Towards a Better Understanding of Web Applications

Ahmed E. Hassan and Richard C. Holt
Software Architecture Group (SWAG)
Department of Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1
CANADA
{aeehassa, holt}@plg.uwaterloo.ca

ABSTRACT
This paper presents a framework to recover the architecture of web applications. Developers can visualize and navigate the recovered architecture. Furthermore, they can analyze the architecture to gain a better understanding of their web application.

Keywords
Web Applications, Software Architecture, Architecture Recovery, Reverse Engineering

1 INTRODUCTION

The web browser's ubiquitous and simple interface has opened the door for the development of many new distributed applications. At the start, web pages were simple static HTML pages linked together. The web pages served well their purpose of providing easy and open access to information across the world. Subsequently new technologies such as Java and Java Script Pages were introduced. These have facilitated the development of large-scale applications, which harness the power of the web.

Too often, software-engineering principles are not applied to the development of these web applications. As Pressman notes, the reluctance of web developers to adopt well-proven principles is worrisome [12]. The techniques used nowadays by web application developers are similar to the ad hoc ones used by their predecessors in the 1960s and 1970s.

As web applications evolve and their complexity increases, ad hoc methods won't suffice. Developers need a better understanding of their software system when they join a new team or maintain old code. Lack of documentation[1] and the original developers[2] increases the cost and time needed to maintain large web applications.

Reverse engineering and software visualization have been proposed as techniques to improve the understanding of large traditional non-web applications. Research [5] has demonstrated the use of semi-automated techniques and tools to recover the design of large applications. The Portable Bookshelf (PBS) environment [10] combines much of the knowledge and techniques developed over the last decade in program understanding. It has been used to recover the design of large applications such as Linux (800 KLOC[3]) [2], Apache (80 KLOC) [4], and Mozilla (2.1 MLOC[4]) [9]. This paper describes the reuse and extension of the capabilities of PBS to support the design recovery of web applications. We developed a set of tools capable of parsing and extracting relations between the components of web applications. Also, we modified PBS's visualizer to handle the heterogenous nature of web applications.

2 THE COMPONENTS OF A WEB APPLICATION

In a web application, many components are inter-linked together to implement its functionality. The following components exist in web applications: web browsers (used by the clients); web servers; web pages; application servers; application pages[5]; databases; distributed objects[6] such as CORBA, EJB and COM; and multimedia web objects such as images, videos, and etc.

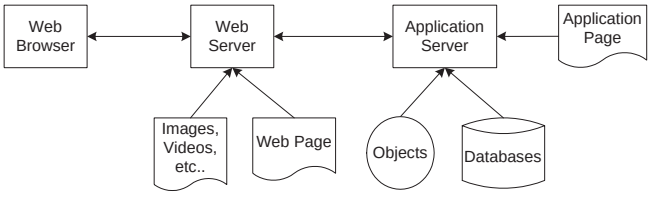

Figure 1: Dataflow Between the Components of a Web Application

Figure 1 shows the data flow between the various com-

[1]Documentation associated with a web application generally does not exist and if it does, it is rarely complete or up-to-date.

[2]The web community has a high turnover rate: the average employment length is just over one year [8].

[3]KLOC: Thousand Lines of Code.

[4]MLOC: Million Lines of Code.

[5]Application pages are web pages which contain executable code that is executed inside the application server before the page is transmitted back to the requesting browser.

[6]These object are not objects in the sense of source code object-oriented programming objects such as those defined in C++ or Java. Instead, they are pieces of compiled code that provide a service to the rest of the software system through a defined interface.

ponents of a web application. The user of the application employs the web browser as the interface to gain access to the functionality of the web application. The user interacts with the browser by clicking on links and filling-in form fields. The browser in turn transmits the user's actions to the web server. Requests are sent using the HTTP protocol. Upon receiving the request, the web server determines if it can fulfill the request directly or if the application server must be invoked. The application server and the web server may reside on the same machine or on different machines. Many web servers and application servers can serve requests for a single application. The web server can serve HTML pages and multimedia content such as images, videos, or audio files; or it can forward the request to the application server. The application server processes the application page and returns an HTML page to the web server. Finally, the web server returns the requested page to the web browser, which displays it to the user.

3 VISUALIZATION NEEDS OF DEVELOPERS

Traditional software architecture diagrams show the various modules and source files that compose the software system and their interactions. Web applications are composed of many components and each component may have its own internal architecture or design. A web application developer is particularly concerned with the topology of the components. For example, when studying the architecture of web applications, the internal architecture of the web server and the web browser are not shown as they add more complexity to the visualized system and do not contribute to the overall understanding of the software system. The web server and the browser represent infrastructure systems similar to the operating system and the windowing system whose architectures are not shown when visualizing traditional software systems. The architecture diagrams for web applications need to show the distributed objects, database tables, multimedia objects (such as movies, pictures and audio files) that are scripted together to implement large sophisticated web applications.

4 ARCHITECTURE RECOVERY OF WEB APPLICATIONS

We use a semi-automated process to recover the architecture of web applications. The process uses tools/extractors to examine the source code of the application. The tools' output is combined with input from a system expert to produce architecture diagrams. We extend the Portable BookShelf (PBS) [11, 3, 10] environment to address the differences between web applications and traditional software applications.

Figure 2 shows the process of creating an architecture document using the PBS environment. First the artifacts of the software system (such as source code, documentation, execution traces, web pages, etc.) are processed using specialized extractors. The extractors automatically generate facts about the software system based on these artifacts. The facts could be detailed such as: *function "f" uses variable "a"* or at a higher level such as: *file "f1" uses file "f2"*. The level of detail of the extracted facts depends on the extractor and the level of analysis that is to be performed on the recovered facts. For example, for architecture level analysis, facts at the function level are not needed and can be lifted to a higher level. The generated facts are stored in TA format [6, 7]. Figure 3 gives more detail about how this extraction takes place for web applications.

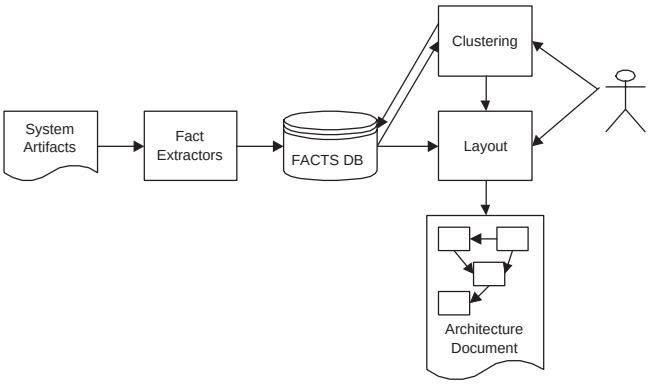

Figure 2: Using the Portable BookShelf to Create an Architecture Document

To reduce the complexity of the generated diagrams, the software system is decomposed into meaningful subsystems using clustering techniques. The clustering is performed automatically first by a tool that proposes decompositions based on several heuristics such file naming conventions, development team structure, directory structure, or software metrics [13, 14, 1]. The developer later manually refines the automatically proposed clustering using their domain knowledge and available system documentation. The decomposition information along with the extracted facts is stored in TA format.

Finally, an automatic layout tool processes the stored facts to generate diagrams such as the one shown in Figure 6. The layout tool attempts to minimize the line crossing in the generated architecture diagrams. The developer may choose to modify the generated layout.

The aforementioned process combines tool support and human interpretation to recover the architecture. Tool support dramatically reduces the recovery time, and human interpretation is paramount in organizing and clustering the large amount of extracted information.

Extractors for Web Applications

Web applications are developed using a variety of lan-

guages and are formed of components for which the source code may not be available or an appropriate extractor may not exist. The properties of web applications present many challenges for traditional software architecture recovery frameworks that usually depend on a single extractor. Our approach uses a set of extractors that cooperate to emit the facts from the examined web application. Figure 3 shows an overview of the various extractors and their input and the type of facts generated by them. We use five types of extractors: an HTML extractor, a Server Script extractor, a DB Access extractor, a Language extractor, and a Binary extractor. Each extractor is responsible for examining a component or a section of a component and generating the appropriate facts.

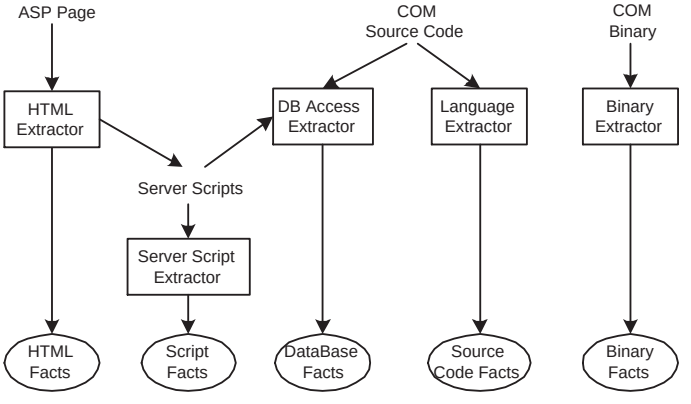

Figure 3: Conceptual Architecture of the Fact Extractors

Once all the facts are emitted, the clustering information is combined and all the data (facts and clustering) are processed to reduce their complexity. Then a layout tool prepares the facts for display by a visualizer.

The various extractors are invoked by a shell script which crawls the directory tree of the source code for the web application. The script determines the type of the component and invokes the corresponding extractor. For example, if the script determines that a file is a binary file, the Binary Extractor is invoked. Each extractor generates a set of facts and stores its results in a file with the same name as the input file and the name of the extractor as the suffix. Later, another script crawls the previously processed directories and consolidates all the generated files into a single file called `THEFACTS`. The `THEFACTS` file is combined with the clustering information which decomposes the web application into subsystems.

5 CASE STUDY

We will now present the architecture of a web application. The architecture was extracted using the process described in the previous section.

The Hopper News Web Application

Hopper News is the web site of a fictitious newspaper. It features sections for local, national, international news, sports, weather, and classified advertisements. In this sample application, only the classified advertisement section is implemented. The rest of the links on the main web page, shown in Figure 4, are not functional. The only functional link is the lowest link in the page which links to the classified advertisement page, shown in Figure 5.

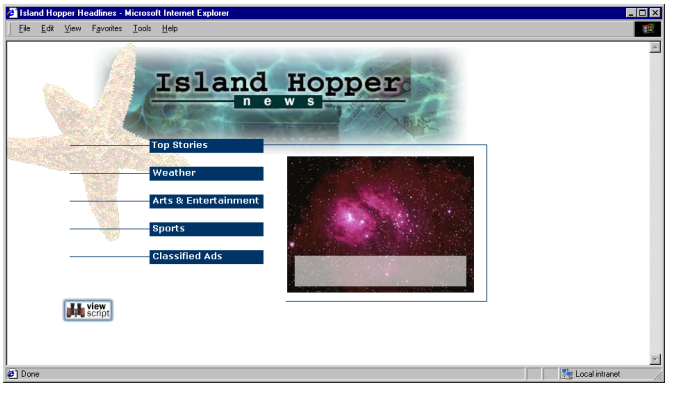

Figure 4: Main Page for Hopper News Web Application

The application provides two main functionalities: It permits a person to browse advertisements, and it permits registered users to place advertisements. To place an advertisement, the user must pay a fee.

Figure 5: Hopper News Classified Advertisement Web Page

The Recovered Architecture for Hopper News

Using our modified PBS framework, we recovered the architecture, as shown in figure 6. This figure is a screenshot from our viewer. The facts have been clustered according to the main business concepts in the application: *Browse advertisements*, *Place advertisements*, *User*, *Advertisement*, *Product*, and *Payment*.

The `UTIL`, `SYSTEMLIBS`, and `ASPOBJECTS` subsystems are

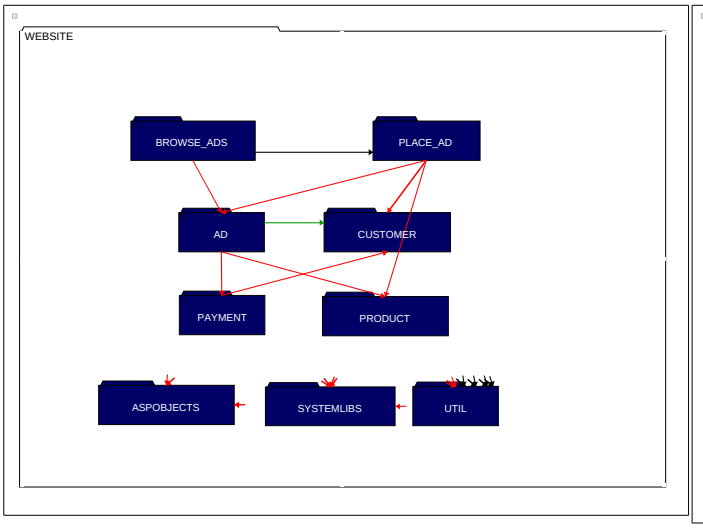

Figure 6: Recovered Architecture for Hopper News

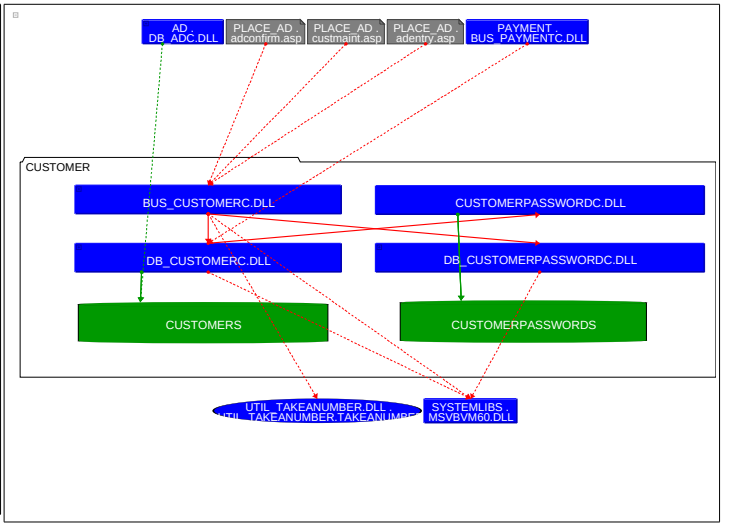

Figure 7: Hopper News Customer Subsystem

used extensively by all other subsystems in the application. Arrows to these subsystem have been truncated to ensure the clarity of the diagram.

The viewer presents the recovered architecture and enables developers to analyze the architecture by using interactive queries such as "*What are all subsystems that use the Customer subsystem?*". The viewer shows different relations between the different entities. We use three different colors to group the relations into dependency types. Black arrows indicate a hyperlink dependency between two entities, red arrows indicate a control dependency and green arrows indicate a data dependency. When the cursor is positioned on a specific arrow, a detailed description of the relation is displayed.

Color and icon shape is used to represent the various entities. Blue folders represent subsystems, blue ovals represent COM objects, grey pages represent Active Server Pages, blue boxes represent DLLs[7], and green cylinders represent database tables.

The viewer permits us to delve into the structure of every subsystem. We double-click on the subsystem of interest and the viewer opens a new diagram showing the internals of the subsystem. Figure 7 shows the internal structure of the `CUSTOMER` subsystem. It shows the various database tables that are part of the `CUSTOMER` subsystem and visualizes their interaction with the `DLL`s that encode the business rules of the web application. The viewer also shows how the internal components of the `CUSTOMER` subsystem interact with the components that are not part of it.

6 CONCLUSION

We have shown that current work in software engineering can be adapted and modified to reverse engineer web applications. We have developed tools to recover the architecture of web applications. We have shown how to visualize the recovered architecture as simple box and arrow diagrams. This visualization helps the developers of web application understand the application's architecture, allowing them to more rapidly and confidently update the application to handle new requirements.

[7]Dynamic Link Library.

ACKNOWLEDGEMENTS

To validate our framework, we used web applications provided by Microsoft Inc. and Sun Microsystems of Canada Inc. In particular, we would like to thank Wai-Ming Wong from Sun for his assistance in our analysis of the web applications.

REFERENCES

[1] I. T. Bowman. Architecture Recovery for Object Oriented Systems. Master's thesis, University of Waterloo, 1999.

[2] I. T. Bowman, R. C. Holt, and N. V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. In *IEEE 21st International Conference on Software Engineering*, Los Angeles, USA, May 1999.

[3] P. J. Finnigan, R. C. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. A. Müller, J. Mylopoulos, S. G. Perelgut, M. Stanley, and K. Wong. The software bookshelf. *IBM Systems Journal*, 36(4):564–593, 1997. Available online at `<http://www.almaden.ibm.com/journal/sj/364/finnigan.html>`

[4] A. E. Hassan and R. C. Holt. A Reference Architecture for Web Servers. In *7th Working Con-*

ference on Reverse Engineering, Brisbane, Queensland, Australia, Nov. 2000.

[5] Hausi A. Müller, Jen H. Jahnke, Dennis B. Smith, Margaret-Ann Storey, Scott R. Tilley and Kenny Wong. Reverse Engineering: A Roadmap. In *Proceedings of Future of Software Engineering*, Limerick, Ireland, June 2000.

[6] R. C. Holt. *An Introduction to TA: the Tuple-Attribute Language*, Mar. 1997. Available online at <http://plg.uwaterloo.ca/~holt/papers/ta.html>

[7] R. C. Holt. Structural manipulations of software architecture using Tarski relational algebra. In *Proceedings of WCRE'98*, Oct. 1998.

[8] R. Konrad. Tech employees jumping jobs faster, 2000. Available online at <http://news.cnet.com/news/0-1007-202-2077961.html>

[9] E. H. S. Lee. Analyzing Mozilla, 2000. Available online at <http://plg.uwaterloo.ca/~ehslee/pub/mozilla.ppt>

[10] The Portable Bookshelf (PBS). Available online at <http://www.turing.toronto.edu/pbs>

[11] D. A. Penny. *The Software Landscape: A Visual Formalism for Programming-in-the-Large*. PhD thesis, University of Toronto, 1992.

[12] R. S. Pressman. What a Tangled Web We Weave. *IEEE Software*, 17(1):18–21, Jan. 2000.

[13] V. Tzerpos and R. C. Holt. A Hybrid Process for Recovering Software Architecture. In *Proceedings of CASCON '96*, Toronto, Canada, Nov. 1996.

[14] V. Tzerpos and R. C. Holt. Software botryology: Automatic clustering of software systems. In *Proceedings of the International Workshop on Large-Scale Software Composition*, 1998.