Understanding Web Applications through Dynamic Analysis

Giuliano Antoniol, Massimiliano Di Penta and Michele Zazzara
antoniol@ieee.org dipenta@unisannio.it zazzara01@libero.it

RCOST - Research Centre on Software Technology
University of Sannio, Department of Engineering
Palazzo ex Poste, Via Traiano 82100 Benevento, Italy

Abstract

The relevance and pervasiveness of web applications as a vital part of modern enterprise systems has significantly increased in recent years. However, the lack of adequate documentation promotes the need for reverse engineering tools aiming at supporting web application maintenance and evolution tasks.

A non trivial web application is a complex artifact integrating technologies such as scripting languages, middleware, web services, data warehouses and databases. The task to recover abstractions requires the adoption of dynamic analyses to complement the information gathered with static analyses.

This paper presents an approach and a tool, named WANDA, that instruments web applications and combines static and dynamic information to recover the as-is architecture and, in general, the UML documentation of the application itself. To this aim we propose an extension of the Conallen UML diagrams to account for detailed dynamic information. The tool has been implemented and tested on several web applications. Its architecture has been conceived to allow easy customization and extension. The paper presents our tool in the context of a program understanding task; however, it can be usefully applied to many other tasks such as profiling, security and dependability verification and application restructuring.

Keywords: web applications, UML, dynamic analysis

1. Introduction

Initially conceived as stand alone means to browse online catalogs, to access guestbooks or to handle counters, Web Applications (WAs) rapidly changed their role. Today they are vital company assets, tightly integrated into many business critical systems. Online e-banking, electronic marketplaces, stock market and e-trading, electronic access to public administration services are just a few examples of domains where the WA importance is clearly demonstrated.

Much in the same way as experienced with *traditional* applications, a significant growth of WA size and complexity is observed. However, the turbulent WA market diffusion has not fostered the adoption of disciplined development and evolution processes. Indeed, the market pressure plays a central role: very often the WA window of opportunity is very narrow, thus development and deployment cycles are dramatically reduced, sacrificing an adequate level of documentation.

Suddenly enough, very often, the development of a WA is thought of as a craftsmanship activity, where the developer is a *craftsman* or, even worse, an *artist* rather than an engineer. The adoption of modern technologies is mostly limited to shorten the implementation phase; often the development focus is centered and limited to the look and feel, the presentation aspects. The latter trend constitutes a problem in a *scenario* of WA continuous maintenance and evolution. As pointed out by Holt et al., WAs will be the next generation of legacy applications, the future challenge [1] for software maintenance. In other words, there is the need for developing a new generation of methodologies, methods, technologies and tools to help maintaining and evolving WAs.

As highlighted by several papers dealing with WA life cycle published in the last five years, the WAs present peculiarities that make a reverse engineering process somehow more complex than for traditional software systems. Moreover, the adoption of many different technologies made WAs very complex objects. WAs are built upon middlewares, they interact via API with DataBase Management Systems (DBMS), sometimes they act orchestrating several web-services to accomplish a specific task, etc.

To deal with all these aspects, static analysis does not suffice. This is already well known for traditional software: tasks such as the recovery of design patterns [2], scenarios and sequence diagrams [3, 4] are only some examples.

Proceedings of the 12th IEEE International Workshop on Program Comprehension (IWPC'04)
1092-8138/04 $ 20.00 © 2004 IEEE

WAs tend to be even more highly interactive and dynamic than traditional applications; HTML pages can be dynamically built by *server pages*, thus the WA interface changes at run-time. Moreover, even pieces of code (e.g., client-side scripts) can be dynamically generated. Static analysis is likely to give only an imprecise and approximate picture, and only dynamic analysis allow a proper understanding of complex and dynamic application behavior (e.g., the client-side logic). Dynamic analysis also allows to track several other information, such as the session and cookie data, the DBMS tables and queried entities, or the frequency of exercising a particular link [5], and the type of link actually exercised (e.g., hyperlinks, submit with GET or POST).

To recover the architecture of a WA, it is often mandatory to inspect the interactions of the WA with external commands, files, and databases. Indeed, a WA is almost always a complex object encompassing external programs and applications, HTML and dynamically generated components, as well as DBMS interaction [6]. Dynamic analysis, in this case, constitutes a means to overcome the difficulty due to the fact that program parameters or databases query strings are almost always dynamically generated.

Finally, with the diffusion of the service-centric software engineering, today's WAs (and, in general, any modern software system) can be thought of as the composition of several distributed, interacting services. However, in most cases the instance of the service that is invoked from a client is subject to late binding. Similarly to polymorphic calls, late binding forces the adoption of more complex analyses, in that static analysis may not suffice to recover accurate information.

This paper proposes an approach and a tool, named *WANDA* (Web ApplicatioNs Dynamic Analyzer) for the dynamic analysis of WAs. HTML pages (regardless of containing server-side scripting) are instrumented; information collected during executions is stored in a database. Dynamic information is combined with static information to extract UML documentation: component, deployment, sequence and class diagrams. Component and deployment diagrams highlight not only the multi-tier architecture of the WA, but also the interaction with web services, DBMS, files, etc. Sequence diagrams show the interaction between boundary, control and entity classes.

Finally, class diagrams represent the WA according to Conallen UML extensions [7]. The Conallen UML model has been further extended by defining novel stereotypes and tagged values to represent the frequency of invocation for each association.

By annotating diagrams with the frequency of events, the maintainers can immediately visualize the interactions actually taking place during the WA in field usage. Dynamic information highlights executed use-cases and scenarios, as well as finer grained details, such as the number of times a link was followed, the frequency of access to databases and type of operations performed, or the load of a link between two peers. To ease the interoperability with visualization and CASE tools, *WANDA* diagrams are represented in XML.

With respect to existing tools and approaches, this paper proposes:

- A thorough instrumentation to the WA, combined with data flow analysis, to analyze session handling, database, file and web service accesses. Such information allows to recover sequence diagrams stereotyping *boundary*, *control* and *entity* objects, as well as detailed deployment and component diagrams;
- An approach to separate the analysis of the HTML from that of the server-side scripting language;
- The extension of Conallen diagrams to account for dynamic information, i.e., for tagging associations and links between peers with frequencies and types of values/operations;
- A model to represent WA dynamic information; and
- The recovery of static and dynamic Conallen UML diagrams, extended as outlined above.

The paper is organized as follows. After a review of the related works, the *WANDA* architecture is described in Section 3. Section 4 explains how diagrams are extracted, extended to encompass dynamic information, and represented. Section 5 reports and discusses results from a case study used to show the tool potentialities. Section 6 concludes and outlines the directions for future work.

2. Related Work

Reverse engineering of WAs is quite a recent field. Antoniol et al. in [8] proposed an approach, based on the Relational Management Methodology (RMM), to recover web site architectures. The first significant contribution was given by Ricca and Tonella, who developed the *ReWeb* tool to perform analyses on web sites [9, 10, 11]. In particular, Ricca and Tonella introduced a graphical representation of the web site to extend to WAs traditional static flow analyses such as reachability, dominance, and data flow analysis. *ReWeb* performs the mentioned analyses plus it detects navigational patterns. Ricca and Tonella also proposed to enhance the analyses considering dynamic information [5]. Clearly, *ReWeb* does not need page instrumentation; on the other hand, web server logs, as already underlined, do not allow fine-grained analyses.

An approach [12] and, then, a tool, named *WARE*, to recover Conallen's UML documentation from WAs has been proposed by Di Lucca et al. [13]. *WARE* performs static analyses on WAs, stores the extracted information into a database and then uses such an information for the reverse-engineering of UML diagrams. Dynamic analysis was used in [14] to complement static information required to detect page clusters. Di Lucca and al. also presented an approach to recover use case diagrams, sequence diagrams and business object models [15, 16]. Their approach is based on static analysis and clustering, and they also agreed that diagram information can be refined using dynamic information.

Boldyreff and Kewish proposed a methodology for the reverse engineering of WAs, with the purpose of identifying duplications and improving maintainability [17]. Vanderdonckt et al. proposed a tool, named *Vaquista* [18], for reverse engineering of the presentation model of WAs. Tilley and Huang [19] compared the reverse engineering capabilities of commercial WA development tools. Stroulia et al. [20] proposed an approach to identify WA services with the purpose of WA interoperability. Commonalities can be also found with Hassan and Holt [1], who proposed a tool for WA architecture recovery. In particular, they also analyzed, although only statically, database accesses, and they also identified three layers in the WA (presentation, business and infrastructure).

With respect to the aforementioned contributions and tools, *WANDA* aims to recover fine grained details that allow a more accurate reconstruction of the WA architecture and dynamic behavior.

3. The WANDA Architecture

The *WANDA* architecture is depicted in Figure 1. It may be thought of as a layered architecture organized into three horizontal layers:

1. A *data layer*, responsible for storing the extracted static and dynamic information, as well as the abstracted UML documentation;
2. A *business logic layer*, that contains all the analysis and transformation subsystems; it manipulates the information at various levels of abstraction;
3. An *interface layer*, to access the original WA, the traces collected by the *in-field* execution of the instrumented WA; the interface layer is also responsible for the visualization of the results and diagrams.

On the other hand, reading the figure from left to right, it is a complementary organization of analyses into different levels of abstraction:

1. Parsing and instrumentation;
2. Information extraction;
3. Model abstraction; and
4. Result presentation.

WANDA has been conceived as a modular infrastructure for the analysis of WAs. For instance, it is quite easy to add/replace the parsers and the static information extractors for supporting multiple scripting languages. Dynamic information is extracted from execution traces and, since the same format is used for all the languages, no (or a few) change is needed for the dynamic information extraction module.

Similarly, the abstraction module relies on the information stored in the data layer, and it can be easily replaced by any one of the approaches proposed in literature.

3.1 Preprocessing and Parsing

When analyzing WAs, the first issue is related to parsing HTML interleaved freely with the server-side scripting language[1]. In this case, even island-parsing [21] approaches (that identify and parse only the code portion of interest, skipping anything else) may fail. To the author's knowledge, most of the existing tools rely on extracting the needed information by regular expressions and pattern-matching.

In our approach, all pages undergo a preprocessing step to separate HTML and scripting language with the final goal to create a semantically equivalent WA entirely and solely written in a pure scripting language (e.g., PHP pages). HTML sections are extracted, stored in a separate repository, and replaced with statements of the form `_html(n)` to dynamically load the needed HTML fragment. The parameter `n` indicates the pointer to the HTML block referred. Figure 2 shows an example of such a preprocessing step. In particular, Figure 2-a shows the original HTML, Figure 2-b the resulting PHP page and Figure 2-c the content of the HTML repository.

Once preprocessed, PHP pages can be easily parsed. To develop a PHP parser we adopted the freely available *JavaCC* [22] tool, a top-down Java-based parser generator.

3.2 Static analysis

In our approach, pure static analysis is limited to server-side scripting. We postponed the static analysis of HTML, SQL, and client-side scripting. Our assumption is that, in

[1] Without loss of generality, from this point we will refer to analyzing PHP. However, the solution for other scripting languages is the same.

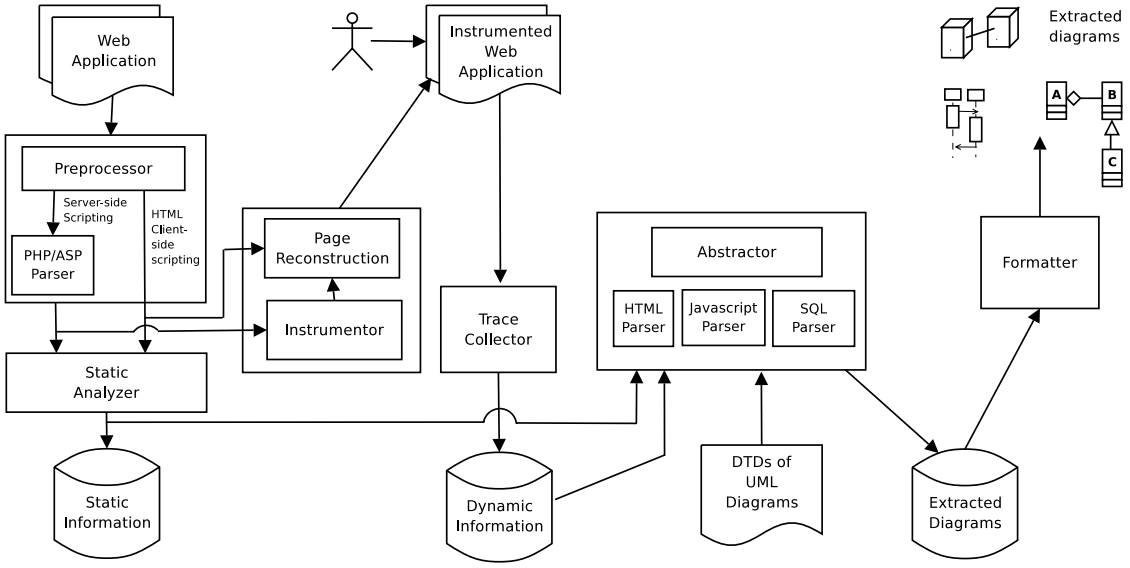

Figure 1. The WANDA architecture.

most cases, these are automatically generated and thus they require the analysis of dynamically collected information.

At this stage, static analysis is mainly devoted to extract metrics, identify building blocks (functions, classes) and particular categories of statements and function/method calls, such as database access, file I/O, or session and cookie management.

3.3 Instrumenting

This module constitutes the core of *WANDA*. Despite the preliminary static analysis performed, most of the information is collected by probes inserted at this stage. Probes are organized into a taxonomy related to the collected dynamic information:

- *Page access:* all pages, both pure HTML pages and PHP pages, are instrumented to keep track of their access[2];
- *HTTP environment variables:* this is a significant source of information. We used them to detect the variable passed to the page as parameters (with GET or POST). Session variables are also tracked;
- *Cookie, header and session management functions:* instrumentation keeps track of the calls to these functions, and to the actual values of the passed parameters;
- *Database and file I/O:* similarly to the previous case, calls to these functions are tracked. This allows recognizing several elements of the WA architecture, such as the different DBMS and database applications accessed, as well as the hosts on which they are located. The query strings sent to the databases are collected to be successively parsed and analyzed;
- *Invocation of web services:* probes allow to keep track of the addresses of hosts on which invoked services are located, the services invoked and the frequency of invocation for each service. In this case the analysis is slightly more complex with respect to the other cases. For the PHP language, we need to identify objects instance of the class *SOAP_Client*[3], in that each message sent to these objects corresponds to a service invocation. Using dynamic analysis, this can be done by simply building an allocated object symbol table and querying it at run-time. Clearly, a different type of analysis may be needed for different languages or web service libraries.
- *Access to libraries and COTS:* the tool keeps track of the access to external components (e.g., *ActiveX*); this constitutes important information for extracting component diagrams.

The above mentioned information is considered, in the author's opinion, to be enough representative of the features contained in the largest part of WAs. However, for more complex WA, the model (and thus the instrumentation) can

[2] At this purpose, the web server has been properly configured, to treat `.html` files as PHP files and thus interpreting the contained probes.

[3] Using the PEAR::SOAP library.

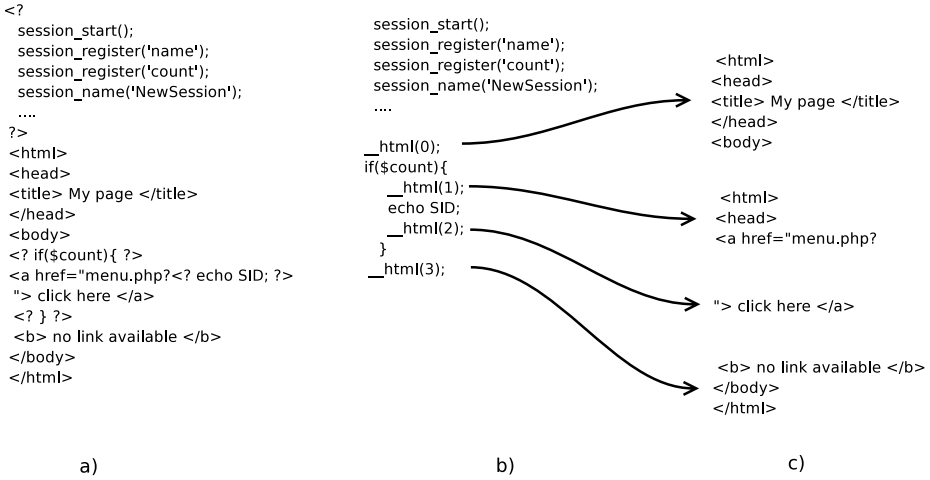

Figure 2. Example of page preprocessing.

be easily extended to encompass other kind of dynamic information.

Once instrumented, the WA is post-processed to allow to reconstruct the pages, i.e., to replace the dummy `__html(n)` instructions with the corresponding HTML block. The alternative would be to do it at run-time, however, we discarded this option to avoid a significant performance degradation. Finally, PHP modules, for encapsulating and supporting the trace collection, are added to the instrumented WA.

3.4 Data repository

As shown in Figure 1, the data layer contains both static and dynamic information. The static model is quite similar to the Conallen model shown in [12], thus detailing it is out of scope of this paper. The dynamic model is shown in Figure 3.

A WA execution trace (or simply a *trace*) is composed of a sequence of navigated pages[4]. Each trace contains information such as the user that interacted with the WA, the beginning and ending timestamp, plus additional details and comments. Comments are optionally inserted by means of a utility provided with the tool. These comments are used to mark and identify the different interactions of different users. Such information is entered before beginning the interaction and when the interaction is ended (otherwise it is impossible to separate multiple interactions).

Each page can be specialized in a *HTML page* or in a *server page*, and it can have a self-association that indicates a *hyperlink*. A *HTML page* can contain client scripts (e.g., Javascript functions), forms, and any other element (for sake of simplicity, not shown in the diagram) as indicated in the Conallen's model.

As stated in the introduction, it is worth highlighting the difference between the information extracted from HTML pages during static analysis and the information extracted by the dynamic analysis. The former is constant and incomplete, due for example to the dynamic generation of forms, Javascript, etc. The latter better reflects what executed and rendered by the browser. Of course, it is time dependent, it varies for each interaction. Finally, the server pages are associated with all the *building blocks*, already mentioned, that are worth analyzing to better understand the WA architecture and behavior.

To avoid significant performance degradation during the execution of the instrumented WA, a fast DBMS, *MySQL rel. 3.23.36-2* has been chosen to implement the data repository. Some figures about performance are reported in Section 5.1.

3.5 Abstractor

The abstraction module works by querying the data layer; it summarizes the results of the queries creating diagrammatic representations. Queries are mostly conceived to detect the building blocks of the WA (to extract its architecture) and their temporal sequence of interaction (to extract sequence diagrams). Moreover, the interaction frequency (e.g., the frequency of exercising an association), the information exchanged between the entities (e.g., passed parameters, state values stored in session data, read and writing from/to files or database) are exploited to significantly enhance the UML representation providing additional, useful

[4]Of course with the possibility of accessing the same page multiple times.

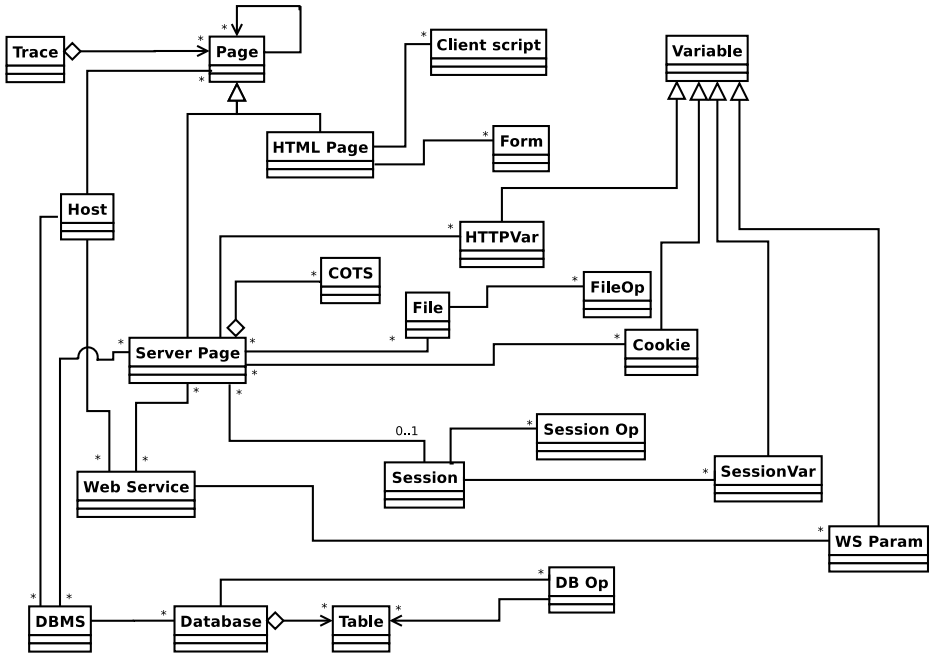

Figure 3. Model for representing dynamic information.

insights.

Finally, as highlighted at the beginning of this Section, we are not limited to the analyses described and currently implemented. Any other kind of analysis, such as clustering or spectral analysis, can be built relying on the extracted information.

3.6 Formatting and visualization

As mentioned above the abstracted diagrams are stored in XML, easing the transformation in any possible visualization format. At the time of writing, transformations are available for the *Dotty* format [23], and also the possibility of exporting diagrams to *Rational Rose*^{TM} is going to be implemented.

4. The Extracted Diagrams

This section details the heuristics adopted to recover UML diagrams from the WA, it also explains how Conallen diagrams are extended to annotate and store additional information. According to the UML standard, the possibility of extending the language relies on stereotypes and tagged values [24]. Recovered results are stored in XML, in a format consistent with the UML XML Metadata Interchange (XMI) Document Type Definition (DTD) [25].

4.1 Deployment and component diagrams

Deployment diagrams indicate the placement of resources constituting the WA on different peers. It is worth noting that the *host* information, associated to entities such as pages, web services and DBMS, aims exactly at identifying peers.

Once peers have been recognized, components such as pages, databases, Commercial-Off-The-Shelf (COTS) components and services are identified and added to each peer representation. Connections between peers are stereotyped with the access protocol, for instance the Simple Object Access Protocol (SOAP) for web services. Connections are tagged with the interaction frequency. In the opinion of the authors, this kind of information is useful for multiple purposes, for instance to identify bottlenecks in the WA architecture.

4.2 Sequence diagrams

Sequence diagrams highlight the interaction between the user and pages, and between pages and other entities. Timestamps allow reconstructing ordered sequences and, noticeably, to provide further useful information, such as statistics about the time spent by an user on a page. This, of course, may or may not be informative with regard to the performed activity, nevertheless it helps to indicate if the page is to be considered a point-of-passage or, instead, an

useful source of information. A sequence diagram involves the following participants:

- *Actors:* these are users or external systems interacting with the WA;
- *Boundary classes:* these are almost always pages containing forms, or HTML pages;
- *Control classes:* these are the server pages that, on their own, generate new *Boundary classes*, and coordinate the interaction with them;
- *Entity classes:* are, generally, the database tables, files, COTS or web services.

The interactions between pages are expressed in terms of links followed, HTTP variables and session variables passed. The interaction with databases is modeled in terms of connection operations, while operations on database tables are represented as queries. Finally, the interaction with web services is described detailing the invoked methods.

4.3 Class diagrams

Class diagrams are reverse engineered from the same information used to extract sequence diagrams. As for the previously described sequence diagrams, along with the distinction between boundary, control and entity classes, the diagrams distinguish the various Conallen's stereotypes. For each association, additional information are added as *tagged values*, namely:

- The frequency (during a navigation or during a given period of time) a link has been followed, a file or a database table was accessed (also specifying the frequency of each type of operation performed), a remote service was invoked;
- Data flow information, such as the variables passed between pages.

Also in this case, we believe that such additional information contributes to enhance the comprehension. While a static UML diagram (class, deployment, component) highlights relationships between classes, peers or components, frequencies show something more, the "strength" of the connection. Finally, the type of operation performed (e.g., database or file access, web service invocation) can help in detecting inconsistencies or potential security pitfalls (e.g., an actor writing to a table when this was not stated in the requirements), and also better comprehending the WA usage.

HTML Pages	22
PHP Pages	78
Database Tables	22
Text Files	1
Web Services	2

Table 1. Case study features.

5. Case Study

To assess the effectiveness of *WANDA*, we performed a dynamic analysis on a WA. At this point, our purpose was not to try analyzing a large commercial WA but, instead, to test the tool on a medium size application containing the main features (databases, text files, web services) we aimed to model.

The chosen WA is a tourism portal for the town of Benevento, developed by a team of fifth year engineering students. The system offers information about: monuments, places of interest, relevant events, public transportation. Moreover, it allows checking the availability of rooms in hotels, and eventually perform the booking. Accessory features, such as web site administration, guestbook and mailing list, are available. Finally, web services are used to provide some real-time information, such as the current temperature, etc.

The system has been developed using PHP and HTML, and relies on a PostgreSQL database. Table 1 reports the main characteristics of the WA.

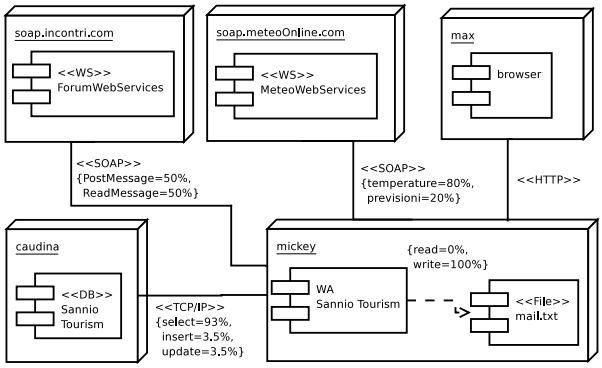

Figure 4. Deployment diagram.

5.1 Case Study Results

The above described WA was instrumented and then, to collect traces, exercised in all the features described in Section 3.3. In the current work this was done by letting a user interacting with the WA for a some hours, trying to discover all the WA features.

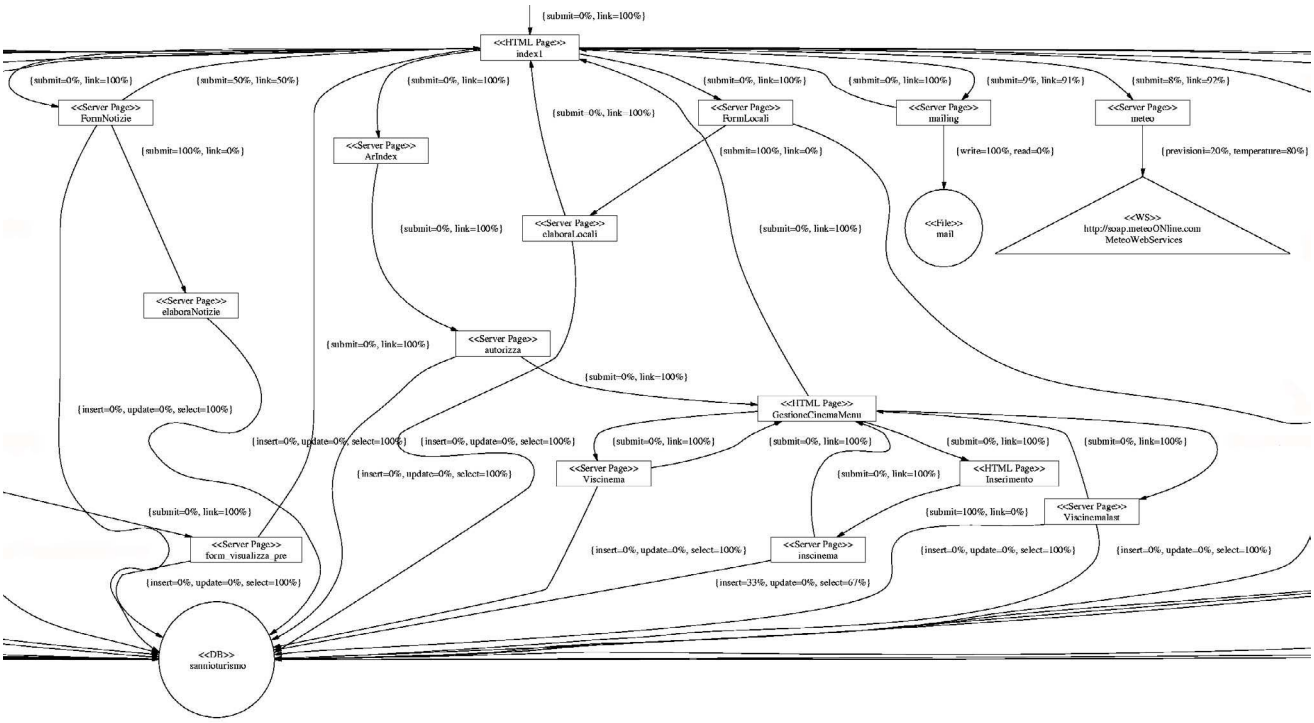

Figure 5. Class diagram with frequency information.

Figure 4 depicts the deployment diagram extracted from the WA. The figure highlights the peers constituting the WA: the web server (*mickey*, which contains the WA and a text file for storing mailing list information) the client (*max*), the DBMS server (*caudina*) and the two hosts on which the invoked web services have been deployed. As shown, tagged values indicate statistics on operations performed on the database, on the remote methods invoked, and on the file operations.

Figure 5 shows one of the many possible views of the class diagram. In particular, stereotypes are used to distinguish server pages, HTML pages, web services (represented as triangles), files and databases (represented as circles). Similarly to the deployment diagram, tagged values indicate, as frequencies, the type of operations performed on files and databases, the methods invoked on the web service, and the way a page passed information or the control to another (i.e. *link* or *submit*). A second type of visualization is shown in Figure 6. Along with the stereotype of the association (e.g., *DB Connect*), the diagram shows, as tagged values, the list of the variables passed between two pages, thus giving an immediate representation of data-flow information.

Finally, Figure 7 represents XML encoded information of an interaction scenario. Such information can be used to recover sequence diagrams. As shown, the scenario involves an actor (identified by a session id), a boundary class (i.e. a HTML page), two control classes (PHP scripts) and an entity (a web service). Each interaction is identified by a timestamp, allowing to reconstruct the temporal sequence.

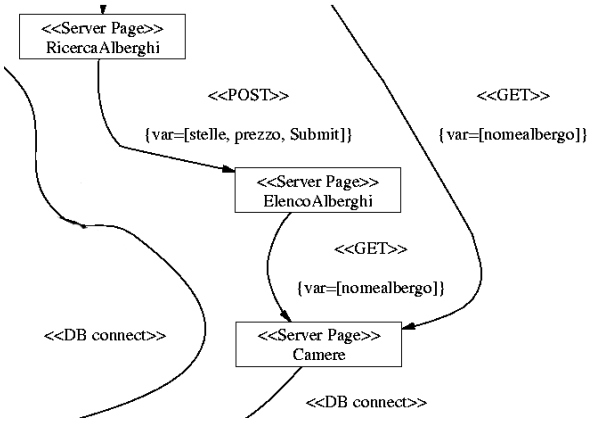

Figure 6. Class diagram: passed variables as tagged values.

```
<?xml version="1.0"?>
<!DOCTYPE sequence SYSTEM "sequence.dtd">

<sequence>
<user>
<session id="ca292115d183abbdea7cfeba459676a6"/>
</user>

<interaction timestamp="1075329599.8232" type="link">
<source name="/ing2/Progetto/index1.html" type="HTML" category="boundary"/>
<target name="/ing2/Progetto/meteo.php" type="Server" category="control"/>
</interaction>

<interaction timestamp="1075331924.8738" type="link">
<source name="/ing2/Progetto/index1.html" type="HTML" category="boundary"/>
<target name="/ing2/Progetto/forum.php" type="Server" category="control"/>
</interaction>

<connection timestamp="1075331196.0831">
<source name="/ing2/Progetto/meteo.php" type="Server" category="control"/>
<target name="http://soap.meteoONline.com/Benevento/MeteoWebServices.wsdl"
              type="WS" category="entity">
<wsfunction name="temperature"/>
</target> </connection>
</sequence>
```

Figure 7. XML representation of an interaction scenario.

5.2 Performance Issues

As highlighted in Section 3, the instrumentation of an application is likely to deteriorate its performance. This may have a negative impact on the interactiveness of the WA. To assess the feasibility of instrumentation and program comprehension tasks with real users, we measured the overhead introduced by the dynamic analysis infrastructure.

The DBMS for storing the traces was installed on the same host as the WA, and response time was measured (using the Unix utility *time*) by repeatedly downloading pages (in particular, pages heavily involving database access tasks) with the *wget* utility. To allow server pages executing their task and generating client pages according to the received parameters, such pages were called by passing the parameters on the URL according to the GET method. Experiments were repeated 30 times and then average values computed. Measures were performed on an Intel Celeron 1.8 GHz, with 256 Mbytes of RAM, Linux Mandrake 8.0 operating system and Apache Web Server. As shown in Table 2, performance degradation is, even if present, acceptable for allowing an *in field* collection of traces.

URL	Instr. [msec.]	Not Instr. [msec.]	Degr. [%]
elaboraSpettacoli.php?ID=5	9.3	7.0	33%
elaborCinema.php?ID=0	9.0	8.6	5%
Alberghi/Camere.php?nomealbergo=Genova	11	9.3	18%
Users/utenti.php?Dati+non+corretti	10	7.3	37%

Table 2. Average performance degradation of the instrumented WA.

6. Conclusions

This paper proposed an approach and a tool to perform dynamic analysis of WAs. HTML pages are instrumented to collect different kinds of information including access to files, databases, web services and COTS. A model to represent WA dynamic information has been conceived and adopted. To promote an immediate visualization of dynamic information, UML diagrams have been extended with information of usage frequency. Further, diagrams have been XML-encoded to facilitate information exchange with other tools.

The infrastructure proposed has been conceived for a more effective extraction of UML diagrams from WAs, and for an enrichment of the information shown, adding frequency information to the diagrams. Moreover, the extracted dynamic information can also be exploited for other purposes, such as restructuring or, in general, for profiling. Finally, the modular structure allows to easily add parsers, extract additional feature and enhance the abstractor engine.

The case study demonstrates the applicability, the effectiveness and the performance of the tool, and also gives an idea about the usefulness of the enriched diagrams. Work-in-progress is devoted to investigate more sophisticated analyses of the extracted dynamic information, and to further validate the tool on larger WAs. To better perform these analyses, we intend to collect WA execution traces during a long period of real use of some WAs.

References

[1] A. Hassan and R. Holt, "Architecture recovery of web applications," in *Proceedings of the International Conference on Software Engineering*, (Orlando, FL, USA), pp. 349–359, May 2002.

[2] D. Heuzeroth, T. Holl, G. Högström, and W. Löwe, "Automatic design pattern detection," in *Proceedings of the IEEE International Workshop on Program Comprehension*, (Portland, OR, USA), pp. 94–103, May 2003.

[3] T. Systä, "On the relationships between static and dynamic models in reverse engineering Java software," in *Proceedings of IEEE Working Conference on Reverse Engineering*, (Atlanta, GA, USA), Oct 1999.

[4] L. Briand, Y. Labiche, and Y. Miao, "Towards the reverse engineering of UML sequence diagrams," in *Proceedings of IEEE Working Conference on Reverse*

Engineering, (Victoria, BC, Canada), pp. 57–66, Nov 2003.

[5] P. Tonella and F. Ricca, "Dynamic model extraction and statistical analysis of web applications," in *Proceedings of International Workshop on Web Site Evolution*, (Montréal, QC, Canada), pp. 43–52, Oct 2002.

[6] G. Antoniol, G. Canfora, A. Cimitile, and A. De Lucia, "Web sites: Files, programs or databases?," in *Proceedings of International Workshop on Web Site Evolution*, (Atlanta,GA, USA), Oct 1999.

[7] J. Conallen, *Building Web Applications with UML (2nd Edition)*. Addison-Wesley Publishing Company, 2002.

[8] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia, "Web site reengineering using RMM," in *Proceedings of International Workshop on Web Site Evolution*, (Zürich, Switzerland), Mar 2000.

[9] F. Ricca and P. Tonella, "Web site analysis: Structure and evolution," in *Proceedings of IEEE International Conference on Software Maintenance*, (San Jose, CA, USA), pp. 76–85, Oct 2000.

[10] F. Ricca and P. Tonella, "Analysis and testing of web applications," in *Proceedings of the International Conference on Software Engineering*, (Toronto, ON, Canada), pp. 25–34, IEEE Computer Society Press, May 2001.

[11] F. Ricca and P. Tonella, "Understanding and restructuring web sites with ReWeb," *IEEE Multimedia*, vol. 8, pp. 40–51, Apr-Jun 2001.

[12] G. Di Lucca, M. Di Penta, G. Antoniol, and G. Casazza, "An approach for reverse engineering of web-based applications," in *Proceedings of IEEE Working Conference on Reverse Engineering*, (Stuttgart, Germany), pp. 231–240, Oct 2001.

[13] G. Di Lucca, A. Fasolino, F. Pace, P. Tramontana, and U. De Carlini, "WARE: A tool for the reverse engineering of web applications," in *Proceedings of European Conference on Software Maintenance and Reengineering*, (Budapest, Hungary), pp. 241–250, Mar 2002.

[14] G. Di Lucca, A. R. Fasolino, and P. Tramontana, "Towards a better comprehensibility of web applications: Lessons learned from reverse engineering experiments," in *Proceedings of International Workshop on Web Site Evolution*, (Montréal, QC, Canada), pp. 33–42, IEEE Computer Society Press, Oct 2002.

[15] G. Di Lucca, A. Fasolino, P. Tramontana, and U. De Carlini, "Abstracting business level UML diagrams from web applications," in *Proceedings of International Workshop on Web Site Evolution*, (Amsterdam, The Netherlands), pp. 12–19, Oct 2003.

[16] G. Di Lucca, A. Fasolino, P. Tramontana, and U. De Carlini, "Recovering a business object model from web applications," in *Proceedings of Annual Conference on Computer and Software Applications*, (Dallas, TX, USA), pp. 348–353, Nov 2003.

[17] C. Boldyreff and R. Kewish, "Reverse engineering to achieve maintainable WWW sites," in *Proceedings of IEEE Working Conference on Reverse Engineering*, (Stuttgart, Germany), pp. 249–258, Oct 2001.

[18] J. Vanderdonckt, L. Bouillon, and N. Souchon, "Flexible reverse engineering of web pages with Vaquista," in *Proceedings of IEEE Working Conference on Reverse Engineering*, (Stuttgart, Germany), pp. 241–248, Oct 2001.

[19] S. Tilley and S. Huang, "Evaluating the reverse engineering capabilities of web tools for understanding site content and structure: A case study," in *Proceedings of the International Conference on Software Engineering*, (Toronto, ON, Canada), pp. 514–523, May 2001.

[20] E. Stroulia, J. Thomson, and G. Situ, "Constructing XML-speaking wrappers for WEB applications: Towards an interoperating WEB," in *Proceedings of IEEE Working Conference on Reverse Engineering*, (Brisbane, Queensland, Australia), pp. 59–68, Nov 2000.

[21] L. Moonen, "Generating robust parsers using island grammars," in *Proceedings of IEEE Working Conference on Reverse Engineering*, pp. 13–22, Oct 2001.

[22] "JavaCC Home." `https://javacc.dev.java.net/`.

[23] E. Koutsofios, "Editing graphs with *dotty*," technical report, AT&T Bell Laboratories, Murray Hill, NJ, USA, July 1994.

[24] G. Booch, I. Jacobson, and J. Rumbaugh, *The Unified Modeling Language User Guide*. Addison-Wesley Publishing Company, 1998.

[25] "XML Metadata Interchange specification version 2.0, OMG group." `http://www.omg.org/docs/formal/03-05-02.pdf`.