Architecture Recovery of Web Applications

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

ABSTRACT
Web applications are the legacy software of the future. Developed under tight schedules, with high employee turn over, and in a rapidly evolving environment, these systems are often poorly structured and poorly documented. Maintaining such systems is problematic.

This paper presents an approach to recover the architecture of such systems, in order to make maintenance more manageable. Our lightweight approach is flexible and retargetable to the various technologies that are used in developing web applications. The approach extracts the structure of dynamic web applications and shows the interaction between their various components such as databases, distributed objects, and web pages. The recovery process uses a set of specialized extractors to analyze the source code and binaries of web applications. The extracted data is manipulated to reduce the complexity of the architectural diagrams. Developers can use the extracted architecture to gain a better understanding of web applications and to assist in their maintenance.

1 INTRODUCTION

A web application is a software system whose functionality is delivered through the web [9]. With the advent of the Internet and the web, many applications are no longer developed using traditional client/server technologies. Instead, new applications are developed using web technologies such as web browsers, web servers, and application servers. A web browser is the user's interface to the application. Internet protocols such as Hyper Text Transfer Protocol (HTTP) are used for communicating between the interface and the rest of the application.

Originally developed as a document-sharing platform, the web is still often considered as such. Consequently, the development of web applications is considered to be an authoring problem and not a software engineering problem. For example, many of commercial and research tools used to analyze the structure of a web application show only the hyperlink relations between the different web pages. These tools fail to show the interaction between the databases, the distributed objects, and the web pages that form a web application [32]. Such relations are important to software developers who must maintain or enhance these applications. For example given a database table, a developer may need to answer questions such as "*Which web page writes data to this table?*", or "*Which object reads data from this table?*". Furthermore, a software architect may need to determine the answer for questions such as "*Does my application have a three layered architecture?*", "*Which part of my application is affected if the* `CUSTOMERSPASSWORDS` *data table is offline for maintenance?*". Currently, such inquiries can only be answered by scanning the source code for answers using tools such as *grep*, consulting documentation, or asking senior developers.

Unfortunately, the documentation associated with a web application does not commonly exist and if it does, it is rarely complete or up-to-date. With a very short development cycle, software-engineering principles are rarely applied to the development of web applications. As Pressman notes, the reluctance of web developers to adopt well-proven principles is worrisome [25]. The techniques used by web application developers are similar to the ad hoc ones used by their predecessors in the 1960s and 1970s. To aggravate matters, the web community has a high turnover rate with an average employment length that is just over one year [19]. The original developers of a maintained web application are often no longer part of the organization. This lack of documentation and system experts increases the cost and time needed to understand and maintain large web applications.

Recent research [1, 2, 5, 6, 13, 26, 33] recognizes the need to adapt traditional software engineering principles to assist in the development of web applications. Reverse engineering and software visualization have been

proposed as techniques to improve the understanding of large traditional non-web applications [14]. These techniques have been used to study systems such as as the Linux Kernel (800 KLOC) [4], the Apache web server (80 KLOC) [12], and the Mozilla web browser (2.1 MLOC) [20].

In this paper, we describe an approach to assist developers in understanding their web applications. We describe a set of tools which parse and extract relations between the various components of a web application. The extracted components and relations are visualized using a specialized viewer. In this paper, we consider this visualization to characterize the software architecture of the system. It allows developers to perform impact analysis on the extracted relations between the components. The recovered architectures for a number of web applications were shown to maintenance engineers who reported that the produced diagrams are useful in assisting them in understanding the systems. In many cases, the engineers had produced similar diagrams by manually examining the source code of various components. In short, they would have used the tool, if it had been available to them.

Paper Organization

The rest of this paper is organized as follows. Section 2 describes the interactions between the various components in a web application. Section 3 describes the types of information that are needed by developers to gain a better understanding of web applications. Section 4 presents a sample web application and its recovered architecture. Section 5 describes our architecture recovery approach for web applications. Section 6 delves deeper into the presented architecture recovery process and describes the *schemas* used to analyze the recovered architecture before visualizing it. Section 7 describes related work. Finally, Section 8 draws conclusions from this work.

2 COMPONENTS OF A WEB APPLICATION

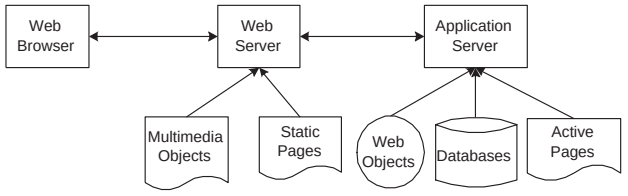

Figure 1: Dataflow Between the Components of a Web Application

Web applications contain many components that are linked together to deliver the functionality of the application (see figure 1). These components are written in various programming languages and run on multiple machines distributed across the network. Each component is written in a language which is appropriate to implement its functionality. Scripting languages are used to assemble the different components, and databases are used by the components to store and share their data.

From our experience in studying and analyzing web applications, we observe that there is a recognizable set of components which comprise these system. We believe that this set is a useful basis for analyzing these applications. This set consists of web browsers (used by the clients), web servers, application servers and the following components:

Static pages These contain only HTML code and executable code that runs on the web browser. They are served by the web server and do not need to be preprocessed by the application server.

Active pages such as Active Server Pages and Java Server Pages. These pages contain a mixture of HTML tags and executable code. When an active page is requested, the application server preprocesses it and integrates data from various resources such as web objects or databases, to generate the final HTML web page sent to the browser. Table 1 shows two active pages: *welcome*1.*asp* and *welcome*2.*asp* which are equivalent to the static page *welcome.html*. All pages say "*Welcome to CNN.COM*" when displayed in the user's browser. The symbols `<%` and `%>` indicate to the application server that the text between them is control code which is to be executed and the output returned to requesting client. For example, in *welcome*2.*asp* the application server needs to locate the *Server* object and execute the *getName()* method, then *Write* the output to the file returned to the browser.

Static Page *welcome.html*	Active Page *welcome1.asp*	Active Page *welcome2.asp*

`<html>`
`Welcome to`
`CNN.COM`
`</html>`
	`<html>`
	`Welcome to`
	`<% Write("CNN.COM") %>`
	`</html>`
		`<html>`
		`Welcome to`
		`<% Write(Server.getName()) %>`
		`</html>`

Table 1: Examples of Active Pages

Web objects These are pieces of compiled code which provide a service to the rest of the software system through a defined interface. They are supported by distributed technologies such as CORBA, EJB and DCOM. They are not objects in the sense of source code object-oriented programming objects such as those defined in C++ or Java.

Multimedia objects such as images, and videos.

Databases These are used to store data that is shared among the various components.

Figure 1 shows the data flow between the various components of a web application. The user of the application employs the web browser to access the functionality of the web application. The user interacts with the browser by clicking on links and filling 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 web server serves directly static HTML pages and multimedia content such as images, videos, or audio files; or it forwards the request to the application server. The application server processes active pages and returns the result to the web server as static HTML pages. The web server in turn returns the HTML page back to the requesting web browser, which displays it to the user.

3 VISUALIZATION NEEDS OF DEVELOPERS

Web Applications are large complex software systems that contain a rich structure with many interesting relations between their components. The choice of the appropriate relations and components is task dependent. For example, a developer modifying an object used by many active pages is interested in a different set of relations than a database administrator trying to track all the components using a specific database table. Unfortunately, the only relations shown by current web application visualization tools are the hyperlinks between the static pages in a web application. This limited visualization is not sufficient for a developer to understand the structure of the web application and the interactions between its various components.

Previous studies in program maintenance and understanding conducted on the development of traditional software systems [22, 27, 28, 29] assisted us in defining a set of useful relations and components to recover and display in generated diagrams. Studies have shown that developers use visualization tools to pinpoint locations of interest in the system's code (for example to locate all the files that use a database table); they delve into these locations of interest to improve their understanding using their code editors. In response, our recovered architecture diagrams do not show all the detailed relations and components, instead they show an overview of the system. For example, we do not show the internal structure of code inside the same file. We show the inter-file relations. Furthermore, we use techniques such as containment and information hiding to reduce the complexity of the visualized systems.

In a web applications each component has its own internal architecture or design. The web application developer is particularly concerned with the topology of the components: how they interact together and how they are glued together using the various web technologies. On the other hand, the web application developer is not usually interested in the internal architecture of the web server or the web browser, as they add complexity to the visualized system without contributing to the overall understanding of the system. The web server and the browser represent software infrastructure 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 should show the main components of a web application, such as web objects, database tables, multimedia objects that are glued together to implement large sophisticated web applications.

When developers search for a better understanding of a system, the four most common search targets used by developers are function definitions, uses of a function, variable definition, and all uses of a variable [28]. We adapted analogous relations for web applications, but at a higher level of abstraction. Instead of showing variables, functions and their interrelations, we show database tables, distributed objects, and their interrelations.

4 AN EXAMPLE WEB APPLICATION

We have recovered the architecture of several large commercial and experimental web applications. These applications had over 200 distributed web objects and over 15 databases per application. Space restrictions do not allow us to present the architecture of such large systems, so we will illustrate our approach to architecture extraction in terms of a small example system, called Hopper News.

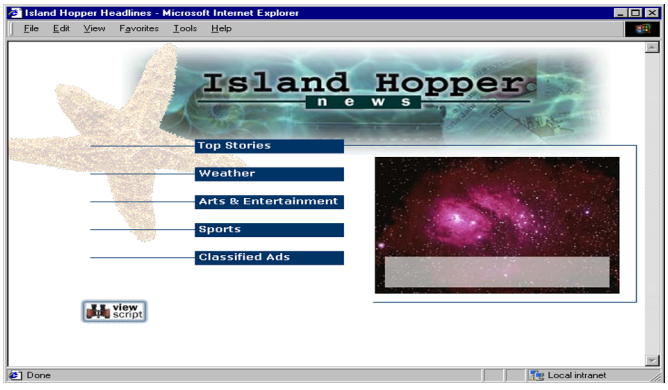

Figure 2: Main Page for Hopper News Web Application

Hopper News is the web site for a fictitious newspaper. It features sections for local, national, international news, sports, weather, and classified advertise-

ments. In this sample application, only the classified advertisement section is implemented. The rest of the links on the main web page, shown in Figure 2, are not functional. The only functional link is the lowest button in the page which links to the classified advertisement page, shown in Figure 3. The application is developed for the Microsoft Windows platform and contains components written in HTML, VBScript, VB, and C++. The application is developed by Microsoft to showcase building web application on the Windows platform.

Figure 3: Hopper News Classified Advertisement Web Page

The application supports two main functions: 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.

The Recovered Architecture for Hopper News
Using our approach, we recovered the architecture, as shown in figure 4. The architecture of Hopper News has the following layers:

Layer 1 consists of the Presentation Logic. It provides the user interface to the two functions of the application: browsing and placing advertisements.

Layer 2 consists of the Business Logic and Database. It encapsulates the business rules based on the entities in the application domain (such as Customer, Ad, and Product). For example, in figure 4 we see that the `Place Ad` subsystem does not interact with the `Payment` subsystem. The information about the cost of placing an advertisement is encapsulated in the `Ad` subsystem. Such information is considered to be a business rule. Also this layer provides persistent storage (Databases) for the business data.

Layer 3 provides infrastructure to support the basic functions needed by the upper layers, such as access to the local file system, or string manipulation functions. As this web application is developed on the Microsoft Windows platform, the *Infrastructure* layer contains Microsoft Windows specific components such as Active Server Pages Objects, Windows libraries and general utilities.

The architecture of Hopper News can be considered as a three-tiered architecture as follows:

Tier 1 is the Presentation Logic, which is the same as layer 1.
Tier 2 is the Business Logic, which is part of layer 2.
Tier 3 is the Database, which is the rest of layer 2.

The parts of the Database tier are localized according to the Business Logic they support, so Figure 4 combines these two tiers into a single layer (layer 2).

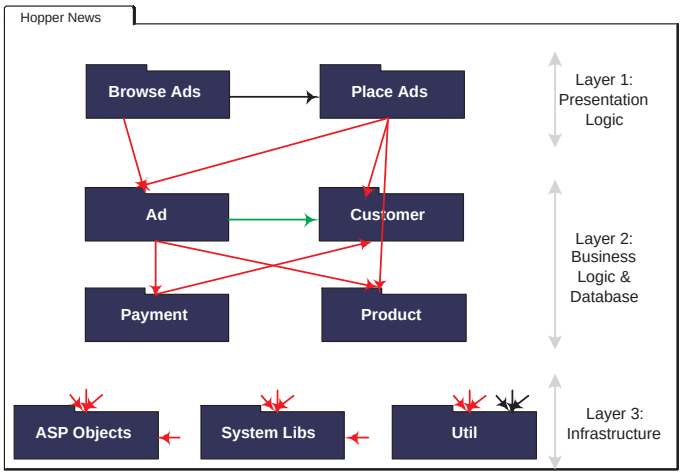

Figure 4: The Recovered Architecture of Hopper News

Since the Infrastructure subsystems are used extensively by all other subsystems, arrows to these subsystems have been truncated to simplify the diagram.

The viewer enables developers to browse and analyze the architecture using interactive queries such as "*What are the subsystems that use the* `Customer` *subsystem?*".

The viewer shows the relations between the various components. When the cursor is positioned on a specific arrow, a detailed description of the relation is displayed. From our analysis of various web applications, we decide to use three colors to group the relations into three categories:

- Black arrows indicate a hyperlink dependency.
- Red arrows indicate a control dependency.
- Green arrows indicate a data dependency.

Color and icon shape are used to represent the various kinds of components. Blue folders represent subsystems, blue ovals represent web objects, grey pages represent active and static pages, blue boxes represent DLLs[1], and

[1] A Dynamic Link Library (DLL) is a shared library on the Microsoft Windows operating system.

green cylinders represent database tables.

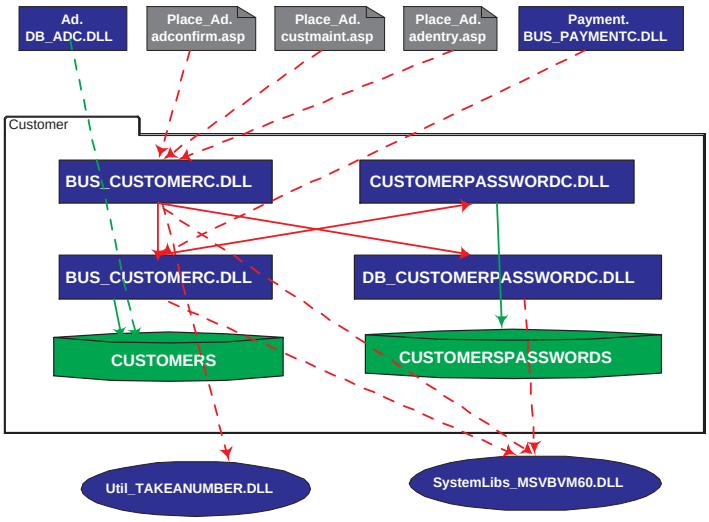

Figure 5: Hopper News Customer Subsystem

The viewer permits us to delve into the structure of each subsystem. To do this we double-click on the subsystem of interest and the viewer opens a new diagram showing the internals of the subsystem.

Figure 5 shows the internal structure of the `Customer` subsystem. It shows the various database tables that are part of the `Customer` subsystem and shows 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 aren't part of it.

5 RECOVERING THE ARCHITECTURE OF WEB APPLICATIONS

To recover the architecture of web applications, we use a semi-automated approach. A set of tools/extractors examine the source code of the web application. The tools' output is combined with input from a system expert to produce architecture diagrams. This approach is an adaptation of a similar approach used by the Portable BookShelf (PBS) [10, 23, 24] environment to recover the architecture of traditional software systems. The PBS environment incorporates knowledge and techniques developed over the last decade in program understanding and architecture recovery.

Figure 6 shows the steps involved in creating the architecture diagrams shown in the previous section. First the components of the software system are processed using specialized extractors. The extractors generate facts about the components, relations and attributes of the software system. 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

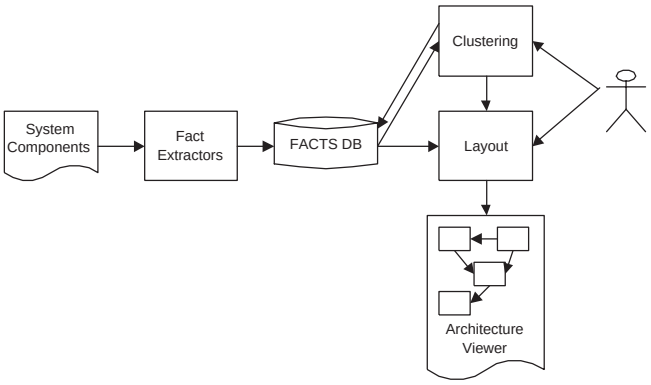

Figure 6: Our Architecture Recovery Approach

analysis to be performed on the recovered facts. For example, for architecture level analysis, facts at the function level aren't needed and can be lifted to the level of files or subsystems. The extracted facts are stored in TA format [15, 16]. Figure 8 illustrates how the extraction phase takes place for web applications. Each extractor generates facts that conform to a domain model (*schema*), as will be explained in the following section.

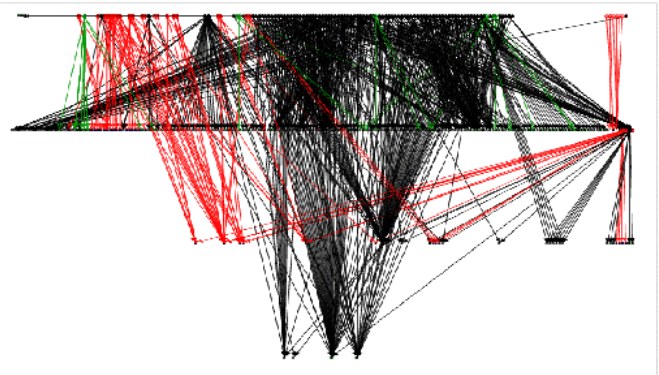

Figure 7: View of Facts Without Clustering

Once the facts have been produced, an immediate attempt to visualize them would lead to an architecture view which resembles figure 7. The figure shows a complicated graph of the relations between the different components of a software system. Each small dot represents an artifact of the software system (such as a source file, a database, *etc.*), and each line between two dots indicates the existence of a relation (such as uses, or calls) between two of the artifacts. The developer cannot use the diagram to gain a better understanding of the software system because of the complexity of the diagram. Instead of showing all the extracted relations and artifacts in the same diagram, we decompose the artifacts of the software system into smaller meaningful subsystems. The clustering is initially performed by a tool that proposes decompositions based on heuristics

such file naming conventions, development team structure, directory structure, or software metrics [34, 35, 3]. 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 4. The layout tool attempts to minimize the line crossing in the generated architecture diagrams [30, 31]. The developer may manually modify the generated layout.

The aforementioned process combines tool support and human input to recover the architecture. Tool support dramatically reduces the recovery time of a large software system. Human interpretation is essential in organizing and clustering the large amount of extracted information to produce meaningful architecture diagrams.

Extractors for Web Applications

Web applications are developed using a variety of languages and are composed 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 approaches that typically depend on a single extractor[18]. To deal with web applications, we developed five types of extractors:

- HTML extractor
- Server Script extractor
- DB Access extractor
- Source Code extractor
- Binary Code extractor

Each extractor parses a component or a section within a component and generates the appropriate facts. Together these extractors generate facts from the entire web application. Figure 8 shows an overview of the various extractors and their input and the type of facts generated by them. Once all the facts are emitted, the clustering information is combined and all the data (facts and clustering) are processed to reduce their complexity.

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 Code extractor is invoked. Each extractor stores its generated facts 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 directories and consolidates all the generated facts files into a single file called `THEFACTS`. This file is combined with the clustering information which decomposes the web application into a hierarchy of subsystems.

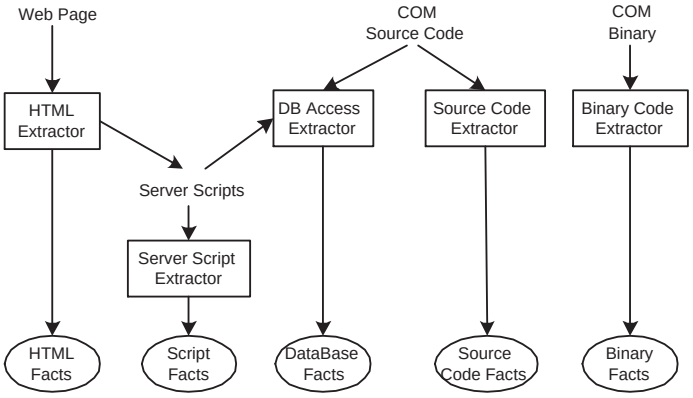

Figure 8: Conceptual Architecture of the Fact Extractors

To process the large code base written in several languages, we use these four extraction techniques: island extraction [17], robust extraction [8], heuristic extraction, and binary extraction.

Island Extraction

In web applications a single source file may contain multiple sections written in different programming languages. Special tags are used in the file to indicate the different sections and the programming language used. For example, an active page may contain sections of HTML code, VBScript code, and JavaScript code. Furthermore, many application servers have their own proprietary language.

To parse each file and extract the relations, it seems that we would have to implement parsers/grammars for each language. Instead we choose to extract only the entities which we are interested in. By considering each processed file as an ocean of tokens, a set of extractor where developed using grammars for each island of interest. Each extractor processes the file and locates the subsections (*islands*) of interest in the file. Once these are located, the appropriate parse is performed to extract the information.

Robust Extraction

Some of the languages used to implement web applications are still in development and aren't well documented (for example, VBScript). Also many application servers provide extensions for the programming languages. As the language reference manuals did not provide a grammar for the language, we based our extractors on the code examples and any available documentation. Then, we extended the parsers to support robust parsing. *Robust parsers* are capable of identifying and tolerating minor syntax errors without causing the parse to fail. This technique enabled us to overcome

many of the problems associated with parsing undocumented features in the studied languages and platform-specific extensions.

Heuristic Extraction
The Database Access extractor uses regular expressions to locate accesses to data tables within source code statements. As shown in figure 8, this extractor takes as input a server script or the source code of a component.

The extractor searches for common database access functions and SQL keywords such as `SELECT` or `INSERT`. The extractor then uses heuristics to validate the matches. For example, once the extractor locates the keyword `SELECT` it searches for the keyword `FROM` and determines, based on the distance and the strings between both keywords, if a database table is being accessed or if the match is coincidental and no database table access has occurred. Because of the use of heuristics, the output of this extractor is reviewed manually to validate it. The Database Access extractor has been tuned to work well for the systems we studied. For other systems, we expect that the extractor won't perform as well. User intervention is possible to correct the extractor if it fails to recognize database accesses or if it mis-recognizes database accesses.

Binary Code Extraction
The Binary Code extractor examines the binaries for compiled components. It extracts the function calls and the data from the symbols table stored in the binary. This extractor is used to analyze components when:

- We do not have access to the source code. The component may have been purchased as binary code or the source code of the components wasn't given to us for analysis due to its sensitive nature.
- We do not have a Source Code extractor. Many languages are used to develop web applications and we may not have a language extractor for each one.

6 SCHEMAS

As shown in the previous section, our recovery process depends on a set of extractors. Each extractor emits different type of data. For example, the Source Code extractor emits relations such as function calls, whereas the DB Access extractor emits relations such as database table updates. To generate architecture diagrams, we need to combine these types of data. We need to combine the output of all individual extractors to generate useful diagrams that show the inter-component interactions. Furthermore, we need to be able to reduce the details of the extracted facts to improve the readability of the architecture diagrams. Yet we should not throw away the detailed facts as the user may need to get more details. An ad-hoc methods may be used to achieve this goal. But we need to be able to later incorporate other extractors in the process with very minimal modifications. We use *schemas* to solve these problems.

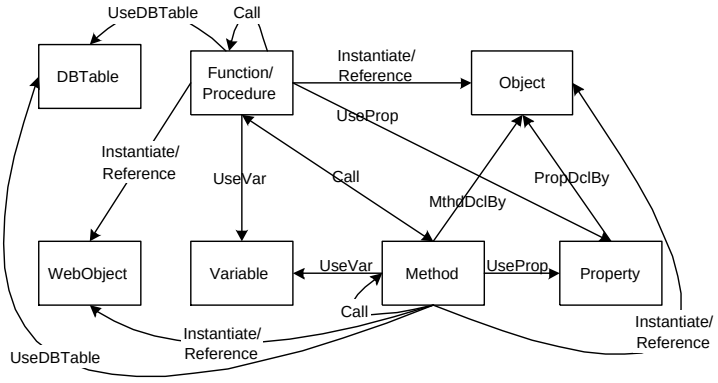

Figure 9: Entity Level Schema for the VBScript Language

Each fact extractor produces facts which conform to a particular domain model or *schema*. A schema is essentially an entity-relational model. It consists of *entities*, *relations* between these entities and *attributes* attached to the entities and relations. For example, as shown in Figure 9 the schema for the VBScript language has entities such as `Method`, `Procedure`, and `Object`. It has relations such as `UseVar`, `UseDBTable`, and `Call`. Each entity has attributes such as "`LineNumber`" and "`Filename`". We do not show the attributes to make the diagrams easier to read.

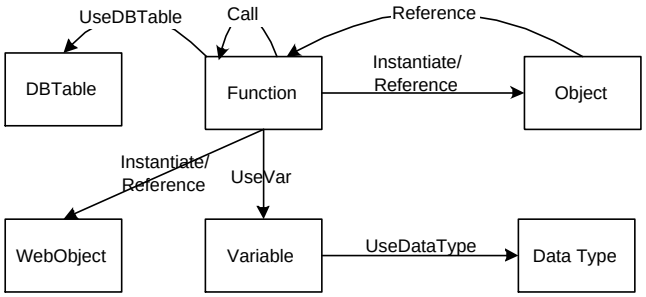

Figure 10: Entity Level Schema for the JavaScript Language

Figure 10 shows the schema for the JavaScript language. Given the VBScript and the JavaScript schemas we can take their union to create a schema for facts for both VBScript and JavaScript. With this union schema, we are able to consolidate facts into a common fact base. This allows us to study the interaction between components written in different programming languages. Figure 11 shows a union schema for both VBScript and JavaScript. All the common entities of the JavaScript and the VBScript languages are present in the common schema. They are prefixed with `VBS`

for VBScript entities and `JS` for JavaScript entities. As can be seen in figure 11, the common schema uses inheritance relations to indicate the mapping between the union schema and the language specific schema. For example, a `UserObject` entity is a super class of VBScript `Object` and JavaScript `Object`.

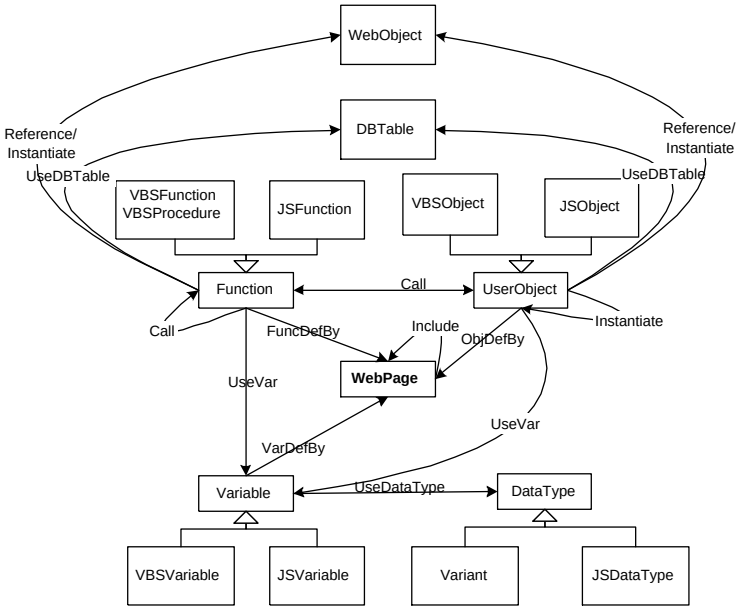

Figure 11: A Union Entity Level Schema for VBScript and JavaScript

If we were to visualize the extracted facts for the entire web application at this level of detail, we would get very complicated diagrams — similar to figure 7. At this level of detail we are visualizing every entity in our code base. This includes entities that aren't accessible outside of a particular source file, such as local variables inside of a function. To reduce the clutter in the architecture diagrams, we perform another two steps of schema transformations. These schema transformations raise the level of abstraction from the source code level to the architecture level. Figure 12 shows the three layers of schemas used in our approach.

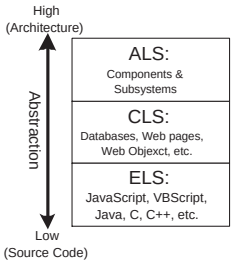

Figure 12: Layers of Schemas

The *Entity-Level Schema (ELS)* describes the permitted relations between the program entities such as functions, variables, objects, database tables and web objects. The ELS is language dependent, as previously indicated. Figure 9 and 10 are examples of an *Entity-Level Schema (ELS)*. Figure 11 is a union of such schema.

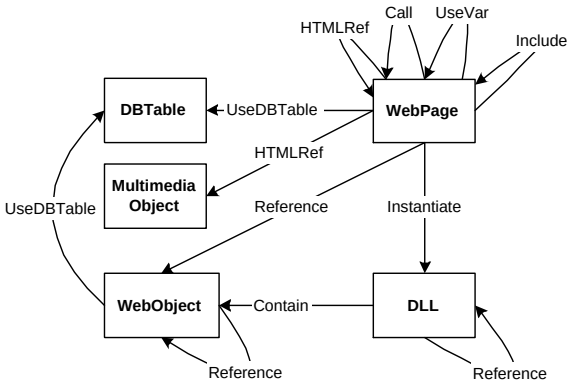

Figure 13: Component Level Schema for Web Applications

The *Component Level Schema (CLS)* describes the relations between the components of a web application such as web pages, database tables, binary libraries and distributed objects. It raises the level of abstraction of the extracted facts from the internal structure of the components to the components themselves. Figure 13 shows the CLS, which contains the various components of a web application and their interrelations. As can be seen in the figure, a `WebPage` component can include other `WebPage`s. The HTML code in a `WebPage` can reference another `WebPage` or a `MultimediaObject` such as a picture, a movie, or an audio file. Also code segments written in JavaScript or VBScript in a `WebPage` can use a variable, instantiate an object, or call a function defined in another `WebPage`. The code segments can also read, update, and insert data in database tables. The code segments can instantiate or reference a `WebObject` that may reside inside a `DLL`. A `WebObject` can in turn reference other `WebObject`s or it can access `DBTable`s.

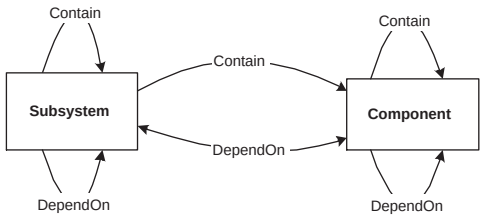

Figure 14: Architecture Level Schema for Web Applications

At the highest level of abstraction we have the *Architecture-Level Schema (ALS)* which describes the relations between the architecture elements which are subsystems and components, see figure 14. The ALS is

independent of language (such as VBScript, JavaScript) and independent of technology (such as DCOM, EJB, or CORBA).

The different levels of schemas enable developers to study their software system at various levels of abstraction. Developers can perform a detailed analysis on the source code or a higher-level analysis on the architecture. They can *drill down* [7] to investigate architecture anomalies at the source code level such as unexpected subsystem interdependencies. Using *roll up* operations, developers can investigate the effects of source code changes on the overall architecture of the software system [21]. More generally, schemas provide a convenient conceptual framework that helps developers think about the structure of web applications and underlies the construction of tools that help developers explore and visualize this structure.

7 RELATED WORK

This section compares our approach to architecture recovery of web applications to related research that addresses web applications. In particular, we address work on the evolution, the architecture recovery and the modeling of web applications.

Web Applications: Evolution

Brereton *et al.* point out that HTML's nested tags are analogous to the block structure of third generation programming languages and that links between pages are analogous to `GOTO` statement [5]. They demonstrate a tool that can track the evolution of pages in a web site. The tool is based on a modified network crawler that visits the web site multiple times over a span of a year and reports the change in the contents of the web pages. Ricca and Tonella developed a similar tool [26].

Both tools are suited toward static web sites which do not have active pages, web object, or databases. Their crawler won't recognize that many apparent changes in the web site may be due to active pages and not to the actual modification of the content of the web page. For example, the tool would indicate that the homepage of a site like *CNN.COM* changes continuously, but in fact the source of the home page rarely changes. It is an active page, which contains executable code that retrieves updated information from a database, at the time the page is accessed.

The approach suggested by Rica, Tonella, and Brereton puts more emphasis on the user's experience. It attempts to track the changes that are sensed by the user of the application. These changes may not be mirrored in the source code of the web application. Also, changes in the source code may not affect the pages viewed by the user. For example, in an early version of an email service web application, the page displaying the user's inbox may retrieve all the email messages from a flat file. In a later version, the email messages may be stored in an SQL database. When accessing the page, the user won't notice the change. Clearly, the architecture of the web application has changed. Our approach analyzes the source of the components of a web application. Using this approach, we can study more sophisticated dynamic web applications. Stated differently, We use a white box reverse engineering approach and they use a black box approach.

Web Applications: Architecture Recovery

The related work discussed thus far is based on tools such as parsers and crawlers. In contrast, Antoniol *et al.* suggest a non-automated technique to recover the architecture [1]. The technique is founded on the Relation Management Methodology (RMM), which in turn is based on the Entity Relationship model. Using RMM, the application domain is described in terms of entity types, attributes and relationships. For example, an exam booking web application would have entities such as students, and courses; and relations such as "takes", and "provides".

RMM is best suited for static web applications whereas our approach can be used to recover the structure of dynamic web applications. The RMM recovered architecture is a high level view of the main entities in the domain and the relations between them. Our approach extracts the implementation view of the system. The high level view of a system tends to remain stable across different releases compared to the implementation architecture which changes as new technologies are deployed, a common occurrence in the fast moving web application domain. For example, an RMM recovered architecture would not changed if the courses and the students are now stored in a database instead of a flat file.

Web Applications: Modeling

To aid developers in their analysis, researchers have proposed various methods to model web applications. Each method emphasizes an aspect or a set of interesting relations that the developers can model. Ceri *et al.* present the Web Modeling Language (WebML) [6]. WebML provides a high level conceptual description of a web application. The language is geared towards database-driven applications. WebML is more suited for the high level specification of web application than for modeling the actual implementation because it lacks the concepts needed to model control flow. For example, relations between the objects and the call graph cannot be expressed using WebML's constructs.

Conallen's work on extending UML [11] to model web applications is the most similar to our work [9]. Conallen presents the Web Application Extension (WAE) for UML. Web pages are modelled as UML com-

ponents. Every web page is modelled using two different aspects:

1. The *server side* aspect where he shows the page's interaction with other pages, the business logic objects, the databases and the server provided resources.
2. The *client side* aspect where he shows the page's interaction with the browser's built-in objects and Java applets. This aspect is more concerned with the page's layout and presentation.

By contrast our approach focuses mainly on the server side aspect of modeling web applications. Conallen has demonstrated the generation of skeleton code for a web application based on a UML specification of the application. His work can be thought of as the forward engineering of web applications where our research is concerned with assisting developers who did not specify their web application using UML but need to understand them. We could convert our diagrams into UML compliant diagrams using a schema transformation. Current UML tools do not provide the features available in our specialized viewer, such as querying and hierarchal navigation.

8 CONCLUSION

In the work we have done, we concentrated on the extraction of web applications developed on the Microsoft Windows platform. While we do not foresee problems with extractions for applications developed on other platforms, we recommend that further research be done to generalize and improve our approach. Also detailed empirical studies are needed to verify the benefits of our tools for web developers.

We presented an approach to recover the architecture of web applications. The approach uses a set of specialized parsers/extractors which analyze the source code and binaries of web applications. We described the schemas used to produced useful architecture diagrams from highly detailed extracted facts. The recovered architecture is shown as simple box and arrow diagrams. This visualization helps developers of web application understand the structure of their application, helping them to rapidly update it to handle new requirements.

The need for tools to assist developers of web application is clear and justifiable. Web applications are tomorrow's legacy software. With a short release cycle, a "*no-documentation*" culture and high employment attrition rates, web development companies face severe challenges to stay competitive in a highly volatile market.

ACKNOWLEDGEMENTS

To validate our approach, 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] G. Antoniol, G. Canfora, G. Casazza, and A. D. Lucia. Web Site Reenginnering using RMM. In *Proceedings of euroREF: 7th Reengineering Forum*, Zurich, Switzerland, Mar. 2000.

[2] C. Boldyreff. Web Evolution: Theory and Practice, 2000. Available online at `<http://www.dur.ac.uk/cornelia.boldyreff/lect-1.ppt>`

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

[4] 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.

[5] P. Brereton, D. Budgen, and G. Hamilton. Hypertext: The Next Maintenance Mountain. *Computer*, 31(12):49–55, Dec. 1998.

[6] S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing Web sites . In *The Ninth International World Wide Web Conference (WWW9)*, Amsterdam, Netherlands, May 2000. Available online at `<http://www9.org/w9cdrom/177/177.html>`

[7] S. Chaudhuri and U. Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1), Mar. 1997.

[8] G. Clarke and D. T. Barnard. Error Handling in a Parallel LR Substring Parser. *Computer Languages*, 19(4):247–259, 1993.

[9] J. Conallen. *Building Web Applications with UML*. object technology. Addison-Wesley Longman, Reading, Massachusetts, USA, first edition, Dec. 1999.

[10] 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>`

[11] T. O. M. Group. *Unified Modeling Language Specification*. The Object Management Group, June 1999. Available online at <http://www.rational.com/media/uml/post.pdf>

[12] A. E. Hassan and R. C. Holt. A Reference Architecture for Web Servers. In *7th Working Conference on Reverse Engineering*, Brisbane, Queensland, Australia, Nov. 2000.

[13] A. E. Hatzimanikatis, C. T. Tsalidis, and D. Christodoulakis. Measuring the readability and maintainability of Hyperdocuments. *Software Maintenance: Research and Practice*, 7:77–90, 1995.

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

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

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

[17] Island Grammars. Available online at <http://losser.st-lab.cs.uu.nl/~visser/cgi-bin/twiki/view/Transform/IslandGrammars>

[18] R. Kazman and S. J. Carrière. Playing detective: Reconstructing software architecture from available evidence. *Automated Software Engineering*, Apr. 1999.

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

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

[21] E. H. S. Lee. Software Comprehension Across Levels of Abstraction. Master's thesis, University of Waterloo, 2000.

[22] T. C. Lethbridge and N. Anquetil. Architecture of a Source Code Exploration Tool: A Software Engineering Case Study. Tr-97-07, School of Information Technology and Engineering, University of Ottawa, 1997.

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

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

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

[26] F. Ricca and P. Tonella. Visualization of Web Site History. In *Proceedings of euroREF: 7th Reengineering Forum*, Zurich, Switzerland, Mar. 2000.

[27] S. E. Sim. Supporting Multiple Program Comprehension Strategies During Software Maintenance. Master's thesis, University of Toronto, 1998. Available online at <http://www.cs.utoronto.ca/~simsuz/msc.html>

[28] S. E. Sim, C. L. A. Clarke, and R. C. Holt. Archetypal Source Code Searching: A Survey of Software Developers and Maintainers. In *Proceedings of International Workshop on Program Comprehension*, pages 180–187, Ischia, Italy, June 1998.

[29] S. E. Sim, C. L. A. Clarke, R. C. Holt, and A. M. Cox. Browsing and Searching Software Architectures. In *Proceedings of International Conference on Software Maintenance*, Oxford, England, 1999.

[30] K. Sugiyama and K. Misue. Visualization of Structural Information: Automatic Drawing of Compound Digraphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(4):867–892, July 1991.

[31] K. Sugiyama, S. Tagawa, and M. Toda. Methods for Visual Understanding of Hierarchical System Structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, Feb. 1981.

[32] S. Tilley and S. Huang. Evaluating the Reverse Engineering Capabilities of Web Tools for Understanding Site Content and Structure: A Case Study. In *IEEE 23st International Conference on Software Engineering*, Toronto, Canada, May 2001.

[33] S. R. Tilley. Web Site Evolution. Available online at <http://mulford.cs.ucr.edu/stilley/research/wse/index.htm>

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

[35] 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.