Research

Reverse engineering Web applications: the WARE approach

Giuseppe Antonio Di Lucca¹, Anna Rita Fasolino²,*,† and Porfirio Tramontana²

¹RCOST—Research Centre on Software Technology, Department of Engineering, University of Sannio, Via Traiano Palazzo ex Poste, 82100 Benevento, Italy
²Department of ‘Informatica e Sistemistica’, University of Naples ‘Federico II’, Via Claudio 21, 80125 Naples, Italy

SUMMARY

The rapid, progressive diffusion of Web applications in several productive contexts of our modern society is laying the foundations of a renewed scenario of software development, where one of the emerging problems is that of defining and validating cost-effective approaches for maintaining and evolving these software systems.

Due to several factors, the solution to this problem is not straightforward. The heterogeneous and dynamic nature of components making up a Web application, the lack of effective programming mechanisms for implementing basic software engineering principles in it, and undisciplined development processes induced by the high pressure of a very short time-to-market, make Web application maintenance a challenging problem. A relevant issue consists of reusing the methodological and technological experience in the sector of traditional software maintenance, and exploring the opportunity of using reverse engineering to support effective Web application maintenance.

This paper presents an approach for defining reverse engineering processes involving Web applications. The approach has been used to implement a process, including reverse engineering methods and a supporting software tool, that helps to understand existing undocumented Web applications to be maintained or evolved, through the reconstruction of UML diagrams. The proposed reverse engineering process has been submitted to a validation experiment, the results of which showed the usability of the process for reverse engineering Web applications with different characteristics, and highlighted possible areas for improvement of its effectiveness. The experiment and the lessons learned from it are presented in the paper. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: Web application; reverse engineering; program understanding; clustering; UML diagrams

*Correspondence to: Anna Rita Fasolino, Department of ‘Informatica e Sistemistica’, University of Naples ‘Federico II’, Via Claudio 21, 80125 Naples, Italy.
†E-mail: fasolino@unina.it

Copyright © 2004 John Wiley & Sons, Ltd.
1. INTRODUCTION

In the Internet era, the diffusion of Web applications in various productive contexts is growing apace, and the way business processes are carried out is altering 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 providing. 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 generated at runtime on the basis of the user input. The Web application components may reside on distinct computers according to a client–server or multi-tier model, and may be integrated by different mechanisms that generate various levels of coupling and flow of data between the components.

In order to satisfy a growing market request for Web applications and to cope with their increased technological complexity, the software community has advised of a need for specific methods and techniques able to support a disciplined and more effective development process. In the last few years, several methodologies for producing Web sites and Web applications have been proposed. Some proposals derive from the area of hypermedia applications, such as the Relationship Management Methodology (RMM) [1], and the Object-Oriented Hypermedia Design Method (OOHDM) [2,3], while others, such as the methodology by Gnaho and Larcher [4], have been developed specifically for Web applications. Further approaches for producing Web applications are illustrated in [5–7], while [8] proposes a development model, based on Web site usability, aiming to promote user satisfaction. The W3I3 project defined a methodology and a notation language, the Web Modelling Language (WebML), for specifying complex Web sites and applications at the conceptual level and along several dimensions [9,10]. Some extensions to the UML notation [11] have been proposed by Conallen [12,13] to make the UML language suitable for modelling Web applications.

However, the high pressure of a very short time-to-market often forces the developers to implement the code of the application 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 in a short time, using no disciplined development process, and without respecting software engineering principles. Poor quality and inadequate documentation must 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 and evolution. The software community is now seriously addressing the problem of defining and validating similar approaches for Web applications.

According to Chikofsky and Cross [14], reverse engineering is ‘the process of analysing a subject system to identify the system’s components and their interrelationships and create representations of the system in another form or at a higher level of abstraction’. The definition of a reverse
engineering approach always depends on a number of factors: a reverse engineering process can be run to accomplish different purposes, such as re-documenting an existing application, supporting its maintenance, testing or evolution, assessing its quality. Depending on the specific aims to be achieved, a set of different views of the reverse engineered application have to be identified, that can support achievement of the goal. Moreover, suitable methods and tools allowing the software views to be effectively obtained from the subject applications have to be defined and validated. Generally, a number of methodological and technological issues need to be solved before a reverse engineering process involving an existing application can be defined.

The problem of reverse engineering Web applications is addressed in this paper, which presents an approach called WARE (Web Application Reverse Engineering), defining and implementing a reverse engineering process involving Web applications and a supporting tool. The process has been designed using the Goals/Models/Tools (GMT) paradigm proposed in [9,15], that offers a structured method for defining reverse engineering processes. The process aims to support the comprehension of existing undocumented Web applications to be maintained, through the reconstruction of UML diagrams providing distinct views of the Web applications. The feasibility and effectiveness of the proposed process have been explored by a validation experiment. Results from the experiment and lessons learned from it are presented in this paper.

The remainder of the paper is organized as follows. Section 2 presents background information including a conceptual model depicting the main features of a Web application, and related works on analysing existing Web applications. Section 3 describes how the GMTs paradigm can be used to define and implement Web application reverse engineering processes, while Section 4 presents the WARE’s reverse engineering process, proposed to achieve the comprehension of existing undocumented Web applications. Section 5 illustrates the tool designed and implemented to support the proposed process. In Section 6 a validation experiment carried out to assess the feasibility and effectiveness of the process is described, while Section 7, finally, provides concluding remarks.

2. BACKGROUND

In this section, a conceptual model enabling the reader to focus on the relevant features of an existing Web application is preliminarily presented. In addition, the section provides an overview of the techniques and tools recently proposed in the literature for analysing Web sites and Web applications.

2.1. A conceptual model of a Web application

Unlike a Web site that simply provides its users with 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 [12].

The technology evolution observed in the last few years in the field of Web applications has had a strong impact on the structural complexity of this type of software. According to the classification proposed by Tilley and Huang [16], 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 characterized 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).

Focusing our attention on class 3 applications, which provide the most complex user functions with the support of dynamic technologies, a reference conceptual model describing the relevant entities of the Web application and the main relationships between these entities can be introduced, to represent the structure of this type of application.

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. The client pages 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 characterized 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). Forms are used to collect user input and to submit the input to the server page, which 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.

A 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 will be used to describe the inclusion of an entity in another entity.

Figure 1 shows a UML class diagram that will be assumed in the paper 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 conceptual model is based on the one proposed by Conallen [12,13,17]. Conallen’s proposal focused on forward engineering, and consisted of a model for building Web applications using the UML language, where some extensions of the UML are introduced.
Since in this paper we consider a Web application from a reverse engineering point of view, the UML model we propose 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, our 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 our 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.

Several other models for representing a Web application can be found in the literature. Most of them have been proposed for developing new Web applications and extend the traditional development approaches with specific models describing Web-related characteristics. Some examples are the OOH method, which integrates the traditional object-oriented models with a navigational view and a presentation view of the application [18], the RMM methodology [1], which proposes entity-relationship-based diagrams for designing Web sites, the development approach described in [10,19]
for specifying complex Web sites at the conceptual level using the WebML modelling language and, finally, the OOHDM methodology, which permits the construction of customized Web applications by adopting object-oriented primitives in order to build a conceptual model, a navigational model and an interface model of the Web application [20].

These models have not been taken into account in this paper for representing an existing Web application, since the information they represent does not reach the degree of granularity required for reverse engineering purposes.

2.2. Related work

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 [21] 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. [22] propose reusing the software engineering tool Rigi [23] as a means of analysing and visualizing the structure of Web applications.

Other techniques and tools have been defined ad hoc for managing existing Web applications. Chung and Lee [24] 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 [25,26] present the ReWeb tool to perform several traditional source code analyses of Web sites: they use a graphical representation of Web sites and introduce the idea of pattern recovery over this representation. The dominance and reachability relationships are used to analyse the graphs, in order to support the maintenance and evolution of the Web sites. Schwabe et al. [27] 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. [28] 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. [29] 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. Di Lucca et al. [30] present a method and a tool [31] for reverse engineering Web applications. The method defines the reverse engineering activities that are needed to obtain a set of views of a Web application at different abstraction levels. The views are cast into UML diagrams that can be obtained with the support of a reverse engineering tool.

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 [32]. Kirchner [33] 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. [34] 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. [35] describe the TERESA tool, which 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.

The techniques and tools presented in these papers 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 characteristics. 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. In the following section, a general approach will be presented, which provides a comprehensive framework for defining or selecting the analysis tools to be adopted in a reverse engineering process.

3. USING THE GOALS/MODELS/TOOLS PARADIGM TO DEFINE A REVERSE ENGINEERING PROCESS FOR WEB APPLICATIONS

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 GMT paradigm described in [9,15], a reverse engineering process is characterized 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. Possible goals, models, and tools characterizing Web application reverse engineering processes are presented below.

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 [36].

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 Section 2.1 several models proposed in the literature for representing a Web application were briefly discussed. Among these models, those proposed by Conallen’s methodology for developing Web applications can be considered: according to Conallen, UML diagrams are exploited to specify several views of a Web application, including the user requirements view, the structural view, the implementation view, and the view of the dynamic interactions among the structural entities of the application [12,13,17]. These views are modelled by
use case diagrams, class diagrams, component diagrams, and sequence/collaboration diagrams, which are currently adopted for producing the specifications of a software system. Some examples of these diagrams will be provided in Section 6.

With respect to other models presented in the literature, the suite of diagrams proposed by Conallen has the advantage of using the standard UML language to represent a Web application from different points of view (e.g. the user, and the designer point of view) and at various degrees of detail, providing a variety of information (either about the structure or about the dynamic model of the application) that can be successfully used to support different tasks involving an existing Web application. Experiments described in the literature show the adequacy of these diagrams for the purposes of supporting comprehension [31,37], testing [32], analysis and restructuring [26], and maintenance [24] of a Web application.

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 runtime 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 program slicing, necessary in order to localize 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 will be precisely set out.

In the next section, the GMT approach will be used to define a reverse engineering process aiming to support the comprehension of undocumented Web applications.

4. THE WARE’S REVERSE ENGINEERING PROCESS FOR COMPREHENDING A WEB APPLICATION

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 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 localization of the set of software entities that collaborate to implement the functional requirements of the application.

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 organization 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 specialization 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 will be deduced between the use case and the Web application items involved in the sequence diagram. In Section 6, some examples of these diagrams are shown, referring to some case studies we carried out.

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. Additional techniques for analysing the Web application structure and identifying relevant subsets of its components were also considered: clustering techniques were defined to carry out this analysis. 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, the WARE-tool, that partially automates the execution of most of the process tasks: this tool is described in the next section. Figure 2 illustrates the process, while additional details about each step of the process are provided below.

### 4.1 Static analysis of the Web application

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 Section 2. 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, and the
statements producing *link*, *submit*, *redirect*, *build*, and other relationships are identified and localized 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 [38], or the GXL Graph Exchange Language, which enable the exchange of information derived from programs, which will be conveniently represented in a graph [39,40], or by using any tagged syntax format designed to represent the necessary information efficiently.

### 4.2. Dynamic analysis of the Web application

In a dynamic Web application, the set of entities making up the application can be significantly modified at runtime, thanks to the facility offered by script blocks, of producing new code that will be 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 (cf. the class *Built Client 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 runtime 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.

4.3. Automatic clustering of the Web application

The information about the Web application obtained by static and dynamic analysis can be used to
produce a graph whose nodes represent the set of Web application entities, and whose edges describe
the different relationships among these entities. In [37], this kind of graph has been named WAG, Web Application connection Graph. WAG analysis may support the comprehension of the application.
However, since this graph may be large (in terms of the number of nodes and edges) even in the case
of small size Web applications, in order to simplify the analysis of large WAG graphs, some kind of
automatic clustering [41] can be used to decompose this graph into smaller cohesive parts.

In the third step of the reverse engineering process, the automatic clustering approach proposed
in [37] is applied, in order to group software items of a WAG into meaningful (i.e. highly cohesive) and
independent (i.e. loosely coupled) clusters. This clustering approach evaluates the degree of coupling
between entities of the application (such as server pages, client pages and client modules) that are
interconnected by Link, Submit, Redirect, Build, Load in Frame, and Include relationships.

Each type of relationship is associated with a different weight (the Build relationship is considered
stronger than Link and Submit ones, and the Submit relationship is considered stronger than the Link
one), and the coupling between items is quantified on the basis of the number and the type of direct
relationships (or edges) between them: the more the interconnections between items, the stronger their
coupling.

Once the coupling between WAG entities has been evaluated, a hierarchical clustering algorithm is
used to produce a hierarchy of clustering configurations, providing different partitioning of the WAG
at different levels of abstraction‡.

From the clustering configurations generated by the hierarchical algorithm, the ‘optimal’ one is
selected, that is the configuration that maximizes a quality metric based on evaluation of the degree
of intra-connectivity of the clusters and of the degree of inter-connectivity between clusters: intra-
connectivity is a weighted mean of the inner edges of a cluster, and inter-connectivity is a weighted
mean of the edges among clusters.

This ‘optimal’ configuration is considered the most suitable for including clusters implementing
functions at higher levels of abstraction than that of the cluster’s single items. However, depending on
the type of edges between its inner nodes, a cluster may actually implement a well-defined function, or
may not. Therefore, validation of the clusters, based on a Concept Assignment Process [42] has to be
carried out.

The Concept Assignment Process allows valid clusters, i.e. clusters of entities actually implementing
a well-defined function, to be distinguished from invalid ones. According to the cluster classification
proposed in [37], invalid clusters may be classified as spurious, divisible, or incomplete clusters.

‡Agglomerative hierarchical clustering algorithms start from the individual items of the graph, gather them into small clusters,
which are in turn gathered into larger clusters, up to a final cluster containing everything. The result is a hierarchy of clusters.
A spurious cluster includes items exhibiting a low functional cohesion (i.e. they contribute to the implementation of distinct unrelated functions), a divisible cluster comprises items that may be separated into smaller, cohesive sub-clusters, while an incomplete cluster does not include all the items necessary to implement a function. Invalid clusters can be analysed to see whether they can be transformed into valid ones. For example, incomplete clusters may be merged with further components, while spurious and divisible clusters may be split into valid smaller ones.

During the Concept Assignment Process, the code of the Web application will be analysed and the application execution will be observed to support understanding of the cluster behaviour. The output of the phase consists of the set of validated clusters, where each cluster will include a set of Web application items and will be associated with a description of the function it implements.

4.4. Abstraction of the UML diagrams

In the final step of the reverse engineering process, UML diagrams have to be abstracted on the basis of the information retrieved in the previous steps. The following abstraction processes have been defined for reconstructing them.

The class diagram depicting the Web application will be obtained by analysing the information about the Web application entities and relationships retrieved by static and dynamic analysis. This class diagram will be drawn according to the conceptual model presented in Section 2.1. In particular, the class diagram modelling the relevant entities of a given Web application and their relationships will be an instantiation of the model in Figure 1, where each Web page and each inner page entity will be represented as a class, with a stereotype describing the type of entity according to the classes represented in Figure 1 (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 will be represented, with names conforming to the ones modelled in Figure 1 (such as link, build, redirect, include, etc.).

Moreover, according to Conallen’s notation, each class will be characterized by attributes corresponding to the variables it references, and by methods corresponding to the functions and scripts included in it.

As for the use case diagrams, they can be deduced on the basis of the clustering results, according to the approach proposed in [37]: each validated cluster may be associated to a use case implementing a user functional requirement of the application. Relationships among use cases may be deduced by analysing the links between corresponding clusters. As an example, if a cluster associated with a use case ‘A’ is linked only to 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 merely with suggestions about how use case diagrams could be drawn. Establishing the right kind of relationship is not a straightforward task, even in forward engineering [43], so a careful analysis of the clusters is recommended to define and validate the relationships between use cases. In addition, once a first ‘flat’ diagram including all the recovered use cases of the application has been produced, further use case diagrams at higher abstraction levels can be deduced with a bottom-up approach, that analyses the relationships between use cases and then gathers sets of related use cases together in single, more...
abstract ones. For example, a use case exclusively included by another one will be grouped with it to provide a more abstract use case belonging to a higher level use case diagram.

As for the UML sequence diagrams abstraction, for each use case (i.e. validated cluster) it is possible to produce a sequence diagram whose objects derive from classes associated with the cluster’s entities, while interactions among objects can be deduced both from the relationships between classes in the class diagram and from the results of the dynamic analysis. Dynamic analysis results are indispensable in order to deduce the sequence of messages exchanged between entities involved in a given relationship. Each message exchange will be associated with a label indicating the corresponding action type (such as load, build, submit, etc.), and may have a list of parameters deduced by analysing the data flow between interconnected items in the Web application.

5. THE WARE-TOOL

The reverse engineering process proposed in the previous section can be executed with the support of a reverse engineering tool [31], named the WARE-tool. The WARE-tool has been designed as an integrated environment including several components arranged in the software architecture shown in Figure 3. As the figure illustrates, the WARE-tool is composed of three layers: the Interface Layer, the Service Layer, and the Repository. The Interface Layer implements the user interface providing access to the functions offered by the tool and visualization of recovered information and documentation both in textual and graphical format. The Service Layer implements the tool services, and includes two main components: Extractors and Abstractors. The Repository stores the information extracted and abstracted about the Web application using intermediate format files and a relational database. Additional details about the WARE-tool’s layers are provided in the following.

5.1. WARE-tool Service Layer

The Service Layer of the WARE-tool 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 information retrieved directly.

The Extractors included in the WARE-tool are source code parsers implemented in C++ language; the set of analysable source code languages is constantly in evolution. The current version of the tool includes parsers that analyse the HTML version 4.0 language and some scripting languages, such as the Javascript and VBScript languages usable at the client side of a Web application, and the ASP and PHP scripts at the server side. Parsers do not recover an abstract syntax tree from the analysed code, but just recognize and identify in the code the information needed to re-build the required diagrams, by means of lexical and syntactical analysis. Parsers are also able to recognize some lexical and syntactical errors contained in the source files. Finally, the parsers store the extracted information in an intermediate representation form (IRF) that is implemented as a tagged XML-like file (see [31] 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. The relational database mirrors the conceptual model of a Web application presented in Section 2.
As for the Abstractors, the Clustering Executor implements the clustering algorithm for collapsing the Web application into highly cohesive and loosely coupled clusters. The Query Executor is the abstractor that implements predefined SQL queries on 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 sequences of function calls or link activations in a given page. Finally, the UML Diagram Abstractor implements several abstraction tasks supporting the recovery of UML diagrams, including
Figure 4. Some snapshots from the WARE-tool. (a) WA components reachable from a selected component; (b) definition of subsets of WA components.

the class diagrams at different degrees of detail (e.g. providing only client pages, or static pages, or filtering out the forms, etc.), as well as sequence diagrams and use case diagrams of a Web application.

5.2. WARE-tool Interface Layer

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

The main functions available to the user include (1) parsing of a Web application code, (2) translation of the IRF into the database, (3) software comprehension activities, such as (3.1) executing automatic clustering, (3.2) exploring the inventory of the Web application entities and viewing their source code, (3.3) displaying the set of entities that are reachable from a selected entity. In addition, the user is able to select a subset of Web application entities, according to a given partitioning criterion implemented by the WARE-tool.

As an example, the tool enables automatic identification of entity subsets whose files are included in the same directory, or belong to the same reachability set. The subsets obtained can be validated by the user, associated with a description and stored in the database. Figure 4(a) shows the items reachable from a selected component (i.e. the client page selected in the left box in the figure) and the graphic visualization of this subset. Figure 4(b) shows the form allowing the user to create a subset by selecting the components from the inventory list. The same form is used to display the results of the clustering execution.

Another available user function provides graphic visualization of the class diagram depicting the static software architecture of the Web application. Currently, the diagram is graphically visualized
with the support of some freeware graph displayers, such as VCG [44] and Dotty [45]: this visualization does not support the UML notation style, but different shapes and colours are used to draw different kinds of entity and relationship deriving from the reference conceptual model of a Web application (cf. Figure 1). For example, a box is used for drawing a static page, a trapezoid for a built client page and a diamond for a server page.

Users are allowed to formulate customizable queries on the database, by choosing the type of application item, relationship, or parameter to be searched for and displayed. Finally, some summary measures of the Web application, such as the number of Web application pages, scripts, or the LOC count are automatically computed by the tool and shown to the user on request.

6. EXPERIMENTING THE REVERSE ENGINEERING APPROACH

The reverse engineering approach described in this paper has been submitted to a validation experiment in order to assess its feasibility and effectiveness, and to explore possible areas for improvement.

The validation 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 two or three people, and each team was assigned a single application.

For 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 [16], three of them were class 3 applications including dynamic client and server pages (hereafter we will call these applications 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; 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 online 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 also used to define the navigational structure 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.

§WA6 implements the online reference guide of a programming language and is considered primarily a static class 1 application, even if it contains some client scripts, since these scripts are just used to provide examples about the programming language, but do not implement any user function.
Table I. Statically retrieved data from the analysed Web applications.

<table>
<thead>
<tr>
<th>Item type</th>
<th>WA1</th>
<th>WA2</th>
<th>WA3</th>
<th>WA4</th>
<th>WA5</th>
<th>WA6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server page</td>
<td>75</td>
<td>105</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Static page</td>
<td>23</td>
<td>38</td>
<td>19</td>
<td>80</td>
<td>45</td>
<td>275</td>
</tr>
<tr>
<td>Built client page</td>
<td>74</td>
<td>98</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Client script</td>
<td>132</td>
<td>225</td>
<td>113</td>
<td>261</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>Client function</td>
<td>48</td>
<td>32</td>
<td>60</td>
<td>68</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Form</td>
<td>49</td>
<td>100</td>
<td>5</td>
<td>0</td>
<td>25</td>
<td>5</td>
</tr>
<tr>
<td>Server script</td>
<td>562</td>
<td>2358</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Server function</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Redirect (in server scripts)</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Redirect (in client scripts)</td>
<td>0</td>
<td>0</td>
<td>41</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Link</td>
<td>45</td>
<td>266</td>
<td>121</td>
<td>162</td>
<td>448</td>
<td>1508</td>
</tr>
</tbody>
</table>

6.1. Carrying out static analysis

In the first step of the reverse engineering process, the teams carried out static analysis with the support of the WARE-tool, with the aim of detecting the Web applications’ items and their relationships. Table I 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 in Figure 1.

Thanks to the automation of the analysis by the WARE-tool, 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 s to 62 s) that depended on the size of the application analysed (cf. Table V about effort data).

6.2. 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 runtime 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 (cf. the Link, Submit, and Redirect associations in Figure 1). 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
Table II. Dynamically retrieved relationships from the analysed Web applications.

<table>
<thead>
<tr>
<th>Relationship type</th>
<th>WA1</th>
<th>WA2</th>
<th>WA3</th>
<th>WA4</th>
<th>WA5</th>
<th>WA6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Link</td>
<td>0</td>
<td>1</td>
<td>9</td>
<td>0</td>
<td>27</td>
<td>0</td>
</tr>
<tr>
<td>Submit</td>
<td>0</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Redirect (in client script)</td>
<td>0</td>
<td>0</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Redirect (in server script)</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Figure 5. The WAG of the WA1 application.
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 II 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.

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 V.

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

6.3. Clustering the Web applications

In the third step of the process, in order to support comprehension of each Web application, automatic clustering was executed by the WARE-tool.

The optimal clustering configurations proposed by the tool for each Web application were submitted to a Concept Assignment Process, in order to understand the functions implemented by the clusters and to validate them. The process was carried out with the support of the WARE-tool, and by inspecting the code and observing the Web application execution.

The optimal clustering of the WA1 application presented 49 clusters, with an average number of 3.58 pages. Figure 6 shows the graph of this clustering, where each node represents a cluster. The size of this graph is sensibly smaller than the WAG, since it includes 49 nodes, that is, less than 30% of the nodes of the original WAG.

During this phase of the experiment, some general indications about cluster validation emerged. In particular, it was observed that validation of the clusters often depends on the role of hypertextual links allowing interconnection of Web pages. Indeed, a Web application may often include redundant
Figure 6. The optimal clustering of the WA1 application.

Table III. Results from clustering validation.

<table>
<thead>
<tr>
<th></th>
<th>WA1</th>
<th>WA2</th>
<th>WA3</th>
<th>WA4</th>
<th>WA5</th>
<th>WA6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of initial clusters</td>
<td>49</td>
<td>101</td>
<td>27</td>
<td>49</td>
<td>31</td>
<td>115</td>
</tr>
<tr>
<td>Number of spurious clusters</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Number of split clusters</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>Number of incomplete clusters</td>
<td>8</td>
<td>15</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Number of valid clusters</td>
<td>41</td>
<td>86</td>
<td>21</td>
<td>48</td>
<td>26</td>
<td>103</td>
</tr>
<tr>
<td>Number of new clusters obtained from the spurious ones</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Number of new clusters obtained from the subdivided ones</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>13</td>
<td>31</td>
</tr>
<tr>
<td>Number of new clusters obtained by merging incomplete clusters</td>
<td>3</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Number of final valid clusters</td>
<td>44</td>
<td>93</td>
<td>28</td>
<td>50</td>
<td>39</td>
<td>134</td>
</tr>
<tr>
<td>Cluster</td>
<td>Assigned concept</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>------------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Main system function selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Teacher login</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Teacher function selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Teacher function overview</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Teacher function coordination</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Tutoring date insertion (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Tutoring date deletion (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Tutoring function selection (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>List of registered students for tutoring (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Teacher registration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Assign a course to a teacher</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>Teacher data deletion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Undergraduate course insertion</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Teacher and course data update</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Teacher and course function selection (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Course deletion (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>Student data deletion (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>Student enrollment function selection (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>List of enrolled students display</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>Examination schedule modification</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Insertion of a new date in the examination schedule</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Deletion of a date from the examination schedule</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>List of students registered to an examination (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Examination schedule function selection (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>Examination schedule list display (available to teacher)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>Bulletin board function selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>List the news in the bulletin board</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>News insertion in the bulletin board</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>News deletion from bulletin board</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Student function selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>Lost password request (available to students)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>Student enrollment function selection</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>Student data update (available to students)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>Student course enrollment</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>Examination function selection (available to students)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>Examination schedule listing (available to students)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>Register a student to an examination</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>Tutoring request function selection (available to students)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>Tutoring requests insertion (available to students)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>Tutoring date deletion (available to students)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>Tutoring date update (available to teachers)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>General constants definition</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>Insertion of a new date in the examination schedule (obsolete functionality)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>Work in progress page (for not yet implemented functionalities)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
hypertextual links that Web programmers insert in the application just to speed user navigation through the Web pages. Such kinds of links will be called navigational links and include those allowing the user to access a previously visited page directly (called back link), or links implementing a short-cut to a longer navigation path in the Web application (called cross link). Instead, we call the links that are actually needed to connect Web pages implementing user functions semantic links.

In the experiment, it was observed that clusters composed of Web pages mostly interconnected by semantic links were more likely to be classified as valid clusters, while clusters including pages that are interconnected by many navigational links were more often considered to be invalid clusters.

The final results of the validation step are listed in Table III, which reports in the upper part the number of initial clusters proposed by the tool, and the number of spurious, split, incomplete, and valid clusters. The lower part of the table shows the final number of clusters obtained by modifying the originally proposed clusters.

For the WA1 application, the result of the Concept Assignment Process was that 41 valid clusters were identified compared with eight incomplete clusters. However, after more accurate analysis, the eight incomplete clusters could be merged into new clusters, each one actually implementing a function. Table IV reports the list of final valid clusters and the concepts assigned to them. All the basic functions offered to students and teachers by the Web application are listed in the table, which includes functions for accessing published information about the course, or for downloading teaching material (accessible only by registered students), functions allowing a student to apply for a course or an examination session, functions allowing a teacher to manage the examination and student tutoring schedule, and so on. Some clusters implementing utility modules, unused functions, and incomplete functionalities were also detected (cf. the last three rows in the table).

6.4. Abstracting the UML diagrams

In the conclusive step of the process, a set of UML diagrams depicting the Web applications at a higher level of abstraction had to be reconstructed.

Class diagrams, use case diagrams and sequence diagrams were recovered by applying the approach proposed in Section 4.4. Some of the tasks needed for abstracting these diagrams were carried out by the WARE-tool, while others required a significant amount of human intervention. In particular, the class diagram describing the modular structure of each application was obtained automatically by the tool. Moreover, a preliminary ‘flat’ use case diagram could be deduced automatically from the optimal clustering configuration of each application by associating each validated cluster with a use case. The relationships between use cases were recovered manually.

Therefore, the recovery of use case diagrams at higher abstraction levels was achieved by a bottom-up approach requiring human knowledge and expertise, and partially supported by the tool.

Analogously, the reconstruction of the sequence diagrams started from the clustering results, and then proceeded manually, with analysis of the interactions between cluster items.

Figure 7 shows an excerpt from the class diagram of WA1, while Figure 8 shows an excerpt of the initial flat use case diagram of the same application, produced from the validated clusters of the optimal clustering configuration. In the latter excerpt, some use cases available to students are reported, while Figure 9 shows a more abstract diagram describing the set of WA1 use cases obtained by gathering together logically related use cases from the flat diagram. For example, the use cases named ‘Students
Figure 7. An excerpt of the class diagram of the WA1 application.

Figure 8. An excerpt of the use case diagram of the WA1 application.
Enrolment’, ‘Examination Schedule’ and ‘Tutoring Management’ correspond to the three groups of use cases contained in round boxes in Figure 8.

In Figure 8, the use case named General Constant Definition derives from a cluster consisting of a single page that was linked to many other pages of the application. This page defines the values of named constants referred to in the application. Of course, it is not actually a use case, since it does not represent any user function, but it can be considered as an internal utility module. However, it was reported in the flat use case diagram, too, in order to keep trace of this module used in the application.
Finally, sequence diagrams associated with use cases could be recovered. Figure 10 reports the sequence diagram associated with the use case allowing addition of a new examination session date to the schedule.

6.5. 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.
Table V. Effort data and computational times from the reverse engineering experiments.

<table>
<thead>
<tr>
<th>Task</th>
<th>WA1</th>
<th>WA2</th>
<th>WA3</th>
<th>WA4</th>
<th>WA5</th>
<th>WA6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static analysis (s)</td>
<td>14</td>
<td>52</td>
<td>19</td>
<td>30</td>
<td>22</td>
<td>62</td>
</tr>
<tr>
<td>Dynamic analysis (man hours)</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>0.5</td>
<td>2</td>
<td>0.25</td>
</tr>
<tr>
<td>Automatic clustering (s)</td>
<td>2</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>412</td>
</tr>
<tr>
<td>Clustering validation (man hours)</td>
<td>10</td>
<td>16</td>
<td>6</td>
<td>5</td>
<td>5</td>
<td>18</td>
</tr>
</tbody>
</table>

Note: Tasks of static analysis and automatic clustering were carried out automatically by the tool WARE-tool running on a PC with an Intel Pentium 4 at 2 GHz CPU.

As for 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 for judgement by the 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 we can conclude that the proposed approach is adequate.

For the efficiency of the reverse engineering process, the distribution of the effort required to carry out some steps of the process was evaluated. Table V lists the effort 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 V 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 we can conclude that the process was also efficient.

In addition, by observing and analysing the activities carried out by the experimenters during the most expensive steps, it was possible to 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...
Figure 11. An example of HTML code including 'named' links.

```html
......
<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 12. An example of a Web page with commented lines providing internal documentation.

```html
%@ 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="login,password">
<meta name="Database" content="../basedatisito.mdb">
<meta name="Images" content="bgmain.gif">
</head>

<body>
......
```

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 11 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.

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 12. 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 recognized 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.

7. CONCLUSIONS

This paper presents the WARE approach defining a reverse engineering process involving Web applications. The process was defined using the GMT paradigm, and includes methods and tools that support the comprehension of existing, undocumented Web applications during maintenance. The aim of the process is to reconstruct UML diagrams illustrating various aspects of the structure, behaviour and dynamic dimensions of a Web application.

The proposed reverse engineering process includes both steps that can be carried out automatically and steps that require intervention by a software engineer. The feasibility and effectiveness of the process have been evaluated by a preliminary validation experiment involving a set of undocumented Web applications.

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.

A relevant research question to be investigated is empirical assessment of how to introduce the proposed reverse engineering approach in maintenance processes in order to obtain significant improvements of the process quality, and of the quality of its products. This empirical evaluation will be addressed in future work.
Moreover, in future work, some evolutions of the WARE-tool will be implemented, in order to obtain a more comprehensive environment supporting the proposed reverse engineering process. In particular, in order to improve the usability of WARE-tool outputs, a bridge towards an environment, such as the Rational Rose Suite [46], will be implemented to produce and maintain Web applications by enabling browsing and design of UML diagrams.

Other work will focus on the development of a tool providing greater automatic support of dynamic analysis execution (currently one of the main limitations of the WARE-tool).

In addition, possible techniques providing automatic support of the comprehension of a Web application page, based on semantic analysis of the pages content, will be investigated and integrated in the WARE-tool. Finally, in order to provide maintainers with further information supporting the comprehension and evolution of an existing Web application, reverse engineering methods for recovering an object-oriented conceptual model of the domain of the Web application will be investigated and implemented in the WARE-tool.

REFERENCES

http://lon.ifs.uni-linz.ac.at:8080/databwe/200003/msg00007.html [26 February 2000].


AUTHORS' BIOGRAPHIES

Giuseppe A. Di Lucca received his Laurea degree in Electronic Engineering from the University of Naples ‘Federico II’, Italy, in 1987 and his PhD degree in Electronic Engineering and Computer Science from the same University in 1992. He is currently an Associate Professor of Computer Science at the Department of ‘Ingegneria’ of the University of Sannio. Previously, he was with the Department of ‘Informatica e Sistemistica’ at the University of Naples ‘Federico II’. Since 1987 he has been a researcher in the field of software engineering and his list of publications contains more than 50 papers published in journals and conference proceedings. He serves in the program and organizing committees of conferences in the field of software maintenance and program comprehension. His research interests include software engineering, software maintenance, reverse engineering, software reuse, software reengineering, Web engineering, and software migration.

Anna Rita Fasolino is an Assistant Professor of Computer Science at the University of Naples ‘Federico II’, Italy, where she holds a Professorship in Software Engineering. She received her Laurea degree in Electronic Engineering (cum laude) and her PhD degree in Electronic Engineering and Computer Science from the University of Naples ‘Federico II’, Italy. From 1998 to 1999 she was at the Computer Science Department of the University of Bari, Italy. Her research interests include software maintenance and quality, reverse engineering, Web engineering, and software reuse.

Porfirio Tramontana received his Laurea degree in Computer Engineering from the University of Naples ‘Federico II’ in 2001. He is currently a PhD student at the Department of ‘Informatica e Sistemistica’ of the University of Naples ‘Federico II’. His present research interests are in the fields of software engineering, software maintenance, reverse engineering, software comprehension, Web engineering, and software reengineering.