). Referring to Web Application Maintainability's formula in section 13.2.3, the weight γ_i could be related to the influence of the attribute A_i on the kind of intervention. Although, such considerations are preliminary and need a further empirical validation, the WAMM's set of metrics seems to furnish a good quantitative support for building a theory in this direction.

PART 4: CONCLUSIONS AND FUTURE WORKS

Chapter 14: Conclusions

A Reverse Engineering approach has been described in this Ph.D. Thesis. The purpose of the approach is the recovering of knowledge from the analysis of the source code of Web Applications, supporting their maintenance and evolution.

A method to extract detailed information about the structure of the components of a Web Application analysing the source code has been proposed and described in this Thesis. The extracted information has been represented as a UML Class diagram, according to the model presented in Chapter 3. The tool WARE, described in chapter 6, allows the browsing of the recovered information in a very easy and intuitive way and provides a useful support to the process of the comprehension of structure and behaviour of the Web Application components. So, it provides a great help for the Engineer that has to carry out any maintenance intervention on the Web Application.

The problem of recovering high-level abstractions has been addressed from several points of view. A clustering technique to group the Web pages in subsets highly cohesive and loosely coupled has been defined to identify the groups of Web pages implementing the provided user functions. Moreover, a method has been defined to assign automatically a concept to a Web page or a group of pages. This method can be also useful for developing a search engine or for migrating to the Semantic Web.

Another problem that has been addressed is the automatic identification of Interaction Design Patterns analysing the source code of Web pages. The identification of these patterns provides further information to better and easier comprehend the functionality realized by Web pages. Moreover, a technique has been proposed to recognize Web pages and components having identical or similar structures. Applying this technique to the set of pages and components of a Web Application, cloned components may be recognized. The presence of cloned components can be due to a poor design of the software. So, a reengineering of the cloned components would improve the quality of the application. Moreover, it is possible to detect plagiarism (comparing pages and components of different Web Applications) and to compare different version of the same Web Application. Another application of this technique consists of comparing the different client pages that are built from the same server page, with the aim to recognize if the outputs of different executions of a server page are the same.

Some other methods and techniques have been defined to abstract business level UML use case diagrams, class diagrams and sequence diagrams. These diagrams show a set of views that are independent from the technological and architectural constraints due to the coding of the Web Application with scripting languages.

Another method has been proposed, facing the problem of the assessment of the vulnerability of the Web Application components to the Cross Site Scripting attacks. This method allows verifying if the components of the Web Application are intrinsically vulnerable to this kind of attacks.

Finally, a maintainability model for Web Applications has been defined. The model includes metrics that can be evaluated from the information extracted by the proposed Reverse Engineering approach. A tool has been developed that automatically calculates these metrics. These measures can represent a useful factor for decision-making processes.

14.1 Future Works

Future works are needed to better integrate the tools that have been realized and to better integrate the produced results. Some of the future works have already been discussed in the previous chapters. So, the focus of this subsection is limited to the future applications of the proposed techniques to face other Web Engineering problems.

In this Thesis, the automatic extraction of information from the source code of Web Applications is based on static analysis techniques. As a future work, the problem of the automatic extraction of information from the analysis of the execution of a Web Application will be addressed. Tools have to be developed to provide the needed automatic instrumentation of the source code of Web pages (the information extracted with the static analysis can be a useful support to this task) and to recover information from the execution of the instrumented Web Applications.

Future works will be addressed to the definition of methods and techniques to support the migration or the integration of Web Applications to Web Services. Also in this case high-level abstractions, recovered with the methods proposed in this Thesis, are a good starting point to define transformation and migration processes.

Other future works will be devoted to the testing of Web Applications. The Reverse Engineering approaches and tools presented in this Thesis have already been used to support the testing of Web Applications [Dil02b]. Tool WARE supports the identification of the components to be tested, while the other methods support the identification of the functionalities to test. Nowadays, some testing tools are under developing, supporting the automatic testing of Web Application.

As regards the quality assessment, the methods and the tools presented in this Thesis will be extended to assess the aging of a Web Application and to assess the efficacy of reengineering maintenance intervention, as regards improving the maintainability of the applications.

Finally, the problem of assessing the accessibility of a Web Application will be addressed too, by defining methods and techniques based on the extracted information recovered by the proposed Reverse Engineering approaches.

References

[Anq98] N. Anquetil and T. Lethbridge, "*Extracting concepts from file names; a new file clustering criterion*", *Proc. of 20th International Conference on Software Engineering,* IEEE CS Press, Los Alamitos, CA, 1998, pp. 84-93.

[Anq99] N. Anquetil, T.C. Lethbridge, "*Experiments with clustering as a software remodularization method".* In *Proceedings of 6th Working Conference on Reverse Engineering* - 1999. IEEE Computer Society Press: Los Alamitos, CA, 1999; 235-255.

[Ase02] A. Asencio, S. Cardman, D. Harris, E. Laderman "*Relating Expectations to Automatically Recovered Design Patterns*", Proceedings of the Ninth IEEE Working Conference on Reverse Engineering, 2002.

[Bal03] Z. Balanyi and R. Ferenc, "*Mining Design Patterns from C++ Source Code*", Proceedings of the IEEE International Conference on Software Maintenance, 2003

[Bal99] M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, K. Kontogiannis, "*Measuring clone based reengineering opportunities*", *Proc. International Symposium on software metrics. METRICS'99*. IEEE Computer Society Press, Nov 1999.

[Bal00] M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, K. Kontogiannis, "*Advanced clone-analysis to support object-oriented system refactoring"*, *Proc*. *Seventh Working Conference on Reverse Engineering*, IEEE Comp. Society Press, 2000, pp. 98-107.

[Bal01] F. Balmas, "*Displaying dependence graphs: a hierarchical approach*", *Proc. of 8th Working Conference on Reverse Engineering,* IEEE CS Press, Los Alamitos, CA, 2001, pp. 261- 270.

[Bak93] B. S. Baker., "*A theory of parametrized pattern matching: algorithms and applications"*, *Proc. 25th Annual ACM Symposium on Theory of Computing*, pp. 71-80, May 1993.

[Bak95] B. S. Baker, "*On finding duplication and near duplication in large software systems"*, *Proc. 2nd Working Conference on Reverse Engineering*, IEEE Computer Society Press, 1995.

[Bak95b] B. S. Baker, "*Parametrized pattern matching via Boyer-Moore algorithms"*, *Proc. Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 541-550, Jan 1995.

[Bas85] V. Basili, D. Hutchens, "*System structure analysis: clustering with data bindings*", *IEEE Transactions on Software Engineering*, 11 (8), 1985, pp. 749-757.

[Bax98] Baxter I. D., Yahin A., Moura L., Sant'Anna M., Bier L., "*Clone Detection Using Abstract Syntax Trees"*, *Proc. International Conference on Software Maintenance*, 368-377, IEEE Computer Society Press, 1998.

[Ban00] A. Bangio, S. Ceri, P. Fraternali, "*Web Modeling Language (WebML): a modeling language for designing Web sites*", in *Proceedings of the 9th International Conference on the WWW (WWW9)* - 2000. Elsevier: Amsterdam, Holland, 2000: 137-157.

[Ben89] P. Benedusi, A. Cimitile, U. De Carlini, "*A reverse engineering methodology to reconstruct hierarchical data flow diagrams for software maintenance*". In *Proceedings of Conference on Software Maintenance* - 1989. IEEE Computer Society Press: Los Alamitos, CA, 1989: 180-189.

[Ben92] P. Benedusi, A. Cimitile, U. De Carlini, "*Reverse engineering process, document production and structure charts*". *Journal of Systems and Software* 1992; 19: 225- 245.

[Ben02] M. Benedikt, J. Freire, P. Godefroid, "*VeriWeb: Automatically Testing Dynamic Web Sites*", Proceedings of the eleventh international conference on World Wide Web, ACM Press New York, NY, USA, 2002, pp.396-407

[Ber84] H.L. Berghel, D.L. Sallach, "*Measurements of program similarity in identical task environments*", *SIGPLAN Notices*, 9(8):65-76, Aug 1984.

[Big93] T.J: Biggerstaff, B.G. Mitbander, D. Webster, "*Program understanding and the concept assignment problem.*" *Communications of the ACM* 1993; **37** (5): 72- 83.

[Boe00] B.W. Boehm et al., "*Software Cost Estimation with Cocomo II*", Prentice-Hall, Upper Saddle River, N.J., 2000

[Bol01] C. Boldyreff, R. Kewish, "*Reverse Engineering to Achieve Maintainable WWW Sites*". In *Proceedings of Eighth Working Conference on Reverse Engineering* 2001. IEEE Computer Society Press: Los Alamitos, CA, 2001: 249-257.

[Bor01] J. Borchers, "A *Pattern Approach to Interaction Design*", John Wiley & Sons, Chichester, West Sussex, UK, 2001.

[Bre98] P. Brereton, D. Budgen, G. Hamilton, "*Hypertext: the next maintenance mountain*", IEEE Computer , December 1998, IEEE Computer Society Press, Los Alamitos, CA.

[BugT1] *Security Focus Bugtraq*, http://www.securityfocus.com/

[BugT2] *Securepoint BugTraq*, http://msgs.securepoint.com/bugtraq/

[Bri99] L.C.Briand, J.Wust, H.Lounis, "*Using coupling measurement for impact analysis in object-oriented systems*", Proc. of IEEE International Conference on Software Maintenance, 1999. IEEE Computer Society Press, Los Alamitos, pp. 475-482.

[Can96] G. Canfora, A. Cimitile, and M. Munro, "*An improved algorithm for identifying reusable objects in code*", *Software Practice and Experiences*, vol. 26, no. 1, 1996, pp. 24-48.

[Cert00] "*CERT® Advisory CA-2000-02 Malicious HTML Tags Embedded in Client Web Requests*", http://www.cert.org/advisories/CA-2000-02.html

[cgi] "*The Cross Site Scripting FAQ*", http://www.cgisecurity.com/articles/xss-faq.txt

[Chu01] S. Chung, Y.S. Lee, "*Reverse software engineering with UML for web site maintenance*", in *Proceedings of 1st International Conference on Web Information Systems Engineering* 2001, IEEE Computer Society Press: Los Alamitos, CA, 2001, (2): 157-161.

[Cim95] A.Cimitile and G.Visaggio. "*Software Salvaging and the Call Dominance Tree*", *The Journal of Systems and Software,* Volume 28, Number 2. February 1995.

[Cim99] A. Cimitile, A. De Lucia, G.A. Di Lucca, and A.R. Fasolino, "*Identifying objects in legacy systems using design metrics*", *The Journal of Systems and Software*, vol. 44, January 1999, pp. 199-211

[Com01] S. Comai, P. Fraternali, "*A semantic model for specifying data-intensive Web applications using WebML*", in *Proceedings of the International Semantic Web Working Symposium* - 2001. http://www.semanticweb.org/SWWS/program/full/paper19.pdf [10 April 2003]

[Con99] J. Conallen, "*Building Web Applications with UML*". Addison Wesley Publishing Company: Reading, MA, 1999.

[Con99b] J. Conallen, "*Modeling web application architectures with UML*". *Communications of the Association for Computing Machinery* 1999. 42 (10): 63-70.

[DaC03] D. Da Costa, C. Dahn, S. Mancoridis, V. Prevelakis, "*Characterizing the 'security vulnerability likelihood' of software functions*", Proceedings of International Conference on Software Maintenance, ICSM 2003, IEEE, CS Press, Los Alamitos, CA, 2003, pp.266-274

[Dat03] Datamonitor, "*Enterprise Security Product Market*", Datamonitor report DMTC0913, Jul 2003.

[Del97] A. De Lucia, G.A. Di Lucca, A.R. Fasolino, et al., 'Migrating legacy systems towards object-oriented platforms', *Proc. of IEEE Int. Conference on Software Maintenance*, ICSM 1997, IEEE CS Press, pp. 122- 129.

[Dil00] G. A. Di Lucca, A.R. Fasolino, U. De Carlini, "*Recovering Class Diagrams from Data-Intensive Legacy Systems*", *Proc. of IEEE Int. Conference on Software Maintenance, ICSM 2000,* San Jose (USA), Oct. 2000, IEEE C. S. Press, pp. 52- 63.

[Dil01] G. A. Di Lucca, M. Di Penta, A.R Fasolino, P. Granato, "*Clone Analysis in the Web Era: an approach to identify Cloned Web Pages*", *Proc. of WESS 2001,* Firenze, Italy.

[Dil02] G.A. Di Lucca, A.R. Fasolino, U. De Carlini, F. Pace, P. Tramontana, "*WARE: a tool for the Reverse Engineering of Web Applications*", in *Proceedings of 6th European Conference on Software Maintenance and Reengineering* - 2002, IEEE Computer Society Press, Los Alamitos, CA, 2002: 241-250.

[Dil02b] G.A. Di Lucca, A.R. Fasolino, F. Faralli, U. De Carlini, "*Testing Web Applications*", in *Proceedings of International Conference on Software Maintenance* - 2002. IEEE Computer Society Press, Los Alamitos, CA, 2002: 310-319.

[Dil02c] G. A. Di Lucca, A.R. Fasolino, U. De Carlini, F. Pace, P. Tramontana, "*Comprehending Web Applications by a Clustering Based Approach*", Proc. of *10th IEEE Workshop on Program Comprehension*, *IWPC 2002, Pages:261 - 270*

[Dil02d] G. A. Di Lucca, A.R. Fasolino, P. Tramontana, " *Towards a Better Comprehensibility of Web Applications: Lessons Learned from Reverse Engineering Experiments*", Proc. of 4*th* *IEEE Workshop on Web Site Evolution*, WSE *2002, Pages:33 - 42*

[Dil03] G. A. Di Lucca, A.R. Fasolino, U. De Carlini, P. Tramontana, "*Abstracting business level UML diagrams from web applications*", Proc. of 5th *IEEE Workshop on Web Site Evolution*, WSE *2003,* pp.12-19

[Dil03b] G.A. Di Lucca, A.R.Fasolino, U.De Carlini, P.Tramontana, "*Recovering a Business Object Model from Web Applications*", Proceedings of the *27th IEEE Annual International Computer Software and Applications Conference*, COMPSAC 2003, Pages: 348 - 353

[Dil04] G.A. Di Lucca, A.R. Fasolino, P. Tramontana, "*Reverse Engineering Web Application: the WARE approach*", Journal of Software Maintenance and Evolution: Research and Practice, Volume 16, Issue 1-2, Date: January - April 2004, Pages: 71-101

[Dil04b] G.A. Di Lucca, A.R. Fasolino, P. Tramontana, U. De Carlini, "*Identifying Reusable Components in Web Applications*", IASTED International Conference on Software Engineering, SE 2004, pp.526-531

[Dil04c] G.A. Di Lucca, A.R.Fasolino, P.Tramontana, C.A.Visaggio, "*Towards the definition of a maintainability model for web applications*", Proceedings of the Eighth IEEE European Conference on Software Maintenance and Reengineering, CSMR 2004, pages:279 - 287

[Dil04d] G.A. Di Lucca, A.R.Fasolino, M.Mastroianni, P.Tramontana, "*Identifying Cross Site Scripting Vulnerabilities in Web Applications*", Proceedings of 6th IEEE Workshop on Web Site Evolution, WSE 2004, pages 91-100

[Dil04e] G.A. Di Lucca, A.R.Fasolino, P.Tramontana, U.De Carlini, "*Supporting Concept Assignment in the Comprehension of Web Applications"*, Proceedings of the 28th IEEE Annual International Computer Software and Applications Conference, COMPSAC 2004

[Dil04f] G.A. Di Lucca, A.R.Fasolino, P.Tramontana, U.De Carlini, "*WARE: a tool for Web Applications Reverse Engineering*", Tool Demonstrations session of the 8th IEEE European Conference on Software Maintenance and Reeengineering, CSMR 2004, pp.24-28

[Dil04g] G.A. Di Lucca, A.R.Fasolino, P.Tramontana, U.De Carlini, "*Reverse Engineering Web Applications using the WARE tool*", chapter of the book "*Tools for software maintenance and reengineering*", Franco Angeli editore (to be published)

[Dil05] G.A.Di Lucca, A.R. Fasolino, P. Tramontana, "*Recovering Interaction Design Patterns in Web Applications*", accepted for 9th IEEE European Conference on Software maintenance and Evolution, CSMR 2005

[Dot] *Dotty.* Available at the URL: http://www.research.att.com/sw/tools/graphviz

[Duy02] D.K. van Duyne, J.A. Landay, J. Hong, "*The design of sites"*, Addison-Wesley, Boston, US, 2002

[Fer02] M. Ferrara, "*Metriche Software per il Riconoscimento di Cloni in Applicazioni Web e una Metodologia per la loro Reingegnerizzazione"*, Laurea Degree Thesis, Dip. Informatica e Sistemistica University of Naples Federico II, 2002.

[Gal95] H.Gall, R.Klösch, "*Finding objects in procedural programs: an alternative approach*", Proc. of 2nd Working Conference on Reverse Engineering, Toronto, Canada, 1995, IEEE CS Press, pp. 208-216

[Gam95] E. Gamma, R. Helm, R. Johnson, J. Vlissides, "*Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Boston, Massachusetts, USA, 1995

[Geo96] J. George and B.D. Carter, "*A strategy for mapping from function oriented software models to object oriented software models*", ACM Software Engineering Notes, vol. 21, no. 2, March 1996, pp. 56-63

[Gin01] A.Ginige and S.Murugesan, "*Web engineering. An introduction*", IEEE Multimedia, **8**(1):14-18, April-June 2001

[Gol01] N. Gold, "*Hypothesis-based concept assignment to support software maintenance*", *IEEE International Conference on Software Maintenance*, IEEE, CS Press, Los Alamitos, CA, 2001, pp.545-548

[Gra03] I. Graham, "A *pattern language for Web Usability"*, Addison-Wesley, Boston, US, 2003

[Gri81] S. Grier, "*A tool that detects plagiarism in PASCAL programs"*, *SIGSCE Bulletin*, 13(1), 1981.

[Har92] D. Harman, "*Information retrieval: Data Structures and Algorithms*", Prentice-Hall, EngleWood Cliffs, NJ, 1992, pp.363-392

[Has01] A.E. Hassan, R.C. Holt, "*Towards a better understanding of web applications"*, in *Proceedings of 3rd International Workshop on Web Site Evolution* - 2001, IEEE Computer Society Press, Los Alamitos, CA, 2001: 112-116.

[Hcip] *HCI Pattern Index*, available at http://www.hcipatterns.org/patterns/borchers/patternindex.html

[Hec77] M. S. Hecht, "*Flow Analysis of Computer Programs*", *Elsevier North-Holland, 1977.*

[Hill] *Pattern Catalog*, available at http://hillside.net/ patterns/ onlinepatterncatalog.htm

[Html] *HTML 4.01 Specification*, W3C Recommendation 24 December 1999, http://www.w3.org/TR/html4/

[Hon00] A. Honrado, R. Leon, R. O'Donnel, D. Sinclair, "*A word stemming algorithm for the Spanish language*", *7th International Symposium on String Processing and Information Retrieval*, IEEE, CS Press, Los Alamitos, CA, 2000, pp. 139-145

[Hor90] S. Horwitz, "*Identifying the semantics and textual differences between two versions of a program"*, *Proc. ACM SIGPLAN Conference on Programming Language Design and Implementation*, 234-245, June 1990.

[Hua03] Yao-Wen Huang, Shih-Kun Huang, Tsung-Po Lin, Chung-Hung Tsai, "*Web Application Security Assessment by Fault Injection and Behaviour Monitoring*", Proceedings of the twelfth international conference on World Wide Web, ACM Press, New York, NY, USA, 2003, pp.148-159

[Idef03] David Endler, "*The evolution of Cross-Site Scripting Attack*", http://pgsit.org/pages/2003/gzuchlinski/libox/websecdocs/XSS.pdf

[Isa97] T. Isakowitz, A. Kamis, M. Koufaris, "*Extending the capabilities of RMM: Russian dolls and hypertext", i*n *Proceedings of 30th Hawaii International Conference on System Science, Maui, HI,* 1997; (6): 177-186.

[Itsd] The ITsecurity.com Dictionary of Information Security, "*Cross Site Scripting (XSS, cross-site malicious content)*", http://www.itsecurity.com/dictionary/xss.htm

[Jan88] H.T. Jankowitz , "*Detecting plagiarism in student PASCAL programs"*, in *Computer Journal*, 31(1):1-8, 1988.

[Kir02] Kirchner M. "*Evaluation, Repair, and Transformation of Web Pages for Web Content Accessibility. Review of some available Tools*". In *Proceedings of 4th International Workshop on Web Site Evolution* - 2002, IEEE Computer Society Press, Los Alamitos, CA, 2002: 65-72.

[Kon95] K. Kontogiannis, De Mori R., Bernstein M., Merlo E., "*Pattern Matching for Design Concept Localization"*, *Proc. 2nd Working Conference on Reverse Engineering*, IEEE Computer Society Press, 1995.

[Kon96] K. Kontogiannis, R. De Mori, E. Merlo, M. Galler, M. Bernstein, "*Pattern Matching for clone and concept detection"*, *Journal of Automated Software Engineering*, 3:77-108, Mar 1996.

[Kon97] K. Kontogiannis, "*Evaluation Experiments on the Detection of Programming Patterns Using Software Metrics"*, *Proc. 4th Working Conference on Reverse Engineering,* 44-54, 1997.

[Lag97] B. Lague, D. Proulx, J. Mayrand, E. Merlo, J. Hudepohl, "*Assessing the Benefits of Incorporating Function Clone Detection in a Development Process"*, *Proc. International Conference on Software Maintenance*, IEEE Comp. Society Press, 1997, pp. 314- 321.

[Lev66] V. I. Levenshtein, "*Binary codes capable of correcting deletions, insertions, and reversals"*, *Cybernetics and Control Theory* 10 (1966), 707-710.

[Liu90] S. Liu and N. Wilde, "*Identifying objects in a conventional procedural language: an example of data design recovery*", Proc. of Conference on Software Maintenance, San Diego, CA, 1990, IEEE CS Press, pp. 266-271

[Liv94] P.E. Livadas and T. Johnson,"*A new approach to finding objects in programs*", J. of Software Maintenance: Research and Practice, vol. 6, 1994, pp. 249-260

[Mai04] Alec Main, "*Application Security: Building in Security during the Development Stage*", Information System Security, May/June 2004, pp. 31-54.

[Man98] S. Mancoridis, B.S. Mitchell, C. Rorres, Y. Chen and E.R. Gansner, "*Using automatic clustering to produce high-level system organizations of source code*", *6th International Workshop on Program Comprehension,* IEEE CS Press, Los Alamitos, CA*, 1998.*

[Man99] S. Mancoridis, B.S. Mitchell, Y. Chen and E.R. Gansner, "*Bunch: a clustering tool for the recovery and maintenance of software system structures*", *IEEE International Conference on Software Maintenance,* IEEE CS Press, Los Alamitos, CA, 1999*,* pp. 50- 59.

[Mar01] J. Martin, L. Martin, "*Web site maintenance with software engineering tools"*, in *Proceedings of 3rd International Workshop on Web Site Evolution* - 2001, IEEE Computer Society Press, Los Alamitos, CA, 2001; 126-131.

[May96] Mayrand J., Leblanc C., Merlo E., "*Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics"*, *Proceedings International Conference on Software Maintenance*, 244-253, IEEE Computer Society Press, 1996.

[Mic00] D. Ross, I. Brugiolo, J. Coates, M. Roe, "*Cross-site Scripting Overview*", http://www.microsoft.com/technet/security/news/csoverv.mspx

[Men01] E. Mendes, N. Mosley, S. Council, "*Web Metrics - Estimating Design and Authoring Effort"*, IEEE Multimedia, January-March 2001, IEEE Computer Society Press, Los Alamitos, CA.

[Mul88] H.A. Müller, K. Klashinsky, "*Rigi - A system for programming in the large*". In *Proceedings of International Conference on Software Engineering* - 1988, IEEE Computer Society Press, Los Alamitos, CA, 1988; 80-86.

[New95] P. Newcomb and G. Kotik, "*Reengineering procedural into object-oriented systems*", Proc. of 2nd Working Conference on Reverse Engineering, Toronto, Canada, 1995, IEEE CS Press, pp. 237-249

[Off02] J. Offutt J. "*Quality Attributes of Web Software Applications*". *IEEE Software* 2002; 19 (2): 25-32.

[Ohm02] K. Ohmaki, "*Open source software research activities in AIST towards secure open systems*", 7th IEEE International Symposium on High Assurance Systems Engineering, 2002. 23-25 Oct. 2002, pp. 37 -41

[Oma92] P. Oman, J. Hagemeister, "*Metrics fo Assessing a Software System's Maintainability"*, Proceedings of IEEE International Conference on Software Maintenance, 1992, IEEE Computer Society Press, Los Alamitos, CA.

[Ore01] V. M. Orengo, C. Huyck, "*A Stemming Algorithm for the Portuguese Language*", *8th International Symposium on String Processing and Information Retrieval*, IEEE, CS Press, Los Alamitos, CA, 2001, pp. 186-193

[Pag02] L. Paganelli, F. Paternò, "*Automatic Reconstruction of the Underlying Interaction Design of Web Applications"*. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE02)* – 2002. ACM Press: NY, 2002; 439-445,

[Pat99] J.F. Patenaude, E. Merlo, M. Dagenais, B. Lagüe, "*Extending software quality assessment techniques to java systems"*, *Proc. 7th International Workshop on Program Comprehension IWPC'99*, IEEE Computer Society Press, 1999.

[Por80] M. F. Porter, "*An algorithm for suffix stripping*", *Program*, 14(3), 1980 :pp. 130-137

[Rei00] D.J. Reiffer, "*Web Development: Estimating Quick-to-Market Software"*, Novenber/December 2000, IEEE Computer Society Press, Los Alamitos, CA.

[Ric00] F. Ricca, P. Tonella, "*Web site analysis: Structure and evolution*". In *Proceedings of International Conference on Software Maintenance* - 2000, IEEE Computer Society Press, Los Alamitos, CA, 2000; 76-86.

[Ric01] F. Ricca, P. Tonella "*Understanding and Restructuring Web Sites with ReWeb*". *IEEE Multimedia* 2001; 8(2): 40-51.

[Ric01b] F. Ricca, P. Tonella, "*Analysis and testing of web applications*", Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, IEEE, CS Press, Los Alamitos, CA, pp. 25 –34

[Ric04] F. Ricca, P. Tonella, C. Girardi, E. Pianta, "*An empirical study on keyword-based web site clustering*", Proceedings. of the 12th IEEE International Workshop on Program Comprehension, IWPC 2004, IEEE, CS Press, Los Alamitos, CA, pp.:204 - 213

[Ros99] G. Rossi, D. Schwabe, F. Lyardet, "*Web application models are more than conceptual models*", in *Proceedings of the First International Workshop on Conceptual Modeling and the WWW* 1999: 239-253.

[Sch91] R.W. Schwanke, "*An intelligent tool for Re-engineering Software Modularity*", *Proc. of 13th International Conference on Software Engineering*, IEEE CS Press, Los Alamitos, CA, 1991, pp. 83-92.

[Sch01] D. Schwabe, L. Esmeraldo, G. Rossi, F. Lyardet, "*Engineering web applications for reuse*". *IEEE Multimedia*, 2001; 8(1): 20–31.

[Sco02a] D. Scott, R. Sharp, "*Abstracting Application-Level Web Security*", Proceedings of the eleventh international conference on World Wide Web, ACM Press New York, NY, USA, 2002, pp.396-407

[Sco02b] D. Scott, R. Sharp, "*Developing secure Web applications*", Internet Computing, IEEE , Volume: 6 Issue: 6 , Nov.-Dec. 2002, pp. 38 -45

[Snf] Snitz Forum 2000, http://forum.snitz.com/

[Sto03] J. Landall, J. Stoltenberg, "*Application Security: Have We Locked the Windows and Left the Door Open?*", Information System Security, May/June 2003, pp. 37-43.

[Str02] Y. Strashnoy, "*The need for Web Application Security*", www.elitesecureweb.com/images/pdf/need_for_web_app_security.pdf

[Tid98] J. Tidwell, "*Interaction Design Patterns*", in Proceedings of the Pattern Languages of Programming, Monticello, Illinois, USA, August 11-14, 1998, available online at http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P29.pdf

[Til01] S. Tilley, S. Huang "*Evaluating the reverse engineering capabilities of web tools for understanding site content and structure: a case study*". In *Proceedings of 23rd International Conference on Software Engineering* - 2001*,* IEEE Computer Society Press, Los Alamitos, CA, 2001; 514- 523.

[Ton99] P. Tonella, G. Antoniol, "*Object oriented design pattern inference*", Proceedings of the IEEE International Conference on Software Maintenance, 1999, Pages:230 – 238

[Ton02] P. Tonella, F. Ricca, E. Pianta, C. Girardi, "*Restructuring Multilingual Web Sites*", in *Proceedings of International Conference on Software Maintenance* - 2002. IEEE Computer Society Press, Los Alamitos, CA, 2002; 290-299.

[Ton03] P. Tonella, F. Ricca, E. Pianta, C. Girardi, "*Using keyword extraction for web site clustering*", *5th International Workshop on Web Site Evolution,* IEEE CS Press, Los Alamitos, CA, 2003, pp. 41-48

[Ton03b] P.Tonella, F.Ricca, E.Pianta, C.Girardi, G. A. Di Lucca, A.R. Fasolino, P. Tramontana, "*Evaluation methods for web application clustering*", Proc. of 5th *IEEE Workshop on Web Site Evolution*, WSE *2003,* pp.33-40

[Tra01] P. Tramontana, "*Un analizzatore statico di codice a supporto del Reverse Engineering di Applicazioni Web*", Laurea Degree Thesis, University of Naples "Federico II", 2001

[Tze00] V. Tzerpos, R.C. Holt, "*On the stability of software clustering algorithms*", *8th International Workshop on Program Comprehension,* IEEE CS Press, Los Alamitos, CA, 2000, pp. 211-220.

[Tze00b] V. Tzerpos, R.C. Holt, "*ACDC: an algorithm for comprehension-driven clustering*", *7th Working Conference on Reverse Engineering,* IEEE CS Press, Los Alamitos, CA, 2000, pp. 258- 267.

[Van01] J. Vanderdonckt, L. Bouillon, N. Souchon, "*Flexible reverse engineering of web pages with VAQUISTA*". In *Proceedings of Eighth Working Conference on Reverse Engineering* - 2001. IEEE Computer Society Press, Los Alamitos, CA, 2001; 241 –248.

[Vcg] Lemke I., Sander G. VCG: A Visualization tool for Compiler Graphs. *The COMPARE consortium,* 1993.

[Whe02] D. A. Wheeler, "*Secure Programming for Linux and Unix HOWTO*", http://dwheeler.com/secure-programs/Secure-Programs-HOWTO.html

[Wig97] T.A. Wiggerts, "*Using clustering algorithms in legacy systems remodularization*", *4th Working Conference on Reverse Engineering,* IEEE CS Press, Los Alamitos, CA*,* 1997, pp. 33-43.

[Won94] K. Wong, S. Tilley, H.A. Müller, M.A. D. Storey, "*Programmable Reverse Engineering*", *International Journal of Software Engineering and Knowledge Engineering*, 4 (4), Dec. 1994, pp.501-520.

[Wel03] M. van Welie, G. C. van der Veer, "*Pattern Languages in Interaction Design: Structure and Organization*", Proceedings of Ninth International Conference on Human-Computer Interaction, Interact 2003, Zürich, Switzerland, pp. 527-534

[Wel04] "*Pattern in interactio*n design: Web Design Patterns" avalaible at http://www.welie.com/patterns

[Yeh95] A.S. Yeh, D.R. Harris, and H.B. Rubenstein, "*Recovering abstract data types and object instances from a conventional procedural language*", *Proc. of 2nd Working Conference on Reverse Engineering*, Toronto, Canada, 1995, IEEE CS Press, pp. 227-236