Architecture Recovery of Web Applications

by

Ahmed E. Hassan

A thesis

presented to the University of Waterloo

in fulfilment of the

thesis requirement for the degree of

Master of Mathematics

in

Computer Science

Waterloo, Ontario, Canada, 2002

©Ahmed E. Hassan 2002

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Abstract

In the last decade, much of the reverse engineering research has concentrated on the development of tools that support traditional software development environments and languages such as COBOL, PL/I, Pascal, C, Java, and C++. This thesis extends this body of research into a new domain: web applications. We develop extractors that analyze the source code and binaries of web applications. Then, we perform algebraic manipulations on the extracted information to generate architecture diagrams that highlight the main components of a web application and the interactions between them. Furthermore we show how to use the extracted architectures to help gain a better understanding of web applications and to help maintain them.

Acknowledgements

This thesis would not have been possible without the continuous support of my family who gave me the strength and will to succeed.

I would like to thank my supervisor Prof. Richard C. Holt for his support and advice. Thanks Ric, I'll carry your lessons and guidance with me through out life. A special thank you to Prof. Joanne Atlee, thanks Jo for being my instructor, for listening to my ideas, and for offering me the opportunity to teach at Waterloo - a very enjoyable experience.

I appreciate the valuable feedback provided by my thesis readers, Prof. Kostas Kontogiannis and Prof. Michael W. Godfrey. I am grateful to Wai Ming Wong for his assistance while I worked on my case studies with Sun Microsystems of Canada Inc.

I am fortunate to work, play, and study side by side with the amazing members of SWAG. In particular, I would like to thank Eric Lee, Ivan Bownan, John Tran, and Thomas Parry for all their help and encouragement.

Thanks for my friends who patiently put up with me while I worked away on my thesis, and accompanied me on the occasional escape away from it.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Ten years ago Tim Berners-Lee created the first web browser and web server, and started the World Wide Web (WWW) revolution. The web revolution has been shaping and will continue to influence our society for years to come. Many technical and non-technical aspects in our life are changing everyday as we become more dependent on the web.

> "Every day it becomes clear that the Net (Web) is taking its place along-side the other great transformational technologies that first challenged, and then fundamentally changed, the way things are done in the world",
> *Lou Gestner, CEO of IBM Corporation.* [Boo00]

The web browser's ubiquitous and simple interface has opened the door for the development of many new distributed applications. In 1994, web pages were simple static HTML pages linked together. The web pages provided easy and open access to information across the world. Subsequently new technologies such as Java were introduced and used as “eye candy” to decorate pages. Suddenly dancing puppets and flashing pictures adorned web pages. Luckily the web, the world realized the potential of the web and large-scale applications that harness the power of the web emerged. Companies started developing web applications, which delivered services to their customers worldwide and provided outstanding value to their shareholders. The days of the web as a medium for just sharing documents are over and the next era of the web as a vehicle for commerce has begun. Software is at the heart of this revolution providing companies with the infrastructure and the tools to develop large software systems.

Too often, software-engineering principles are not applied to the development of web application. As Pressman notes, the reluctance of web developers to adopt well-proven principles is worrisome [Pre00]. The techniques used nowadays by web application developers are similar to the ad hoc ones used by their predecessors in the 1960s and 1970s. They defend their reluctance to adopt software-engineering principles using reasons such as:

1. *The speed of development*: Whereas traditional application development takes many months, web applications are developed in weeks or even days. The speed of web development is commonly referred to as web speed. Developers believe that product requirements and web development technologies are changing at a very high pace; thus they should not spend time carefully specifying and planning the software.

2. *The speed of evolution*: Web applications become obsolete as soon as they are released. For traditional software systems, legacy systems may have been

running for the past ten years. A web application may be considered legacy after only six months. The speed of evolution discourages many developers from following software engineering processes and carefully gathering requirements.

3. *The different sets of concerns*: The mapping of web applications to the traditional object-oriented or procedural systems is not clear because of the development technologies used. Current development tools are geared towards fast one time releases cycles with no incorporation of abstractions nor support for reuse. [GG99] The tools provide many methods to automate the generation of the code for web application but do not provide any facilities for reusing code or encapsulating into development libraries.

4. *The expectations of the user*: The users of web applications are more tolerant of errors as long as they are using new "cool/bleeding-edge" applications. Thus, web application development does not follow verification and testing phases as rigorous as the ones followed in traditional applications development.

5. *The origins of the web*: The web was developed as a document-sharing platform, and is still often considered as such. Consequently, the development of web applications is considered as an authoring problem and not a software engineering problem.

Software engineering researchers recognize the need to apply well-studied and tested principles to the development of web applications. Also they acknowledge the difference between traditional software development and web application development. Modification to software specification [CFB00] and design techniques [Con99] have been proposed to fulfill the need of web applications.

Unfortunately, the web development community has generally not adopted the proposed techniques. After weighing the short term benefits, they believe that the overheads associated with the adoption of these well studied techniques do not justify using them. As the web application domain matures, developers will need tools and methods to help them analyze, design and build web applications. Furthermore, they will need to deal with legacy web applications.

1.1 Overview of Thesis

Eventually, the web community will understand the benefits of using engineered principles in their development process and adopt them. As software evolves and its complexity increases, ad hoc methods won't suffice. Developers need a better understanding of their software system as they join new companies or maintain old code bases. The speed of the development and the rate of change of web applications increases the need for adopting an engineered approach. Legacy web applications are a reality; just a couple of months old they are critical to their organization. Yet the documentation associated with them generally does not exist and if it does, it is rarely complete or up-to-date. In addition, the web community has a high turnover rate: the average employee of many companies developing web applications tend to leave their company in just over one year [Kon00]. Thus the original developers of a maintained web application, may no longer be part of the organization. Lack of documentation and system experts increase 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 applications. Research [Hau00] has demonstrated the use of semi automated techniques and tools to recover the design of large applications. The Portable Bookshelf (PBS) environment [PBS] 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[1]) [BHB99], Apache (80 KLOC) [HH00], and Mozilla (2.1 MLOC[2]) [Lee00a]. In this thesis, we reuse and extend the capabilities of PBS to support the design recovery of large web applications. We develop a set of tools capable of parsing and extracting relations between the components of web applications. Also, we visualize the extracted relations using PBS visualization engine. Finally we evaluate our techniques and tools using case studies of web applications.

1.2 Thesis Organization

The rest of this thesis is organized as follows: Chapter 2 provides a background on the use of software architecture to improve program understanding. Chapter 3 presents the web application domain and describes the type of information that is needed by developers to gain a better understanding of web applications. Chap-

[1]KLOC: Thousand Lines of Code.

[2]MLOC: Million Lines of Code.

ter 4 presents research related to our approach of studying web sites as software systems. Chapter 5 discusses our architecture recovery process for web applications — we present the different tools and techniques used to generate the architecture diagrams. Chapter 6 presents case studies to validate the effectiveness of our extraction and visualization techniques for maintenance and program understanding of large web applications tasks. Finally, Chapter 7 summarizes our results and proposes some areas of future research.

1.3 Major Thesis Contributions

In this thesis, we present a domain model for web applications. A set of tools are developed to extract relations in the domain model based on the source code of the web application. The low-level (*source code*) relations are abstracted to higher-level relations to ease the visualization of the architecture of web applications. The web applications are visualized using simple box-and-arrows diagrams.

The recovered architecture diagrams where shown to developers working on web applications. They acknowledged the usefulness of the diagrams. In many cases, the developers have pointed out that they had drawn by hand the same diagrams by inspecting the source code.

Chapter 2

Background

In this chapter we present concepts related to our thesis in the domain of software engineering. We emphasize the need for good program understanding to reduce the costs and efforts associated with program maintenance and new development. We introduce the concept of software architecture and its different views. In addition, we discuss the techniques used to reverse engineer traditional software systems to recover their software architecture.

2.1 Program Understanding

As the complexity and size of software increase, companies are faced with many challenges to understand and maintain their software systems. Studies indicate that at least fifty percent of the life cycle and budget of a software system are spent on maintaining it [Zve83]. To successfully perform maintenance tasks, developers need a good understanding of the software. Fifty to ninety percent of the maintenance efforts involve program-understanding tasks [Sta84]. The primary business of software is no longer new development; instead it is maintenance [Gla92] and a good understanding is needed to reduce the cost and length of maintenance efforts.

To aid in software understanding tasks, documentation is used to narrate different aspects in the life cycle of a software system. Unfortunately software developers are not interested in documenting their work. Documentation is the poor-stepchild of most software development efforts [LHGF98]. It rarely exists and if it does it is usually incomplete, inaccurate, and out of date. Faced with the lack of sufficient documentation, developers choose an alternative understanding strategy such as searching or browsing source code, the definitive source of accurate information about the system [Sim98]. Developers search the code using tools such as `grep`. They browse the code using simple text editors or cross-reference code browsers such as `LXR`, which permit jumping between variables/functions usage and variables/functions declarations while browsing the source files. The usefulness of this unaided browsing of source code is limited by the size of the software system and the amount of information a person can keep track of while jumping around the source tree [SCH98].

Much of the knowledge about the design of the system, major changes over the years and the troublesome subsystems live in the brains of its developers. Such live knowledge is sometimes called *wet-ware*. When new developers join a team, mentoring by senior members and informal interviews are used to give the new developers a better understanding of the system. Leveraging this knowledge may not always be possible as the software may have been bought from another company, its maintenance outsourced, or its senior developers are no longer part of the company. This problem is exacerbated by the fact that in many cases the developers tend to remain in the same company for no longer than one year [Kon00].

2.2 Software Architecture

The high-level design decisions of a software system become more critical than the details of the used algorithms and data structure, as its size and complexity increases. Software architecture has been proposed by many researchers [GS93, PW92]as a means to document high-level design decisions for complex software. By using simple box-and-arrow diagrams, software designers can easily and succinctly document and explain their design decisions. Although, the term "*software architecture*" has been recently proposed and lacks a clear definition, many agree that software architecture of a system is concerned with the different *components* of the system (*building blocks*); the *interactions* between them, and the *rationale* behind the choice of the components and the type of interactions. The software architecture of a system is used by newcomers to get a better understanding of the system, by system maintainers to familiarize themselves with parts of the system on which they never worked and by veterans for impact analysis.

The software engineering community has recognized the re-occurrence of similar software architectures and has documented these patterns as architecture styles [GS93]. Some of the common architecture styles are Pipe and Filer, Client and Server, and Layered styles. Architecture styles provide a common vocabulary for developer to communicate their designs and to rationalize their decisions. Ideally, the software architecture of a system is proposed before its implementation and updated during the software lifetime to reflect the changes that occur during maintenance. Unfortunately, this is rarely the case. The software architecture of large systems often exists only in the minds of its developers, and is never written down. Due to the recognized importance of software architecture, many techniques have been proposed to recover/update the architecture of a software system from its implementation. These techniques are commonly referred to as Reverse Engineering.

2.3 Architecture Views

A software architecture addresses many concerns and is used by many stakeholders in the organization. To avoid cluttering the architecture diagram with many details, researchers have proposed different views that address the needs of specific stakeholders. For example, in lieu of specifying the data flow and the concurrency of a large system on the same diagram, two separate diagrams (views) are used to represent these aspects separately. In this section, we present the work of Kruchten [Kru95] and Soni *et al.* [SNH95] on architecture views.

Kruchten proposes modelling architecture using four different views and one use-cases view to illustrate and validate the other views. Each view addresses a specific architecture concern for a particular set of stakeholders. In his *4+1 View Model of Architecture*, he defines the following views:

1. *The Logical view* depicts the functional requirements of the system, using abstractions drawn from the problem domain. It is used by the end-user

to ensure that all the required functionalities have been addressed in the implementation of the system.

2. *The Process view* details the concurrency and synchronization mechanisms used in the system. System integrators use it for analysis of the performance and scalability of the system.

3. *The Development view* shows the layout of the code in the development environment. The programmers and their managers use it, as it enables the planning and monitoring the progression of a project.

4. *The Physical view* is concerned with the mapping of the software system on to the hardware. System engineers employ this view to determine the topology of the system and the communication requirements between the different components.

5. *The Scenarios* are the *+1* in the *4+1*. They illustrate the different architecture decisions that are scattered across the four previous views. Sare explained explaining system features

Each view has its own notation and style. For example, OMT [RBP+91] notation is used to present the Logical View and DADS is used to demonstrate the Process View. Recently, there has been a push to use UML to present all the views, the different UML diagrams notations are used to present each view.

Soni *et al.* propose a similar decomposition of the architecture of large systems. Based on an empirical study of many large applications, Soni *et al.* define the following architectures for software systems:

1. *The Conceptual architecture* abstracts the design of the system using its major design elements at a very high level. It also shows the interactions between these high level elements.
2. *The Module architecture* represents the ideal implementation with no dependencies on language specific features. It decomposes the system using layers and functional decompositions
3. *The Execution architecture* is concerned with the topology of the deployed system. It shows the location of the different components and the communication between them.
4. *The Code architecture* shows the organization of the source code, the binaries, and the libraries in the development environment.

Soni *et al.* point out that each architecture encompasses decisions that have been taken at different times in the development process such as design time, implementation time, build time, and run time. Also the degree of change in each architecture varies considerably. For example, the conceptual architecture rarely changes whereas the execution architecture keeps on changing during the lifetime of the product to accommodate new performance requirements and benefit from new technological advances in hardware or software technologies.

2.4 Reverse Engineering

Chikofsky and Cross define reverse engineering to be "analyzing a subject system to identify its current components and their dependencies, and to extract and create system abstraction and design information." [CC90]. Different solutions to the problem of reverse engineering have been proposed and can be categorized into three main approaches: top-down, bottom-up, and hybrid approach.

In a top-down approach, the persons involved with the systems are interviewed and their knowledge is collected, summarized, and displayed using simple box and arrow diagrams. The boxes represent the different components of the system and the arrows represent the dependencies between them. The generated document is referred to as the conceptual architecture. Later, the code of the system is examined to verify that the conceptual architecture matches the actual implementation. As pointed out by Bowman [BHB99], the implementation is likely to have more dependencies than are shown in the conceptual architecture.

In a bottom-up approach the implementation of the system is examined first using special types of parsers, called fact extractors. The fact extractors scan the source code of the system searching for specific patterns in the code and output relation triples such as `function A calls function B` or `function A uses variable C`. The extracted facts are fed to a visualizer that generates different views of the architecture of the system such as call graphs or data access graphs.

Tzerpos and Holt propose a hybrid approach that combines the information derived from interviewing developers, and the extracted facts from the implementation to recover the architecture of large software systems [TH96]. Their approach is composed of the following steps which do not have to be in order and can be in parallel:

- *Extracting facts from the source code*: Fact extractors scan the implementation and output relations that conform to a source model. The extractors could be a modified language parsers that performs a full syntax analysis on the source code; generating an Abstract Syntax Tree that represents the structure of the code. Alternatively, the extractors could be lightweight parsers that simply scan the source file for patterns of interest and discard the rest of the source file. The development of full parsers is time consuming. A full parser is capable of recognizing a superset of the needed patterns; the extra patterns are ignored during the analysis. In addition, a full parser requires the source code for the whole application to perform its operations. The source code may not be available due to many reasons such as legal and competitive reasons [KLDM98]. The performance of a light weight extractor is not as good as a parser but this may not be a major concern if the extraction process runs as a batch and not as a daily process [MNS95].

- *Clustering into subsystems*: The extracted facts are grouped into meaningful subsystems using naming convention or directory structure. The clustering assists in reducing the complexity of large systems and makes the generated diagram clearer and simpler.

- *Refining the clustering using live knowledge*: The derived architecture from the previous steps is reviewed by interviewing the people involved with the system. If approved by the developers then we reached a sufficient clustering. Otherwise the system must be re-clustered and the output re-analyzed.

- *Refining the layout using live information*: The visualized facts are displayed on the screen using automatic layout algorithms that attempt to minimize the line crossings in the graph. Such heuristics do not attempt to convey any of the semantic or secondary notion that developer use when drawing architecture diagrams of the software system. For example, software developers tend to draw a pipe and filter architecture by placing the subsystems vertically on the same line to convey the flow of information from the left most to the right most subsystem. An automatic layout tool may not lay out a pipe and filter as expected by the developers. A manual review of the layout is necessary to emphasize any semantic information.

2.5 Summary of Background

Reverse engineering and software visualization techniques can help developers recover the design and architecture of large software systems and gain a better understanding of it. Software architecture views have been proposed to separate the different design concerns in large software. The software architecture can be recovered using semi-automated reverse engineering techniques. In the last decade, much of the reverse engineering research has been concerned with the development of tools that support traditional software development environments and languages such as COBOL, PL/I, Pascal, C, Java, and C++. In this thesis, we advance this research into a new domain: Web Applications. Geared with the knowledge acquired by reverse engineering researchers in the last decade, we apply and extend many of the traditional concepts developed to the web application domain.

Chapter 3

Web Applications

With the advent of the Internet, a new type of application has emerged: web applications, that use the Internet's infrastructure, are being developed and maintained everyday. Recent reports indicate that web applications represent more than thirty percent of software applications across all industry sectors and this number is expected to grow as the web gains popularity and its user base increases [Eco99].

In this chapter, we examine the web application domain. We present a definition of a web application and a taxonomy of its various types. Later, we present the architectural views of a web application, following *Kruchten's 4+1 views* [Kru95]. In addition, we augment the *4+1 views* with a *Security view* that highlights the design's security concerns and their mapping to the *Physical view*.

3.1 Definition of a Web Application

A web application [Con99] is a software system most of whose functionality is delivered through the web. 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 and servers. A web browser is used as the user's interface to the application and Internet protocols such as HTTP are used for communicating between the interface and the rest of the application. Users follow navigation links between pages to access objects that reside on the server, instead of calling methods in server objects as in traditional applications. For example, a user can surf to an online bookstore company such as *Amazon.com* where (s)he can browse, search, and purchase books. All these activities are simply done using her/his browser to follow links and fill order forms. Whenever the user clicks on a link on an Amazon web page, a request is generated and sent to Amazon's web server through the HTTP protocol. The web server receives the request and in turn invokes the appropriate action to generate a response. For example, a new order object is created to track the user's order. A database may be accessed and queried to retrieve relevant information to fulfill the request. All results are then transcribed to HTML and sent back to the browser, which displays them to the user. We contrast this with traditional client/server systems in which the user has to download specialized client software that communicates to the server using a proprietary protocol. A web site may contain multiple web applications. For example, *yahoo.com* web site is composed of multiple applications such as an email application, a calendar application and a news service application.

Web applications are preferred over traditional applications for the following reasons:

1. *Web applications are more accessible*: The HTTP protocol used in web applications is a standard protocol that can travel across corporate firewalls. Thus, applications are accessible to many users ranging from home users to corporate users. Traditional applications use proprietary protocols that are usually blocked by firewalls, limiting users access to them. For example, If Amazon were a traditional client server application, corporate user would not be able to purchase books. Access to Amazon's server would be blocked by the firewall due to the security risks associated with unknown protocols. In addition, a web application does not require a specialized client. A web browser, which nowadays comes packaged with almost all operating systems, is used as the client. Users do not need to install, configure or maintain client software. Also, the application is accessible on a multitude of platforms as long as a web browser exists for the platform.

2. *Web applications have a lower maintenance and deployment costs*: Since the browser is used as the client software for web applications, there are no costs associated with development of the client's software. Maintaining the web application requires only modifying the code that resides on the server. This reduces the cost of upgrade and deployment of web applications compared to traditional client/server applications.

3.2 Taxonomy of Web Applications

Web applications are not limited to one type of application. They can range from simple static web pages (such as a personal web site, a home page) to sophisticated e-commerce applications (such as *Amazon.com*, *eBay.com*). Figure 3.1 shows the different categories of web applications grouped according to their data and control complexity [MMAC99]:

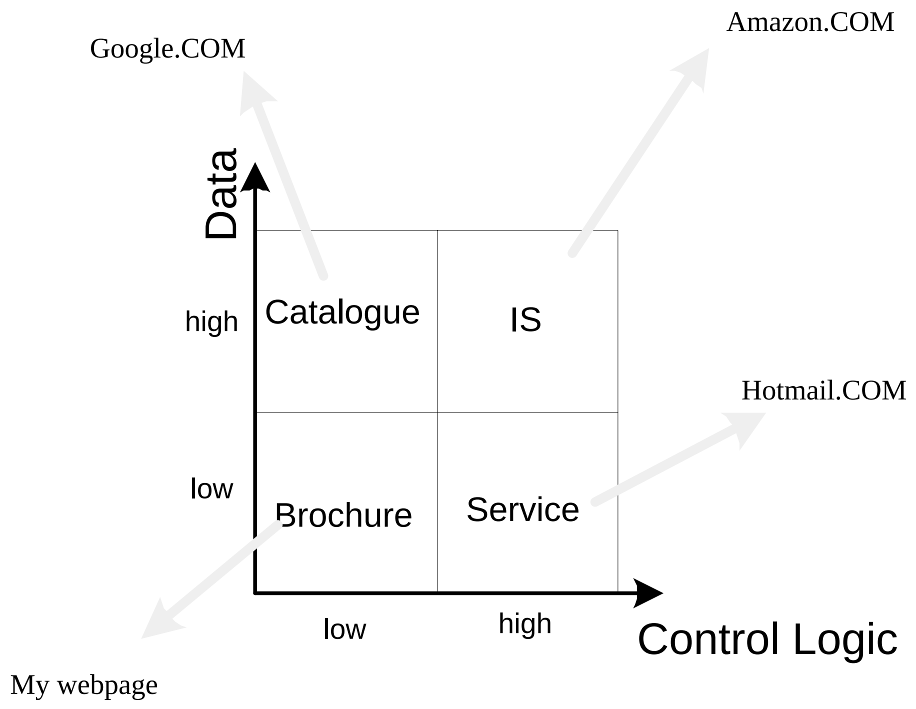


Figure 3.1: Taxonomy of Web Applications

1. *Brochure*: Brochure web applications are the first generation of web applications. They tend not to have much programming logic in them, rather they are composed of simple static web pages. Their developers are referred to as content developers, as they are more concerned with the layout of graphics

and text on a web page and the content is very static and graphics intensive. Examples of Brochure applications include: the personal web page of a person which simply contains their resume and personal information, and web sites that contain technical documents (brochures) about a company's product. Simple editors or specialized HTML editors are used to develop Brochure sites. The number of pages is rather small as it is manually edited and maintained. These sites are more similar to desktop publishing than to traditional software systems. These sites are the easiest to visualize. They are not of interest to our visualization effort, because we are more concerned with the control and data flow across the different components; whereas these sites are rather static with no control or data flow.

2. *Service oriented applications*: These sites are dedicated to providing a service to web users, such as free email service or online word-processing systems. In these applications, the layout of the data is a secondary concern. Instead, the developer is concerned with implementing the logic needed to provide the services online. For example, the developer of an online email service is more concerned with the different functional steps needed to store and retrieve email messages. The layout of the mail message displayed is of secondary interest. During maintenance, the developers need a good understanding of the control flow between the different components of the applications.

3. *Data intensive applications*: Theses are sites that provide an interface to browse and query large quantities of data, such as online library catalogues. The main emphasis in these applications is on the data, with minimal amount

of logic or control involved. Large commercial examples of these applications are search engine sites such as *Google.com*, and online news sites such as *CNN.com*. A search engine simply provides an interface to query a large database that indexes web pages, there are no or minimal control concerns involved in the development of such application. Data Intensive applications are closely tied to their database. A clear picture of the data flow is vital during maintenance.

4. *Information system applications*: These applications are a mix of Service Oriented and Data Intensive applications. An example of these sites is an online library system where you can; in addition to browsing books, borrow, reserve and recall books. Most large electronic commerce sites are in this category such as *Amazon.com*. The developers of Information System applications are concerned with the data flow (for browsing and retrieving books) and control flow (for the different phases involved with ordering and shipping a book).

We observe that developers need a good understanding of the data and control flow in their application as needed in traditional applications. In addition, web applications have more dependencies and interesting relations such as the navigation links between the different pages of the web application. Unfortunately, current web application development tools are implementation oriented that emphasize fast, one-time release with no continuity and process enforcement [MMAC99]. This emphasis on implementation productivity with no concern on the maintenance and evolution of web applications is attributed to the fast pace of their development and the immaturity of the web applications domain.

3.3 Web Application Architecture Views

Web applications are distributed applications that use web technologies as their infrastructure. They use web browsers as their clients, the HTTP protocol to communicate between clients and servers, and the HTML language to express the content transmitted between servers and clients. They use a myriad of technologies and a single architecture view is complicated and not sufficient. In this section, we describe a set of architecture views of a web application. We concentrate on the four views proposed by Kruchten [Kru95]: *Logical*, *Process*, *Physical*, and *Development* views. Each view captures specific design decisions and all views must be examined together to gain a good understanding of the whole application. We note that web applications are in need of an additional view to show the security architecture of the application. As web applications are composed of many different components and they use the Internet, a public network, they are more vulnerable to attacks. The security of web applications merits its own view. Therefore we augment the *Kruchten's 4+1 views* with a *Security view*.

3.3.1 Physical View

The Physical view presents the mappings of the components in the Development view to the components in the environment. The environment of web application is composed of many components that are inter-linked together to implement its functionality. Web applications have a rich environment, which contains the following components:

- Web browsers (used by the clients)

- Web servers
- Web pages
- Application servers
- Application pages[1]
- Databases
- Distributed objects[2] such as CORBA, EJB and COM.
- Multimedia web objects such as Images, Videos, and etc.

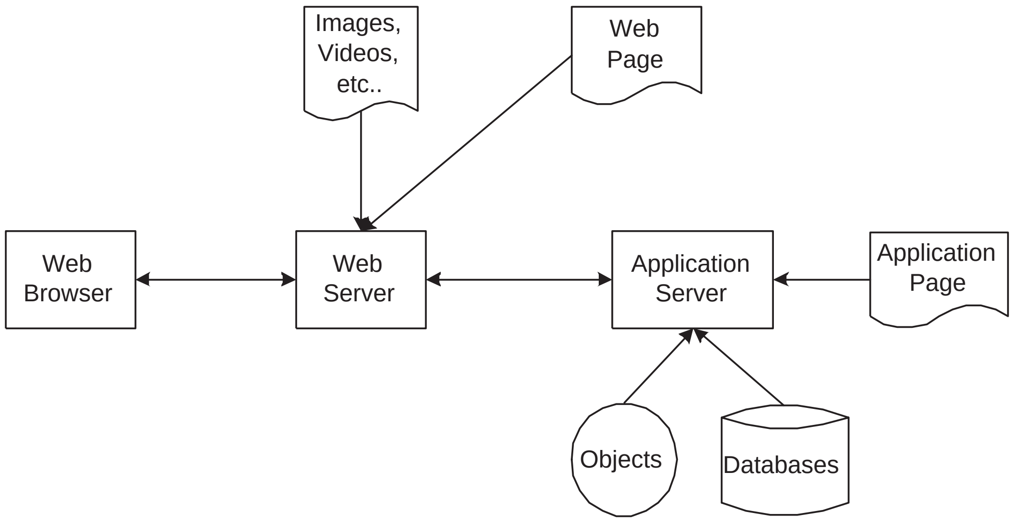


Figure 3.2: Physical View of the Architecture of a Web Application

[1]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 client.

[2]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.

Figure 3.2 shows the data flow between the different components in the physical view of a web application. The user of the application employs the web browser as the interface to gain access to the web applications' functionality. 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. In addition, many web servers and application servers can serve requests for a single application. This technique enables the distribution of web application processing and increases the fault tolerance of a web 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 resulting page to the requesting web browser.

3.3.2 Development View

The Development view focuses on the mapping of the Logical view conceptual components to the actual implementation artifacts. It presents that actual software module organization in the development environment, such as the source code files, or the directory structure. Web applications are developed using many of languages and technologies, compared to traditional applications, which are usually developed in one language. The Development view for web applications must highlight the additional details such as:

- The Link structure of the application pages
- User's session management techniques
- Application page generation technology

Many web pages are linked together to form the web application. The links between the different pages are verified regularly using scripts to avoid dead links and dead paths, which are considered as bugs. Concepts such as *exit links*[3] and *adverting links* [4] are used to quantify the different types of links in the web application.

Web applications use the HTTP protocol as their communication medium. The HTTP protocol is a stateless[5] protocol. The client connects to the web server and requests a page using the HTTP protocol. Once the page has been served, the connection between the client and the server is terminated. If the client needs to request another page, the connection with the server must be re-established. For the server, each request is considered as an independent one. The server cannot determine if the requesting client is the same as a previous client. This technique permits servers to handle a large number of clients, as the clients consume server resources only when they are requesting a document. Otherwise the connection is terminated and the resources are freed. However, the different application pages

[3]Exit links are links to other web sites than the current web site

[4]Advertising links are links to web sites that host advertisements

[5]HTTP is a stateless protocol because each request for a new web page is processed without any knowledge of previous pages requested

need to identify the user requesting a page as being the same user that had requested previous pages or a new user. For example, they need to determine if the user has provided her/his username and password or if they should ask the user to login. In addition, application pages must be able to determine if a user has exited the application. In a traditional application, as soon as the user exits by terminating the process, the connection with the server is terminated and the application is aware that the user has exited. In web applications the users can exit the application by simply closing their browser window or surfing to a new web site, no signals are sent to the server to signal the departure of the client. Many web applications use timeout to determine if the user has exited the application. To summarize, the stateless nature of the HTTP protocol prevents a web application from recognizing if two requests have originated from the same user during the same session. This described problem is called the user's session management problem. Many techniques exist to overcome it such as Cookies[6] or hidden fields in the page are used to store a session identifier to recognize returning clients from new ones. The application may choose a combination of different techniques for different subset of application pages. The choice and the rationale are detailed in the Development view.

Application pages are a mixture of HTML tags and control code. The control code is used to personalize web pages and the HTML tags are used to format the output of the page. When an application page is requested, the application server preprocesses it and integrates data from various resources such as distributed

[6]A cookie is a message given to the browser by the server. The browser stores the message and sends it back to the server each time it requests a page from the server.

objects or databases, to generate the final HTML web page sent to the browser. The developer has two options for developing their application pages:

1. *Code-based*: The first option focuses on the control code. The HTML page is fully generated by an executable program. Print statements (such as *printf("..")*) are scattered through out the code and are used to output HTML code. When the page is requested, a program is executed and the parameters sent by the browser are communicated to the executable through command line parameters or environment variables. All the output generated by the program is sent directly to the browser. Some examples of this technology are: CGI, ISAPI, NSAPI, and Java Servlet. Figure 3.3 shows a sample of an *Executable-based* application page. First, the C code is compiled and stored in the application server directory. The application server executes the binary whenever this particular server page is requested. Figure 3.4 shows the resulting HTML page that is sent to the requesting browser.

```
main() {
      printf("Content -type:test/html \n\n");
      printf("<html>/n");
      printf("Hello World  \n");
      printf("</html>");
}
```

Figure 3.3: "Hello World" CGI Written in C

```
<html>
Hello World
</html>
```

Figure 3.4: The Final HTML Displayed in the Browser

2. *Template-based*: In Template-based application page, the focus is on HTML. The HTML language is extended with tags to embed control code. The tags indicate scripts that must be executed first to complete the HTML page before it is returned to the requesting browser. Many companies provide frameworks for the development of these type of pages such as PHP from ZendTech, AOLserver Dynamic Pages from AOL, Cold Fusion (CF) from Allaire, Active Server Pages (ASP) from Microsoft and Java Server Pages (JSP) from SUN. The scripts embedded in the HTML page are usually interpreted. Some implementations may compile the scripts on first execution and reuse the compiled version later to improve performance. Figure 3.5 gives an example of a simple Template-based application page that uses the VBScript language to specify the control. The control code isn't sent to the requesting browser instead the result of the execution of the code is sent (i.e. "*Hello World*"), as shown in Figure 3.4.

```
<html>
<SCRIPT LANGUAGE="VBScript" RUNATSERVER>
     response.write("Hello World");
</SCRIPT>
</html>
```

Figure 3.5: "Hello World" Template Written in Visual Basic

Template-based pages are easier to develop because they use high level languages that abstract many of the implementation detail. However, they are not as powerful as *Code-based* pages. *Template-based* pages are slower to execute because of their interpretive nature. *Code-based* pages are much faster to execute but have a longer development cycle as the code must be compiled and the binaries updated in the application server.

Common Object Model

Many of the *Template-based* frameworks provide a set of built-in objects to abstract many of the implementation details. These objects shown up in the *Development view*. We present three frameworks: ASP from Microsoft, JSP from SUN, and CF from Allaire and point out the similarities between the object models they use to abstract the environment. Table 3.1 details the common object model between all frameworks and the main purpose of every object is explained.

ASP is a programming framework developed by Microsoft. ASP permits developers to combine HTML, scripts (written in VBScript, JScript, or Perl), and objects (COM, DCOM, and CORBA) to create web applications. The scripts are embedded in HTML file using the “`<%`” tag to indicate the start of a script and the “`<%`” tag to indicate its end.

To counter the popularity of ASP, SUN developed JSP. Using JSP, developers can glue HTML, scripts (written in Java), and object (Enterprise Java Beans) to build powerful web applications. Identical tags (“`<%`” and “`%>`”) are used by both JSP and ASP to to delimit the scripts embedded inside of an HTML file.

CF is another web development framework developed by Allaire Corp. Whereas JSP and ASP permit the embedding of scripts in HTML files to implement the control flow in a web application, CF uses a markup language called Cold Fusion Markup Language (CFML). CFML has many tags that represent the control flow of a traditional language such as `<CFIF>` or `<CFELSE>`. Figure 3.6 shows a simple example of the CFML markup language.

Object Name	**Purpose**
Request	The Request object represents a server side model of the web browser request. It is commonly used to retrieve the information entered by a user in a form or to retrieve the cookies stored in the web browser.
Response	The Response object represents the information sent from the web server to the browser. It is used to write output back to the web browser.
Session	The Session object represents a particular user session. A user session starts when the user's browser requests the first page from a web site. It ends after a preset period of time since the last page request. This object stores all the session variables that retain their values as the user moves from one page to the next.
Application	The Application object represents a global space for the whole web application. A web application starts after the first page in an application is requested by the browser. It ends when the application server is terminated. Application variables are shared between all the applications pages. An Application object may store a variable to count how many users have accessed the application since it started. Also it can be used to pool database connections for reuse by different application pages
Server	The Server object represents the internal state of the application server. It stores common configuration properties that are shared among the applications running on the application server. For example, The email of the server administrator may be stored in the Server object.
Error	Each application page has one Error Object that represents all the errors that occurred during the interpretation of the application page by the server.

Table 3.1: Common Object Model

Each framework uses a different programming language to express the control flow in the application page. Nevertheless we can abstract the commonality between them to derive a canonical (common) object model. Table 3.2 shows the mappings across the three frameworks. The ASP column shows the VBScript objects that provide similar functionality to the canonical objects. The JSP column shows the corresponding Java objects and the CF column shows the tags.

```
<html>
<CFOUTPUT>Hello World</CFOUTPUT>
</html>
```

Figure 3.6: "Hello World" Template Written in Cold Fusion

Canonical Objects	ASP Objects	JSP Objects	CF Objects
Request	Request	javax.servlet.ServletRequest	<CFHttpParam>
Response	Response	javax.servlet.ServletResponse javax.servlet.jsp.JspWriter	<CFOutput>
Session	Session	javax.servlet.http.HttpSession	<CFAppl ication> <CFCookie>
Application	Application	javax.servlet.ServletContext	<CFApplication>
Server	Server	javax.servlet.ServletContext javax.servlet.ServletConfig	<CFRegistry>
Error	ASPError	java.lang.Throwable	<CFError>

Table 3.2: ASP, JSP and CF Object Models Mapped to the Canonical Object Model

3.3.3 Process View

The Process view presents the concurrency and distribution of process in the application. A user of a web application interacts with the web browser; (s)he never interacts directly with the server. The browser relays her/his clicks on links and images to the server as HTTP requests. The user interface of a web application is the web browser, which can only display HTML pages. Thus, the web server must always reply to the browser's request with HTML pages. A web application developer needs to personalize the returned HTML page to reflect the user's previous requests or preferences. For example, the application may change the layout of its user interface depending on whether the user is using a desktop or a cell phone browser. The application may also show personalized special offers or reminders the to user. To enable the personalization of web applications, two dynamic techniques are used:

1. *Offline technique*: The Offline technique is used primarily for Data Intensive web applications. Instead of dynamically generating a page for each user, the user population is divided into large subgroups and all possible pages for the sub groups are generated ahead of time. When a user requests a page, the web application serves the page from the already generated pages in her/his group. This technique permits the application to support a large set of users, as the pages are not generated dynamically for every user. It is used in web sites such as *NYTimes.com* where the pages are generated once daily and all users access the static pages when reading the newspaper. *CNN.com* employs this technique but the pages are generated more frequently (for example once

an hour or whenever breaking news occurs).

For this technique, a large number of processes are executed a head of time to pre generate all the pages of the application. A smaller number of processes are used to serve the pre-generated pages.

2. *Online technique*: The Online technique is used for highly dynamic applications where the users cannot be broken into large subgroups. For example, an application that provides email for its user cannot break its user population into groups larger than one, due to the highly personal nature of email. When a page request arrives, the page is dynamically generated and sent back. Such technique permits highly personalized pages but has a high overhead. To overcome this shortcoming, caching technique are often used to reuse previously generated pages; whenever possible. For example, an email web application may cache the application page that lists all the email messages in the user's inbox instead of generating the page every time it is requested.

   For this technique, a large number of processes are used to generate on demand all the pages requested by the client.

Many web applications do not use strictly one of these two techniques; instead they use a combination of both techniques. For example, a page that contains system news can be generated using the *Offline technique*, and the rest of the email application can be generated using the *Online technique.*

3.3.4 Logical View

The Logical view provides a high level abstraction of the system based on the domain of the problem. Diagrams are used to represent the different component in the system and the interactions between them. At the architectural level, web applications can be divided into 3 tiers: *Presentation Logic*, *Business Logic*, and *Database* tier. Figure 3.7 shows the different type of architecture styles used in web applications.

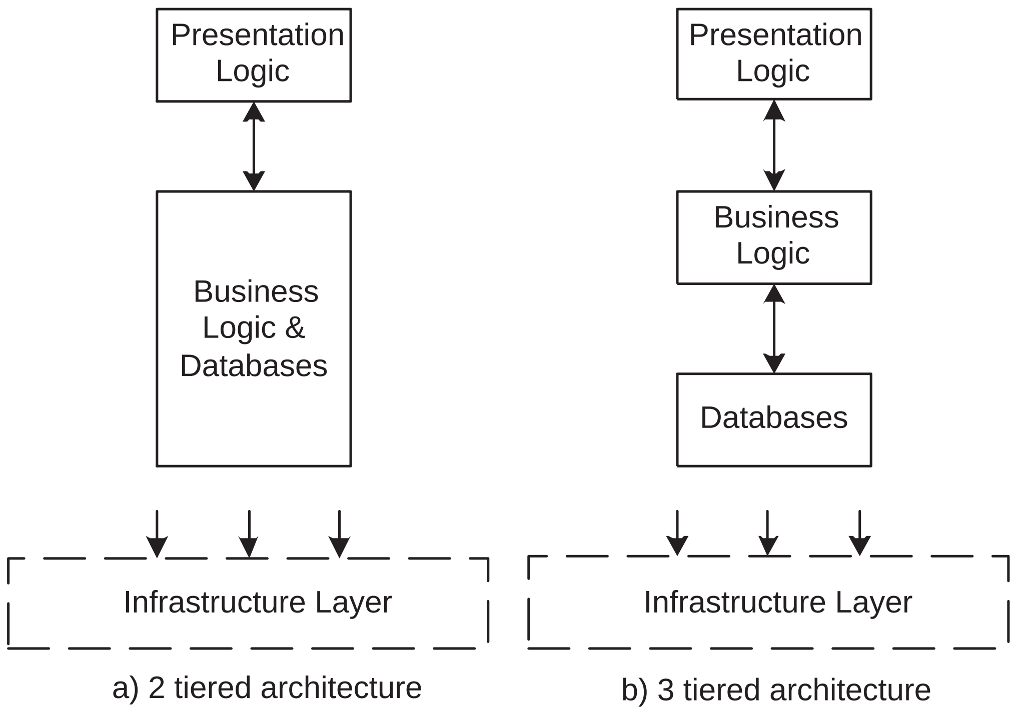


Figure 3.7: High Level Logical View of a Web Application

The *Presentation Logic* tier shows the interaction between the components responsible for the generation of the User Interface (for example: the layout of the buttons and forms for the web pages used to buy books or to view customer comments on the books in a bookstore application). Components in this tier communicate only with the *Business Logic* tier.

The *Business Logic* tier contains all the knowledge required to modify the data components that are contained in the *Database* tier. In a bookstore application, for example, this tier would contain components that are responsible for validating customer credit card numbers, verifying that books are in stock and reordering books if needed.

The *Database* tier contains all the components that are used to provide persistent storage for the application data, such as customer addresses or product inventories.

The 3-tiered architecture provides a good separation of concerns for large web applications, but some applications do not require a separation between the *Business Logic* and *Database* tiers. For example, some *Service oriented* applications do not have a clear separation between the *Database* and the *Business Logic* tier, both tiers are combined together. In this situation, a 2-tiered architecture is used. For both architecture styles, an *Infrastructure* layer exists to provide support for the basic functionality needed by the different tiers, such as access to the local file system.

3.3.5 Security View

Whereas traditional applications are composed of a modest number of centralized components, web applications have a massive amount of distributed communicating components. All these components live on an unprotected network (the Internet), which is accessible by anyone unless some type of access control and protection is in place. The *Security* view reflects the application's security and access control aspects. It shows the authentication and security levels needed to access the different component of the application. Figure 3.8 shows the *Security* view for a bank application. The bank application provides multiple functionalities for the bank's customers, employee, partners and administrators. Employees can manage their benefits and can access customers' information to assist them in choosing the most appropriate investment solutions. Bank partners such as insurance companies or credit card companies can gain access to customer's information and provide special customized offers to them. Customers can access their account. Also, the application provides a listing of all the products and services provided by the bank.

The *Security* view specifies the authentication method used: username and password authentication or a digital certificate. In addition, the view details when and how encryption will be used. For example, to access the "Employee Benefits" the employee only needs to authenticate her/him self using a username/password and the data will be transmitted in the clear unencrypted, as the employee can only access these pages through the company's private network. On the other hand, customers must authenticate themselves using a certificate and all the data transmitted during their use of the application must be encrypted as they are using the application over the Internet. In addition, the employees must authenticate themselves using digital certificates to access costumer information. No authentication or encryption is needed to access the "*Services Listing*".

Figure 3.8 shows the authentication and encryption facets of the application at

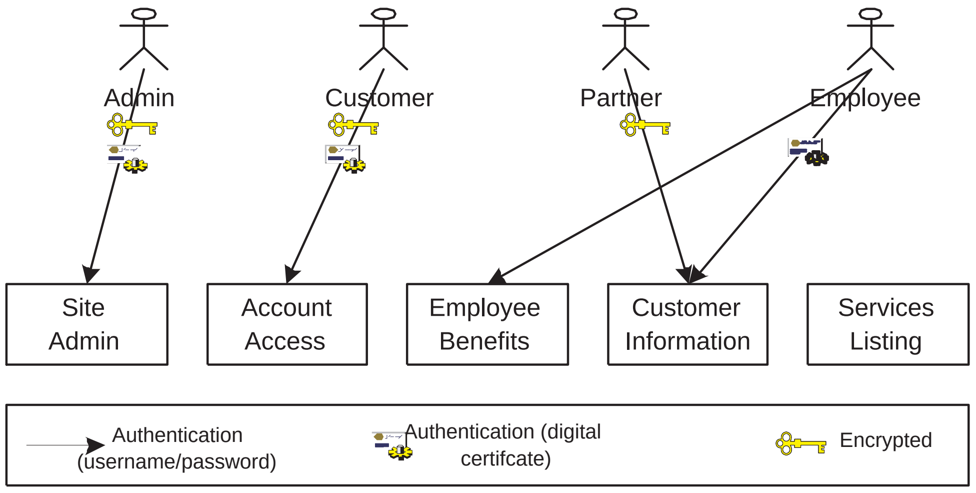


Figure 3.8: Security View for a Bank Web Application

the highest level. It is the results of the aggregation and abstractions of lower level subsystems. Figure 3.9 shows an example of the decomposition of the“Account Access” subsystem. We note that the level of access shown in Figure 3.8 is the strictest level of access employed in the subsystem. Many components of the systems may support less strict access. For example, the “Help” component of the “Account Access” subsystem is accessible without any authentication or encryption.

3.4 Summary of Web Application

To sum up, web applications are complex systems that use a mixture of technologies. They are becoming more complex and are often mission critical. Developed

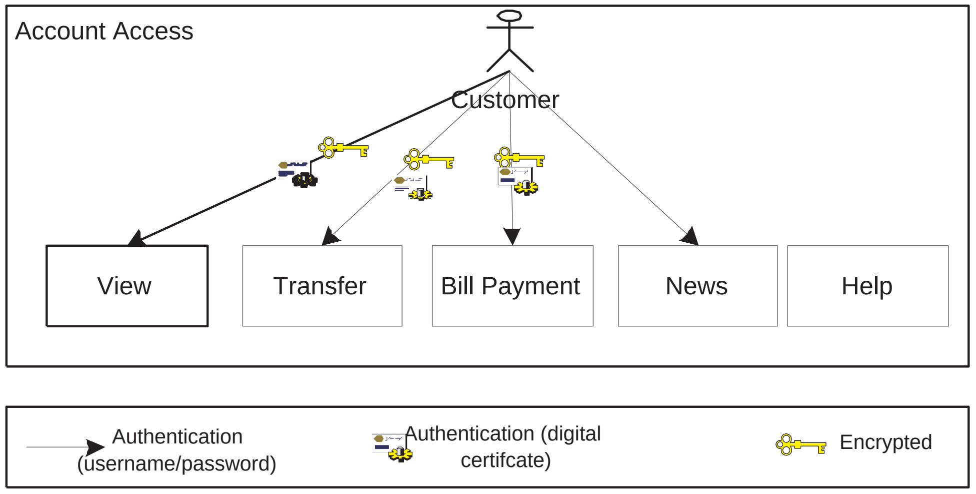

Account Access
Customer
View
Transfer
Bill Payment
News
Help
Authentication (username/password)
Authentication (digital certifcate)
Encrypted

Figure 3.9: Security View for the Account Access subsystem

using a mixture of languages glued together using scripting languages, the code base of a web application dwarfs the code base of traditional client/server systems and they are continuously evolving at a much higher pace than traditional software systems [Boo00]. Unfortunately, current web application development tools are implementation oriented with emphasis on fast one time release with no continuity and process enforcement. In the next chapter, we propose a model for understanding large web applications by recovering the design artifacts from the implementation using source code and binary extractors. We abstract the implementation details and model the applications structure using simple diagrams that convey the designer's intentions.

Chapter 4

Related Research

The study of web applications is a fairly new field but there have been significant research contributions from different groups worldwide. We focus mainly on two areas of research that are most related to our research on the architecture recovery and visualization of web applications: web engineering and the modeling of web applications.

4.1 Web Engineering

Web Engineering is concerned with the establishment and use of sound scientific, engineering and management principles to ensure the successful building and deployment of dependable web-based systems and web applications [Web99a]. Much of the Web Engineering concerns are rooted in the fields of software engineering and distributed systems engineering [Bol00]. Research in the field focuses on applying and adapting classical software engineering techniques to the web domain. Currently, the main publishing outlet for researches in the field is through workshops such as the International Workshop on Web Site Evolution [Web99b, Web00] and the International Workshop on Web Engineering [Web99a].

A key paper by Hatzimanikatis *et al.* in 1995 is the earliest publication in the field [ACD95]. The paper focuses on the development of a simple quality model to measure the readability and maintainability of hyper documents. The authors state that their main intent is to provide a methodology and approach based on classical software metrics. Other researchers have followed Hatzimanikatis *et al.* lead and recognized the similarities between the development of software and the development of a web site. 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 in code [BBH98]. 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 [RT00]. Both tools are heavily geared toward static web sites (*Brochure* sites). The crawler will fail to recognize that the changes in the web site may be due to the dynamic nature of the web application and not to actual modification of the content of the web page. For example, the tool would signal that the homepage of a site like *CNN.COM* changes continuously. In fact, the source of the homepage rarely changes. It is an application page, which retrieves updated information from a database, every time it is accessed. The approach followed by Rica, Tonella, and Brereton puts more emphasis on the user's experience and attempts to track the changes that are sensed by the end user of the application. These changes may not be mirrored in the source code of the web application or changes in the source code may not show in the crawled pages. For example, in an email service web applications, the page displaying the user's inbox may originally retrieve all the email messages from a flat file. Later, 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 all the components of a web application. Using this approach, we can study more sophisticated dynamic web applications not just brochure sites. Stated differently, our approach is a white box reverse engineering approach and their approach is a black box one.

Whereas the aforementioned approaches including our approach use tools to study the structure of large web applications, Anotiol *et al.* suggest a non-automated technique to recover the architecture [ACCL00]. 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 can be described in terms of entity types, attributes and relationships. For example in an exam booking web application, we find entities such as a student, and a course; and relations such as "takes", and "provides". As noted by the authors, their approach is best suited to *Catalogue* web applications whereas our approach can be used to recover many different types of web applications. In addition, the RMM recovered architecture is a high level view of the main concepts in the system and the relations between them. Our approach extracts the concrete architecture view of the system, based on its actual implementation. The conceptual architecture tends to remain stable across different releases compared to the concrete architecture which changes as new technologies are deployed, a common occurrence in the fast moving web application domain.

Tilley recognizes that as web sites age they suffer similar problems as suffered by traditional applications [Til]. Their structure degrades and their maintenance becomes problematic. Also, he concludes that web maintenance is harder as many of the developers of web sites do not have a computer science background. No studies have been performed on web application development or maintenance to quantify the cost of maintaining web applications.

4.2 Modeling of Web Applications

The main objective of our research is recovering and providing an up to date documentation of the architecture of the software system. We analyze the source code of the system and report the interesting relations and entities to the user through a graphical visualizer. The choice and the definition of the term "interesting" is rather vague and task dependent. For example, a developer modifying an object used by many application has a different set of interesting relations than a database administrator trying to track all the components using a specific database table. To aid developers in performing in their analysis, researchers have proposed different methods to visualize web applications. Each method emphasizes an aspect or set of interesting relations that the developers can visualize. Ceri *et al.* present the Web Modeling Language (WebML) [CFB00]. WebML provides a high level conceptual description of a web application. The language is geared towards Catalogue (data-driven) web applications. It is composed of five models:

1. A *structural model* describes the data flow in the application.
2. A *navigational model* describes the topology of links between the different pages.
3. A *compositional model* describes the files and databases that are grouped together to represent some conceptual concept.
4. A *presentation model* describes the layout of each page and its graphical requirements.
5. A *customization model* describes the different groups of users of the software and their needs.

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 different the source code objects and the call graph cannot be expressed using WebML's constructs and concepts.

Conallen's work on extending UML [Gro99] to model web applications is the most similar to our work [Con99]. Conallen defines a web application as: "a Web system that allows its users to execute business logic with a Web browser". He proceeds to point out the need to model web applications due to their complexity, and he presents the Web Application Extension (WAE) for UML. Web pages are modeled as UML components. Every web page is modeled using two different aspects:

1. Its *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. Its *client side* aspect where he shows the page's interaction with the browser built in objects and Java applets.

In our approach we focus mainly on the server side aspect of modeling web applications, because much of the client side aspect is concerned with the page's layout and presentation. 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 and still need to understand them. Currently our visualizer does not generate UML compliant diagrams; however, the architecture of our architecture recovery system enables us to plug in a UML compliant visualizer.

4.3 Summary of Related Research

In this chapter, we compared research done in the web engineering community and research done to model web applications to our approach. In the following chapter, we present our approach to architecture recovery.

Chapter 5

Architecture Recovery for Web Applications

In this chapter we present an overview of our architecture recovery framework used to generate architecture diagrams for web applications. We discuss the visualization needs of web developers and contrast their needs to those of developers of traditional software systems. Furthermore, we introduce a domain model (*schema*) for web applications. Finally, we explain the different extractors we developed to recover the architecture of web applications and we detail the type of relations that these extractors emit. Each parser generates facts that conform to parts of the presented schema for web applications. Once the facts emitted by all our parsers are combined, the combined facts conform to the whole schema for web applications.

5.1 Recovery Process Overview

We use a semi-automated process to recover the architecture of web applications. The process uses tools/extractors to analyze 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) [Pen92, FHK$^+$97, PBS] environment to address the differences between web applications and traditional software applications. Previously, the PBS system was used successfully to recover the architecture of many large traditional procedural systems such as the Apache web server [HH00], the Linux kernel [BHB99] and the VIM text editor [TGLH00]. The architecture of large object-oriented systems such as the Jigsaw web server [HH00] and the Mozilla web browser [Lee00a, GL00] have also been recovered using PBS.

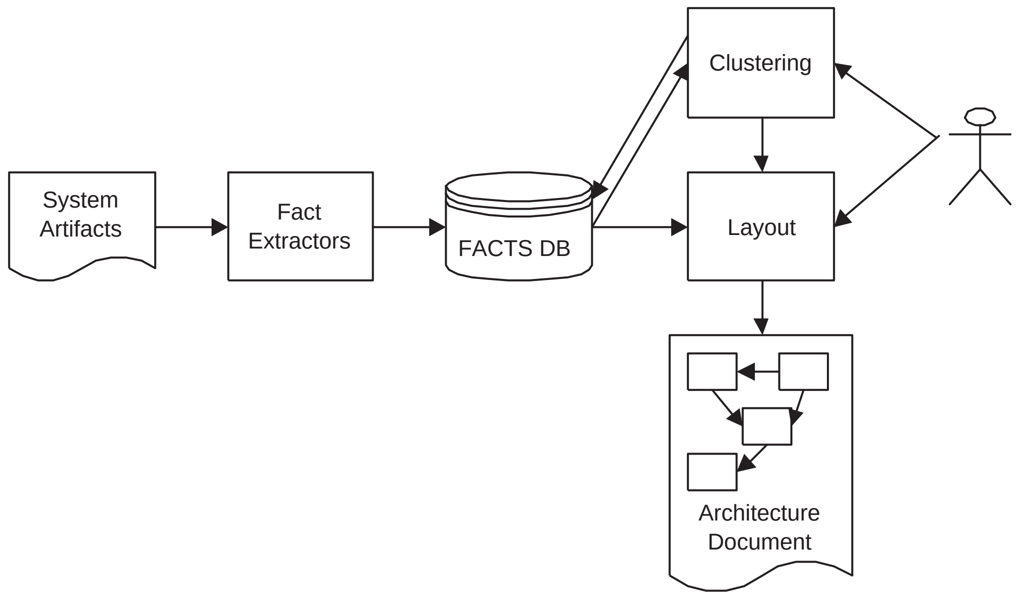


Figure 5.1: Overview of the Portable BookShelf environment

Figure 5.1 shows an overview of the PBS environment. First the artifacts of the software system (such as source code, documentation, execution traces, 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 [Hol97, Hol98] using a specific schema or domain model. The schema permits many tools to operate independently on the extracted facts and reduces the coupling between the fact extracting and the fact consuming tools such as the visualizer. Furthermore, the use of schema permits the integration of different types of facts to produce a single architecture document that contains all the extracted facts.

Once the facts have been produced, a "*first-cut*" attempt to visualize them would lead to an architecture view which resembles Figure 5.2. 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 file of source code, 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

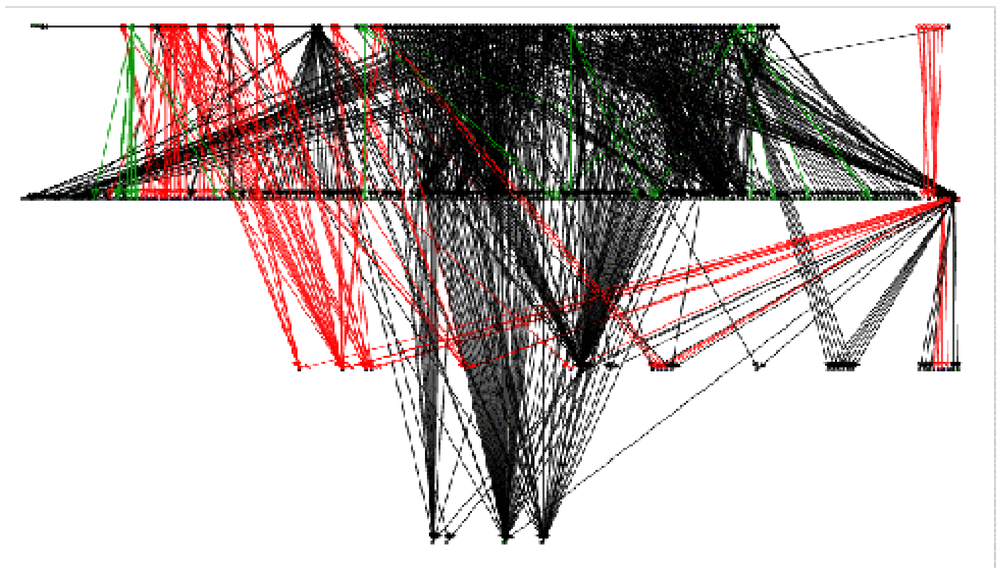


Figure 5.2: Un-clustered Architecture View

of the software system into smaller meaningful subsystems. Figure 5.3 shows the reduction in complexity achieved by decomposing a software system into 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 [TH96, TH98, Bow99]. 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.

Later, an automatic layout tool processes the stored facts to generate diagrams such as the one shown in Figure 5.4. 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 mentioned process combines

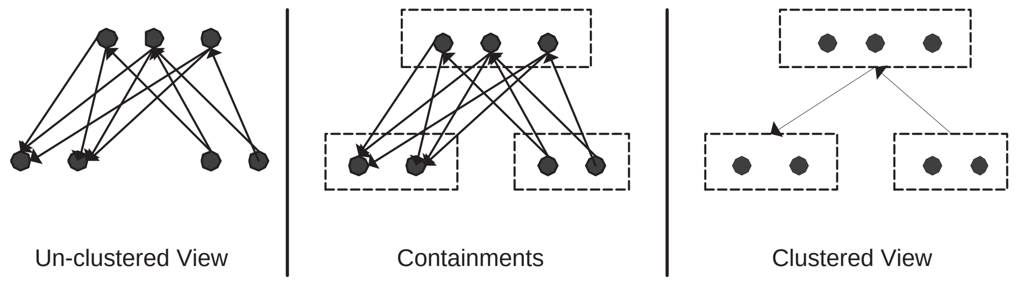


Figure 5.3: Clustering of Software Components

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.

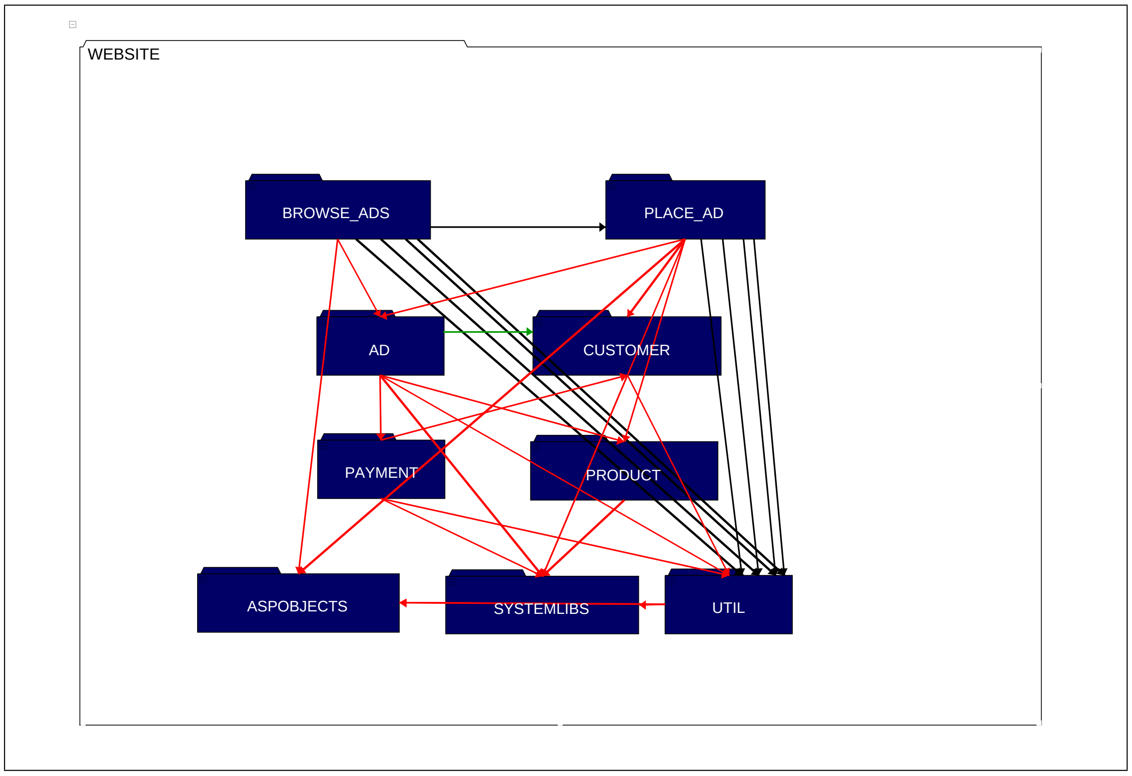


Figure 5.4: Clustered Architecture Diagram of a Web Application

5.2 Visualization Needs for Web Applications

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 more concerned with the system-level topology of the components and their interaction rather than the internal structure of each component in a web application. 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.

Previous studies in program maintenance and understanding conducted on the development of traditional software systems [LA97, Sim98, SCH98, SCHC99] assisted us in determining the needs of developers. The recovered web application architecture must satisfy developers needs to be of any use for them, for example:

1. Developers use visualization tools to pinpoint locations of interest in the system's code; they later delve into these locations of interest to improve their understanding using their code editors. In response, our recovered architec-

ture diagrams for web applications do not show all the detailed relations and components, instead they present an overview of the system. For example, they do not show the internal structure of code inside the same file. They show the inter-file relations. Also we use containment and information hiding techniques to reduce the complexity of the visualized systems.

Traditional Software	Web Applications
Function definition	Object definition
All uses of a function	All uses of an object
Variable definition	Database table schema
All uses of a variable	All uses of a database table

Table 5.1: Mapping from Traditional Applications to Web Applications

2. When searching 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. We adapted these relations to web applications, as shown in Table 5.1. Web applications are composed of multiple stateless pages and data must be stored in a database or in a persistent object so different pages can share them. We show the relations at a higher level of abstraction. Instead of showing variables, functions and their interrelations, we show database tables, distributed objects, and their interrelations.

5.3 Domain Model

Before we describe the architecture recovery of web applications, we define a domain model (*schema*). The schema describes the permitted entities and permitted relations between them. Fact extractors are expected to adhere to the schema by generating facts that conform to it. In addition, tools that manipulate the facts must generate schema conformant facts, which are stored on disk in TA format.

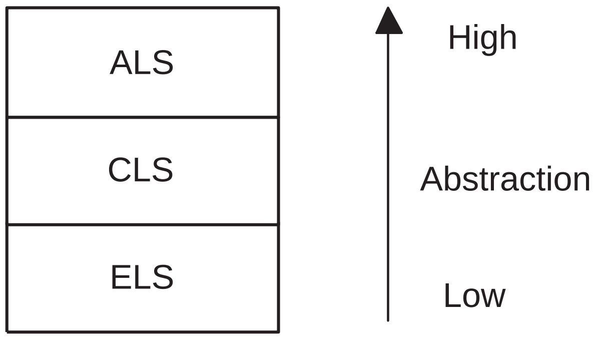


Figure 5.5: Layers of Schema

Our domain model is composed of three layers of schemas (see Figure 5.5). The level of abstraction ranges from language specific entities which are too detailed and too close to the source code level and, to more general architectural level entities.

1. *Architecture-Level Schema (ALS)*: The ALS describes the permitted relations between the architecture elements which are subsystems and components. The ALS is language (such as C, C++, or Java) independent and technology (such as COM, DCOM, or CORBA) independent.

2. *Component Level Schema (CLS)*: The CLS describes the permitted relations between the different components of a web application such as Active Server

Pages (ASP), database tables, binary libraries and distributed objects.

3. *Entity-Level Schema (ELS)*: The ELS describes the permitted relations between the top-level program entities such as functions, variables, user objects, database tables and distributed objects; instead of low-level entities such as statements and expressions. The ELS is language dependent.

The different levels of schemas enable developers to study the 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 execute *drill down* [CD97] operations to investigate anomalies in their architecture at the source code level such as the existence of unexpected dependencies between subsystems. Using *roll up* operations, developers can trace up the effects of source code changes on the overall architecture of the software system [Lee00b].

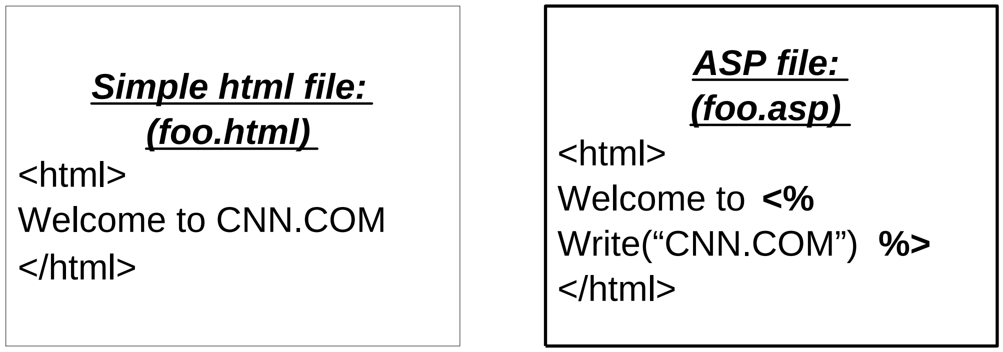
In our case studies, we show how developers can use the formalism, developed by Holt [Hol96, Hol98] and implemented in the PBS tool, to perform impact analysis studies on web applications for migration and maintenance purposes.

5.3.1 Overview of the Schema Layers

In the following sections, we detail the different layers of schema and show examples of the schemas for web applications. We present the ELS schema for the VBScript and JavaScript languages. Then we show how to combine both schemas into a common schema, called the Common ELS for Object-Based Languages. Once we combine the different languages into a common schema, we proceed to explain the CLS schema. The CLS schema reduces many of the details available in the ELS schema. This reduction of detail, commonly referred to as *lifting*, is performed again when we introduce the ALS schema. Later, we explain how our various extractors emit relations that conform to these schemas.

5.4 ELS: Entity Level Schema

An Entity-Level Schema specifies the relations that occur between the program entities according to the syntax and design of the language. The ELS models the whole software system as a flat space without indicating the distribution of entities across files or components. At a higher level, the CLS indicates the relations at the component level instead of just at the entity level. Web applications components are developed in a variety of languages such as Java [GJS96], C [KR98], C++ [Str97], Perl [WCO00], Visual Basic [Cor98], VBScript [Hil96] and JavaScript [KM98]. The ELS for Java, C, and C++ are well known, for more details refer to [Tra99].


Simple html file: (foo.html)

```
<html>
Welcome to CNN.COM
</html>
```

ASP file: (foo.asp)

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

Figure 5.6: An Active Server Page vs an HTML Page

In our work, we address the VBScript and JavaScript languages because they are the most used languages in the development of web applications. Both languages are used extensively in Active Server Pages (ASP). Each ASP page is an independent component that communicates with the rest of components shaping a web application. ASP pages are HTML pages that allow developers to embed segments of control code in HTML pages. The code segments are delimited with special tags (shown in bold in Figure 5.6). The application server preprocesses the ASP page and executes the code segments when the browser requests the page. The results of the execution replaces the actual code in the ASP page and are sent, along with the HTML, back to the requesting browser. The code can be written in JavaScript, VBScript or Perl; and the execution may involve the access of many distributed objects and databases, as shown in Figure 5.7. We now specify the ELS for the VBScript and JavaScript languages.

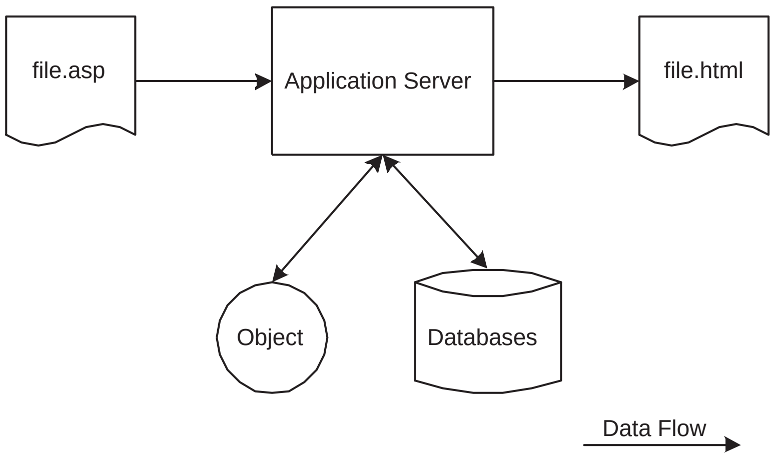


Figure 5.7: Active Server Page Data Flow

5.4.1 An ELS for VBScript

The VBScript language is a subset of the Visual Basic language developed by Microsoft Corporation [MS]. It is a scripting language that is interpreted and runs primarily on the Microsoft Windows platform. Programs written in VBScript can execute inside of a web browser (as client scripts) or inside the application server (as server scripts).

Client scripts have more security restrictions on access to local resources than server scripts. Client scripts are used mainly for aesthetic web page layout and simple data entry verification.

Server scripts are used extensively to glue distributed objects (COM[1] Objects[2] [COM]) together and to provide access to persistent data stored in relational databases.

The VBScript language is weakly typed with only one data type (`Variant`). It supports procedural and object-oriented programming paradigms, but it is not a true object-oriented language. Instead, it is an object-based language that permits developers to specify objects (in terms of methods and properties) but does not support other object-oriented features such as inheritance.

The ELS for VBScript is shown in Figure 5.8. To improve the readability of the

[1]Component Object Model is a component software architecture that allows applications and systems to be built from components supplied by different software vendors. CORBA Object and Enterprise Java Beans (EJB) are other types of component softwares.

[2]A COM Object or COM component is not an object in the sense of source code object-oriented programming objects such as those defined in C++ or Java. Instead, it is some piece of compiled code that provides a service to the rest of the software system. This piece of compiled code would define an interface to access its functionality. This interface defines the component's class. Each instantiations of the component class is a component object.

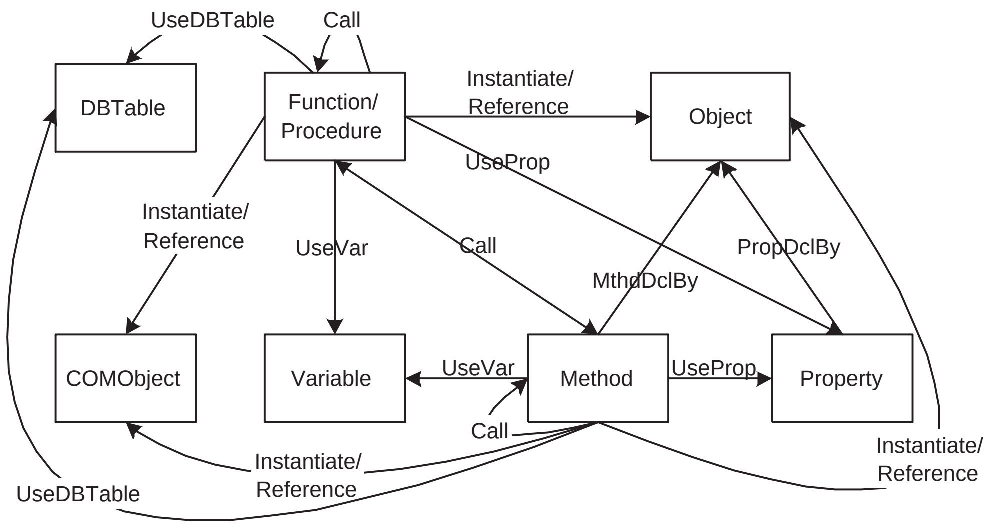


Figure 5.8: Entity Level Schema for the VBScript Language

figure, the `ASPFile` entity has been removed and all the relations associated with it. The omitted relations are: `Object ObjDefBy ASPFILE`, `Function FuncDefBy ASPFile`, and `Variable VarDefBy ASPFile`.

From the figure, we see that an `Object` can have `Properties` and `Methods`. The language supports `Functions`, `Procedures` and `Variables`. All of which can interact together and with `Objects`. Scripts can instantiate and reference many of the built in `Objects` in the language. They can also reference and instantiate their own VBScript objects or other objects (`COMObjects`) that have been written in other languages. The latter feature enables VBScript to act as glue to link objects and components developed in different languages. The language provides APIs to update and retrieve data stored in relational databases (`DBTables`).

The VBScript language is not as well documented as the JavaScript Language. No extensive reference manual exists that describes the language in detail. To develop an extractor for VBScript, we examined many systems and code samples written in VBScript to gather the language's structure and syntax.

5.4.2 An ELS for JavaScript

JavaScript, also called ECMAScript is an object-based scripting language; it is not a subset of any other language. It shares many similarities with object-oriented languages such as the Java and C++ languages. The language is developed under the control of the ECMA [ECMa] standardization organization and is called ECMAScript [ECMb]. JavaScript is the name of Netscape Corp.'s [NS] implementation of the ECMAScript language whereas JScript is Microsoft Corp.'s name. Both implementations are supersets of the ECMAScript language; they provide extensions specific to the implementing company. The extensions add many built-in objects and provide mechanisms for the language to interact more easily with other web components such as `COMObjects` and `DBTables`.

The language can be interpreted or compiled. The language has many built-in `data types` such as `String`, `Number`, `Boolean`, `Object`, `Array`, `Null`, and `Undefined`. New types cannot be defined. The language is loosely typed. The data type of variables do not need to be defined ahead of time and conversion between the different data types is done automatically without the need of a cast operator. The data type of a variable is based on the value of the variable at run-time. JavaScript supports the dynamic evaluation of code segments stored in a variable. The latter two features hinder the accuracy of statically generated facts, as some relations cannot be determined till run-time.

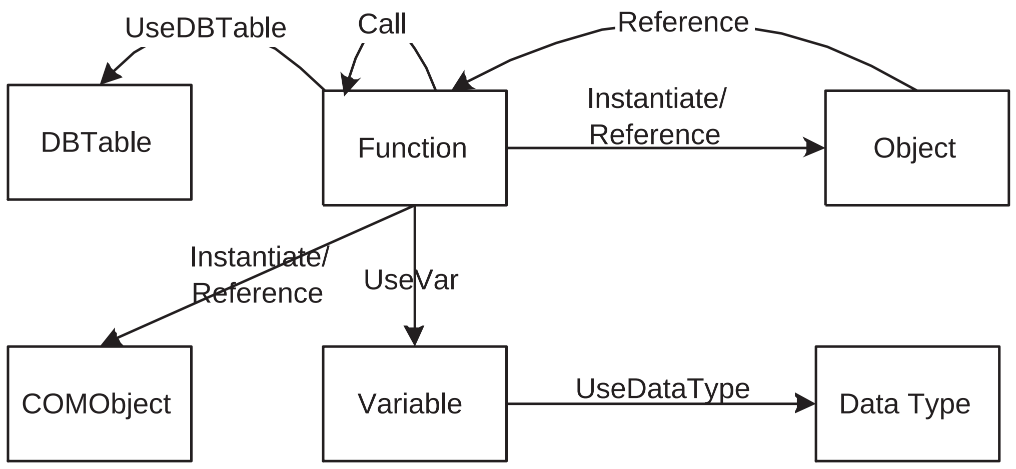


Figure 5.9: Entity Level Schema for the JavaScript Language

Figure 5.9 shows the ELS for the JavaScript language. Again to reduce the clutter in the schema diagram, the `ASPFile` entity has been removed and all the relations associated with it. The omitted relations are: `Object ObjDefBy ASPFILE`, `Function FuncDefBy ASPFile`, and `Variable VarDefBy ASPFile`.

5.4.3 A Common ELS for Object Based Languages

All the web applications we studied are developed to run on the Microsoft Windows platform and use a mixture of VBScript and JavaScript to glue the different components. To enable the reverse engineering of these systems, we use a common Entity Layer Schema.

The Common ELS combines the JavaScript and the VBScript schemas into a single schema, as shown in Figure 5.10. This schema raises the abstraction to a higher level independent of a particular programming language. 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 the figure, the ELS uses inheritance relations to indicate the mapping between the common schema and the language specific schema. For example, a `UserObject` entity is a super class of VBScript `Object` and JavaScript `Object`.

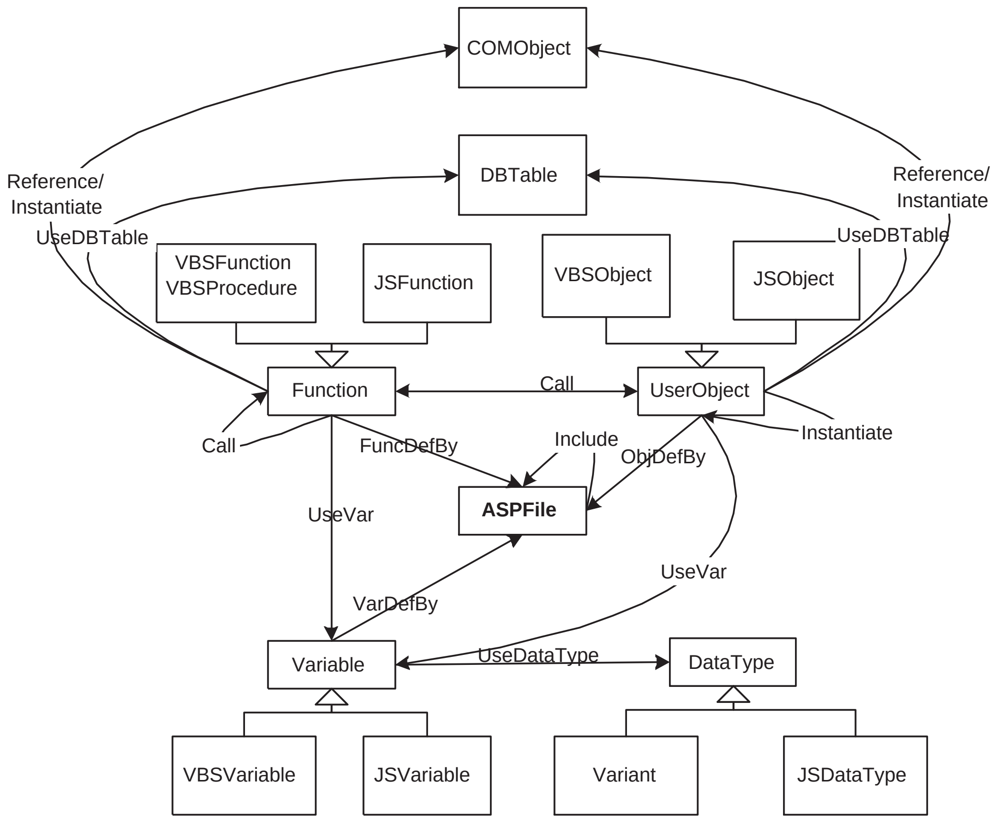


Figure 5.10: A Common Entity Level Schema for Object Based Languages

5.5 CLS: Component Level Schema

Various components shape a web application. The components are glued together using scripting languages such as VBScript and JavaScript. When developers examine the architecture of web applications, they are more concerned with the interactions between the different components rather than the internal design or architecture of the components themselves. The Component Level Schema raises the level of abstraction of the extracted facts from the internal structure of the components to a higher level.

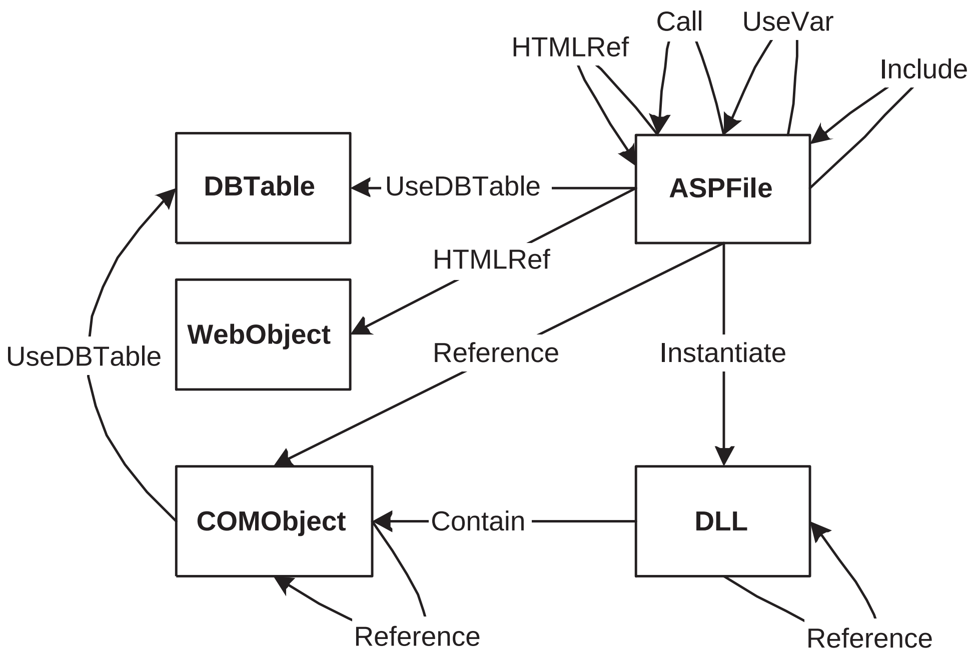


Figure 5.11: Component Level Schema for Web Applications

Figure 5.11 shows the CLS, which contains the various components of a web application and their interrelations. As can be seen in the figure, an `ASPFile` component can include other `ASPFiles`. The `ASPFile` can contain HTML code that references another `ASPFile` or a `WebObject` such as a picture, a movie, or an audio file. It can also contain code segments written in JavaScript or VBScript that use a variable, instantiate an object or call a function defined in another `ASPFile`. The code segments can read, update, and insert data in database tables. The code segments can instantiate or reference a `COMObject` that may reside inside a DLL[3]. A `COMObject` can in turn reference other `COMObjects` or it can access a database table (`DBTable`).

5.6 ALS: Architecture Level Schema

We use boxes and arrows to depict the architecture of large system. The boxes can be components such as those specified in the CLS schema or could be subsystems that contain in turn other components or other subsystems. The ALS for web applications is described in figure 5.12. For a web application the components are entities such as `DLLs`, `ASPs`, `DBTables`, `WebObjects`, and `COMObjects`. Many dependencies between the components exist: data dependencies that are caused by components accessing data stores, control dependencies that show the control flow and activations of many components, and web dependencies that show the hyperlink structure between the different structures. We found that viewing the architecture of applications at the component level is hard to understand (see Figure 5.2), because of the large number of components and the large amount interactions between them.

[3]DLL: a Dynamic Link Library (DLL) is a shared library on the Microsoft windows operating system.

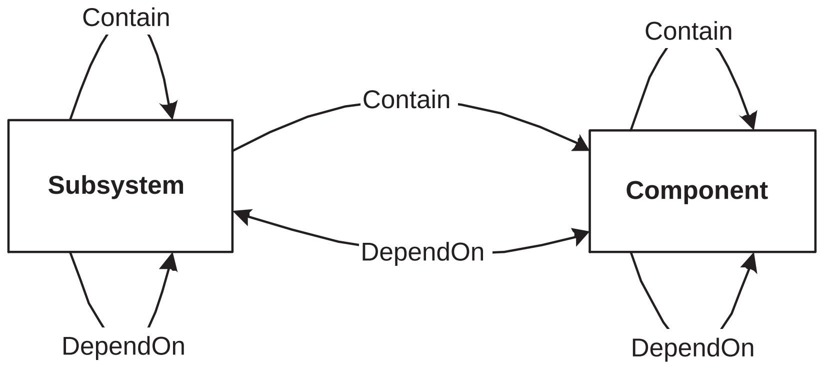


Figure 5.12: Architecture Level Schema for Web Applications

We used the concept of a subsystem to reduce the complexity of the architecture of large web applications into a more comprehensible structure. A system decomposition tree is created with the whole system being at the root of the tree and components at the leaf level. Each component is clustered inside a meaningful subsystem in the tree and subsequently each subsystem is clustered into another subsystem until we reach the root of the tree. For web applications, we have clustered most of the recovered architectures using the directory structure of the source of the web application and file naming conventions. For example, all components whose name starts with the word "customer" would be placed in a customer subsystem. When both mechanisms failed (*e.g.* the application used a flat directory structure), we examined the documentation of the components to get a better understanding of its functionality and determine its appropriate subsystem.

Once the clustering phase is done, the extracted facts and the clustering information are combined and processed by the **Grok** tool. The **Grok** tool is a relational calculator. It raises the extracted facts to the appropriate level of abstractions, inter-subsystem instead of of inter-component. It determines the type of relations that exist between subsystems based on the type of relations between the components that exist in a subsystem.

The output of **Grok** is in TA format and is a textual description of the interrelations between the subsystems and the components. The textual description is not easy to display and comprehend because of the large number of relations and entities involved. Although **Grok** provides many powerful functions to perform analysis of the recovered architectures, a good understanding of the mathematical background [Hol96, Hol98] used by the tool is needed. To ease the architecture analysis and understanding process, the textual output of **Grok** is processed by an automatic layout tool that organizes the facts for a graphical output. A visualization tool [Pen92] is used to view and analyze the recovered architecture. The visualization tool has a much lower learning curve compared to **Grok**'s mathematical textual language.

5.7 Fact Extraction

In traditional architecture recovery, an extractor parses the source files of the software system and emits facts about the system. Many of the traditional software systems are developed in a single programming language and all the source code of the system is available. For a large number of traditional software systems, only one language specific extractor is needed.

Extractors range from lightweight extractors that search for specific patterns of interest in the source code and emit the relevant facts to more detailed parsers.

Such parsers may be modified compilers that emit facts about the source code instead of producing assembly code or binaries. Researchers have been successful in building binary extractors that analyze the program executables and libraries of a software system to understand the dependencies in the system, when a source code extractor for the language is not available [ITBH99].

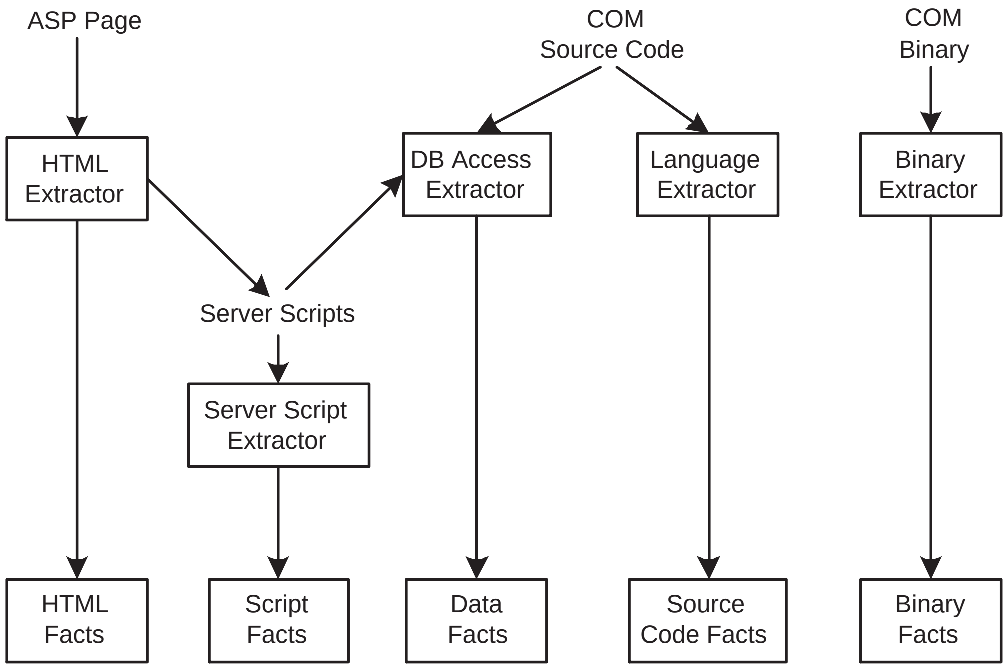


Figure 5.13: Conceptual Architecture of the Fact Extractors

On the other hand, web applications are developed using a variety of languages 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 depend on a single extractor. Our approach uses a set of extractors that cooperate to emit all the facts from the examined web application. Figure 5.13 shows an overview of the various extractors and their input and the type of facts generated by them. We used 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. Each extractor generates facts that conform to the CLS schema for web applications. Once all the facts are emitted, the clustering information is combined and all the data (facts and clustering) are processed by **Grok**, then a layout tool. The output of the layout tool can be viewed and analyzed using a visualizer.

The directory structure of the web applications and the source code directory are crawled by a shell script. 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, then 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 all the previously processed directories and consolidates all the generated files into a single file (`THEFACTS` file). The `THEFACTS` file is combined with the clustering information that is generated using the directory structure and user input. We detail the types of relations generated by each type of extractor in the following sections.

5.7.1 Overview of the Extractors

For each described extractor, we provide a table which provides a detailed description of all the types of relations generated by the extractor. For example, the CLS schema in figure 5.11 contains a relation called `HTMLRef`. The HTML extractor generates fifteen relations that are of type `HTMLRef`. Similarly the DB Access Extractor generates four relations that are of type `UseDBTable`. We decided to show in the schema diagrams just one relation that represents the type of all these relations to ensure the readability of the schema diagrams. To get a detailed description of the generated relations, the reader needs to refer to the relations tables.

5.7.2 HTML Extractor

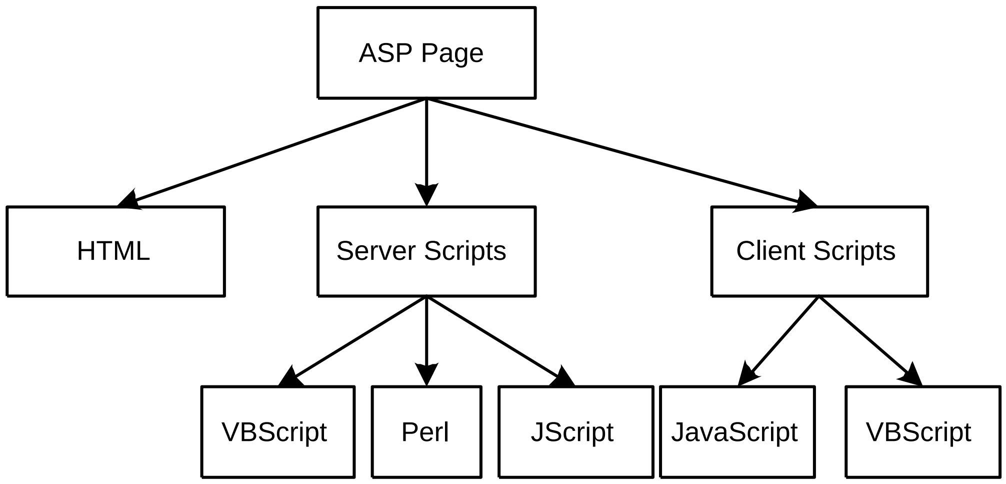


Figure 5.14: The Internal Structure of an ASP component

The HTML Extractor is responsible for processing HTML and ASP files. An HTML page is an ASP page that contains no code segments and is processed only by the HTML Extractor. An ASP page contains various sections. Each section is parsed and analyzed by the appropriate extractor. An ASP page contains (see Figure 5.14):

1. HTML sections that are text with links to other online content. The HTML sections are sent to the requesting browser, without modification by the server.

2. Server scripts that are executed on the server and the result of the execution is sent back to the requesting browser.

3. Client scripts that are interpreted inside of the user's browser. The Client scripts are sent to the requesting browser without modifications by the server.

Each section is written using a different language:

1. HTML Sections are written in HTML.

2. Server scripts are written in either VBScript, Perl or JScript.

3. Client scripts are written in JavaScript or VBScript.

Figure 5.13 shows that the HTML extractor performs two roles. It first emits relations that exist in an ASP/HTML page. Table 5.2 shows the generated relations, where Entity A is the ASP page being processed. The generated information permits the visualization of the hyper/web references between the various pages.

Once all the HTML facts are emitted, the HTML extractor recognizes the other type of sections that are in an ASP page and outputs them to different files, which have the same name as the currently processed ASP page but have different suffixes based on the type of the section and the language used. For example, if the HTML Extractor were to process a file called foo.asp, it would emit the HTML facts and output the following files: `foo.asp.vb_server`, `foo.asp.js_server`, `foo.asp.vb_client` and `foo.asp.vb_client`. The newly created files that contain the client script (`foo.asp.vb_client` and `foo.asp.vb_client` are not processed. But the newly created files that contain the server scripts (`foo.asp.vb_server`, `foo.asp.js_server`) are processed by the server script extractor.

Relation	From (A)	To (B)	Description
background_url	ASPPage	WebObject	Entity **B** specifies the background of entity **A**.
file_include		ASPPage	Entity **A** includes all the contents of entity **B**.
fontdef_url		WebObject	Entity **B** contains the font description for the font used in entity **A**.
formget_url		ASPPage, DLL	Entity **B** is responsible for processing the form fields that are in entity **A**.
formimage_url		WebObject	Entity **B** is the image that the user has to click after they fill the form in entity **A**.
formpost_url		ASPPage, DLL	Entity **B** is responsible for processing the form fields that are in entity **A**.
frame_src		ASPPage	Entity **B** specifies the contents of one of the frames in entity **A**.
image_lowsrc		WebObject	Entity **B** specifies the low resolution image that is displayed in entity **A**.
image_src		WebObject	Entity **B** specifies the image that is displayed in entity **A**.
imagemap_src		WebObject	Entity **B** specifies the image that is displayed in entity **A** for an image map.
layer_src		ASPPage	Entity **B** specifies the contents of a layer of entity **A**.
link_url		ASPPage	Entity **B** specifies the background of entity **A**.
refresh_url		ASPPage	Entity **B** replaces entity **A**.
stylesheet_url		WebObject	Entity **B** specifies the layout of the contents of entity **A**.
virtual_include		ASPPage	Entity **B** includes all the contents of entity **A**.

Table 5.2: Relation Types Produced by the HTML Extractor

No Analysis for Client Scripts

In our architecture recovery we ignore client scripts in ASP pages. The scripts are never processed by our tools. We chose not to analyze client scripts as they are used mostly for aesthetic page layout and simple input checking. Client scripts do not interact with any of the component that reside on the server, instead they interact with components provided by the web browser.

Other architecture diagrams may be generated to show the client script interactions with the browser object but we believe that such architecture diagrams are not as useful for understanding the big picture of large web applications because of the limited interactions of client scripts and their simplicity.

Network Crawling vs. Directory Crawling to Extract Hyper Facts

All the relations shown in Table 5.2 except (the `file_include` and `virtual_include` relations[4]) can be recovered either by crawling the web site directory structure as we do, or by crawling the web site using a network crawler such as the ones used by search engines to index web sites. The network crawler won't find any files that are not accessible from the crawler's starting page. This is not a concern for the directory based crawler as all the present files are examined regardless whether they are referenced by other files or not. Our approach permits the developer to determine dead files that are no longer referenced by other files and which can be removed from the code base. Such a file would show up in the generated architecture diagram with no relations from other files to it (no in-arrows). If links to

[4]The `file_include` and `virtual_include` relations are preprocessed by the web server to include the appropriate file before sending the page back to the requesting browser/crawler.

other pages are generated dynamically then directory crawling won't extract the links but network crawling may or may not extract the links based on its starting page.

Some of the facts generated by a network crawler will not match the relations generated by a directory based extractor as the preprocessing performed by the web server may cause the generation of relations that do not match the actual HTML code. For example if a file includes another file, a network extractor will mistakenly assign all the relations from the included file to the including file.

5.7.3 Server Script Extractor

Server Script Extractors are responsible for processing the server script code segments that are located in an ASP page. As per the example in the previous section, this extractor would process the following files: `foo.asp.vb_server` and `foo.asp.js_server`.

Three different extractors are needed, one for each implementation languages (VBScript, JavaScript, and Perl). For our work we did not develop a Perl extractor as all the systems we analyzed did not use Perl. We developed a light weight extractor for JavaScript and a full extractor for VBScript. Both extractors examine the source code of the scripts and emit the relations shown in Table 5.3.

5.7.4 Database Access Extractor

The Database Access Extractor uses regular expressions to locate data table accesses inside source code statements. As show in figure 5.13, this extractor takes

Relation	From (A)	To (B)	Description
asp_dynamic_ref	ASPPage	COMObject DLL	Entity **B** is instantiated by a script inside of Entity **A.** (For example, Var v = CreateObject("B");)
asp_static_ref	ASPPage	COMObject	A script in Entity **A** references entity **B**. This relation is often generated when a script references built-in objects. (For example, B.doSomething();)
contain	DLL	COMObject	Entity **B** is part of entity **A**.

Table 5.3: Relation Types Produced by the Server Script Extractor

as input the server scripts and the source code of the components.

The extractor searches for matches that resemble common database access functions and SQL keywords such as `SELECT` or `INSERT`. The extractor then employs some 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 matches are coincidental and no database table access has occurred. Table 5.4 shows the different types of relations extracted by the Database

Relation	From (A)	To (B)	Description
db_select	ASPPage, COMObject	DBTable	Entity **A** performs a database select operation on entity **B.**
db_update	ASPPage, COMObject	DBTable	Entity **A** updates elements stored in entity **B**.
db_insert	ASPPage, COMObject	DBTable	Entity **A** inserts new elements into entity **B.**
db_proccall	ASPPage, COMObject	DBTable	Entity **A** calls a stored -procedure whose implementation is defined and stored in entity **B.**

Table 5.4: Relation Types Produced by the Database Access Extractor

Access Extractor. Because of the use of heuristics, the output of this extractor is reviewed manually to validate it. All the server scripts in the ASP pages and the source code files for all the `COMObject` are examined by this extractor to recover databases accessed in the web application.

The Database Access Extractor uses many heuristics and has been tuned to work well for the systems we studied. For other systems, we expect that the performance of the extractor won't be as good as our results. Due to the pipeline architecture of our extraction process, user intervention is possible to correct the extractor if it fails to recognize database accesses or if it mis-recognizes database accesses.

Relation	**From (A)**	**To (B)**	**Description**
dll_dynamic_ref	COMObject	COMObject DLL	Entity **B** is instantiated by entity **A.** (For example, Var v = createInstance B();)
dll_static_ref		COMObject	Entity **A** references entity **B**. This relation is often generated when code uses built-in libraries or objects. (For example, FileSystem.readFile())
contain	DLL	COMObject	Entity **A** becomes part of entity **B** once compiled.

Table 5.5: Relation Types Produced by the Language Extractor

5.7.5 Language Extractor

The Language Extractor could be a full extractor or a light weight extractor that searches the source code of the `COMObject` for interesting statements. We developed a simple lightweight extractors for `Microsoft Windows C`, `Visual Basic` and `Microsoft Windows C++`. All these extractors use Perl regular expressions to generate the relations shown in Table 5.5. For the examined web applications, this extractor was very successful in extracting the facts without any manual intervention. We expect it to work for web applications different than the studied ones.

5.7.6 Binary Extractor

Relation	From (A)	To (B)	Description
dll_static_ref	DLL	DLL	Entity **A** references entity **B**.
contain		COMObject	Entity **A** contains entity **B**.

Table 5.6: Relation Types Produced by the Binary Extractor

Finally the Binary Extractor examines the binaries for compiled components. The Binary Extractor uses the link table stored inside the binary file to determine the relations shown in Table 5.6. The Binary Extractor supports the Microsoft Portable Executable Format (PE) [Cor97] and can only be used to extract relations from Microsoft Windows binaries. Other tools such as `nm` [tPOSIP93] can be used to extract relations from non-Microsoft Windows binaries. The Binary Extractor is used to analyze:

1. components for which we don't have access to the source code. The component may have been purchased or the source code of the components may not be given for analysis to us because of its sensitive nature.

2. components for which we do not have a Language Extractor. Many languages are used to develop web applications and we lack the resources to develop a

language extractor for each one.

5.8 Summary of Architecture Recovery

We have shown that software developers have different needs and concerns when they study and analyze web applications. Traditional architecture recovery framework such as `PBS` can be modified to meet the demands of the developers. A new domain model and a new fact extraction process has been presented.

We present a modified domain model for web applications. The domain model is a three layer model: the entity-level schema (ELS), the component-level schema (CLS) and the architecture-level schema (ALS). Such layering enables the analysis of web applications at different levels of abstractions. At the lowest layer is the ELS which describes all the permitted relations between the top-level program entities such as function, variables, user objects, database tables and distributed objects. We show the ELS for two of the most widely used languages in the development of web applications: JavaScript and VBScript. The CLS raises the level of abstraction to the component level. It describes all the permitted relations between the different components of a web application such as Active Server Pages (ASP), database tables, libraries and distributed objects. At the highest level of abstraction lies the ALS which describes all the permitted relations between the architecture elements such as subsystems and components.

Once we described all the levels of facts and the relations between them, we explained the framework used to extract facts that comply to the schemas we described in this chapter. We detailed the extraction process used to extract facts from the different components that form a web application. The extraction process employs different types of extractor. Each extractor is invoked on the appropriate component and the results of the execution of all the extractors on all the components of the web application are combined into a single fact base using the presented domain model for web application.

Chapter 6

Case Studies

To evaluate the recovery framework we presented in Chapter 5, we recovered the architecture of several web applications. In this chapter, we give an overview of the web applications we studied. We also examine the architecture of two of the studied web applications and show the benefits of using the recovered architecture diagrams to maintain the application and understand its structure.

6.1 Studied Web Applications

In Chapter 5, we explained the framework used to to recover the architecture of web applications. To evaluate this framework, we recovered the architecture of two commercial and two non-commercial web applications. Table 6.1 provides an overview of the functionality of the studied web applications.

To recover the architecture, we employed the process described by Bowman in [Bow99]. The process was modified to address the points of differences between

Application	**Description**
Hopper News (*noncommercial*)	Hopper News is an application that enables users to post and browse classified advertisements online. Users must pay a service charge to post classifieds but any one can browse the classifieds
Exploration Airline (*noncommercial*)	Exploration Airline is a set of applications used by an airline company. The company has an extranet application to enable its partners to connect. It has an intranet application for its employees to check their benefits. It also has an internet application to provide service for its customers.
Wireless (*commercial*)	Wireless is a set of applications used by a wireless service provider to maintain its subscriber base. In addition, the application enables subscribers to maintain their subscription information and permits users to send messages to subscribers (SMS - Short Message Service). The application is accessible through a wireless phone browser using WML or a traditional browser using HTML. We studied only the HTML portions of the application.
Bug Tracking (*commercial*)	The Bug Tracking application enables a development team members to record reported and discovered bugs. They can also track the progression of the bug until it is resolved.

Table 6.1: Description of the Studied Web Applications

web applications and traditional applications. For each studied application, we performed the following steps:

1. We acquired the available documentation and source for the web application. The non-commercial applications had more documentation than the commercial applications as they were developed as samples to showcase the various capabilities of web applications to software developers.

2. The documentation of the system was examined to derive the main concepts and functionality of the system. For example, reading the documentation for the Hopper News application, we discovered that users must post advertisement for products and they must pay fees to do their posting. We recognized the following concepts: Browse advertisements, Place advertisements, User, Advertisement, Product, and Payment.

3. For traditional applications, Bowman suggests compiling the application and executing a couple of simple test cases. For a web application, we install it on a web server. We examine the pages of the application using a web browser. The browsing is used to verify that the application works. It also gives us a better understanding of the functionality provided by the application.

4. Once the last two steps were completed, we developed the conceptual architecture of the web application. The previously developed concepts and the relations between them were depicted through a simple diagram using boxes to represent the concepts and arrows to represent the interactions between the various concepts.

5. Our extractor crawled the source of the web application stored on the local file system of the web server. The extractor generated facts based on the type of the examined files. Table 6.2 gives a count of the various kind of entities, extracted from the studied applications.

6. Using the developed conceptual architecture and the directory structure, we clustered the entities extracted in the last steps into smaller meaningful subsystems. Regular expressions, the directory structure and file naming conventions assisted us in the clustering process. For example, in one application, when the name of a file started with the string "customer", we placed the file in the customer subsystem. At this step, we might examine closely the source of a component if we could not accurately determine the functionality of a specific component to correctly cluster it.

7. The **Grok** tool was used to deduce relations between the subsystems based on relations that existed between the different entities located inside of each subsystem.

8. Finally, the architecture of the systems were visualized using the landscape tool [Pen92]. Developers can use the recovered architectures for impact analysis and for studying the software system.

6.2 Case Study: Hopper News

Hopper News is a sample web application shipped on the MicroSoft Developer Network (MSDN) library CD provided by Microsoft Corp. The application illustrates

Web Application	ASP FILE	DB TABLE	DLL	COM OBJECT	WEB OBJECT
Hopper News	19	8	22	38	12
Exploration Air	167	16	20	58	57
Wireless	89	16	0	216	25
Bug Tracking	71	9	10	76	3

Table 6.2: Count of Entities Extracted from the Studied Web Applications

technologies provided by the Microsoft web development platform to develop web applications.

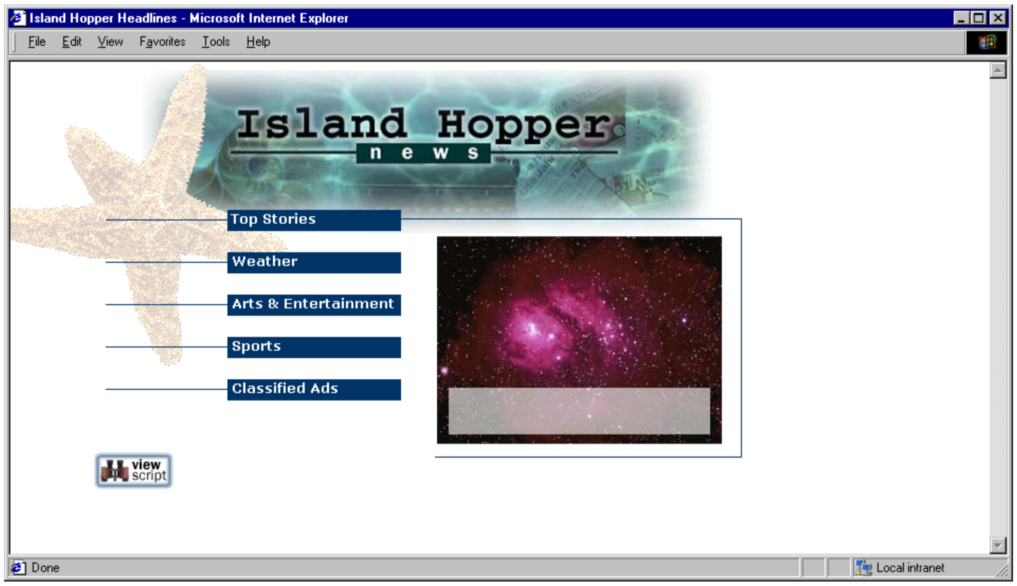
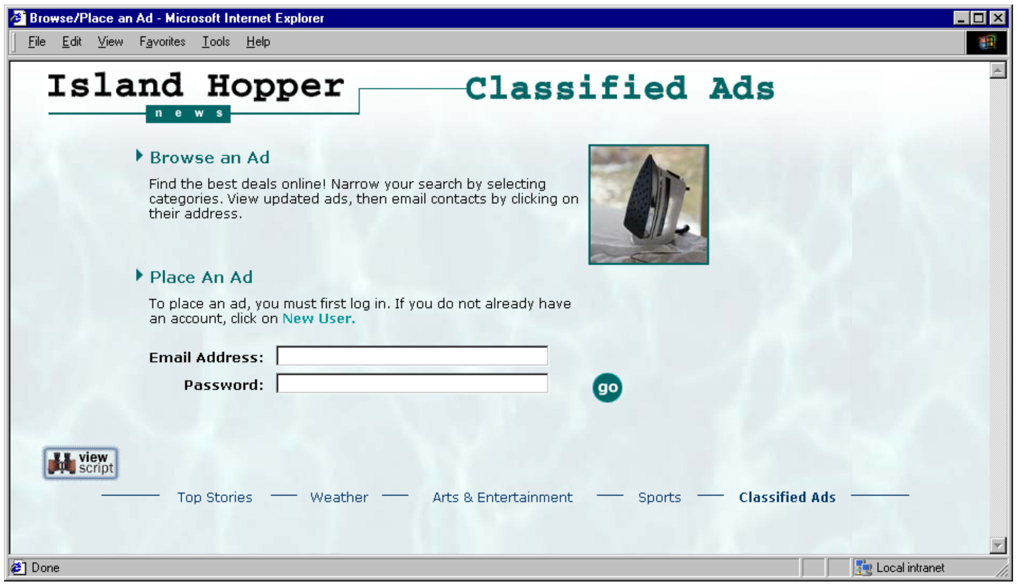


Figure 6.1: Hopper News Main Page

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 6.1, are not functional. The only functional link is the lowest link in the page which links to the classified advertisement page, shown in Figure 6.2.



Find the best deals online! Narrow your search by selecting categories. View updated ads, then email contacts by clicking on their address.

To place an ad, you must first log in. If you do not already have an account, click on New User.

Figure 6.2: Hopper News Classified Advertisement Web Page

6.2.1 Conceptual Architecture for Hopper News

Browsing the application and reading its documentation, we determine that the application provides two main functionalities: it permits any person to browse advertisements, and it permits registered users to place advertisements. To place an advertisement, the user must pay a fee.

Figure 6.3 shows the conceptual architecture of the Hopper News classified advertisement web application. The conceptual architecture is derived from the concepts, and relations we deduced by reading the documentation and using the application. The conceptual architecture is drawn using a drawing tool, whereas all the concrete architecture are screenshots of the PBS landscape viewer.

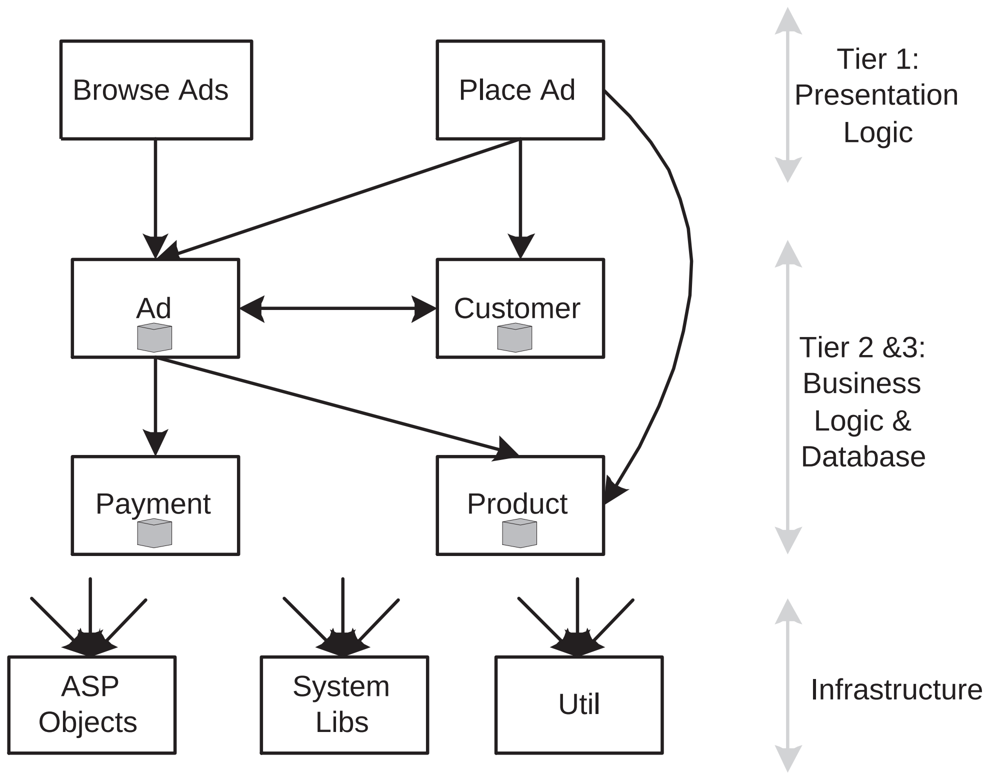


Figure 6.3: Conceptual Architecture for Hopper News

The Hopper News web application has a 3-tiered architecture:

1. The highest tier - *Presentation Logic* - provides the user interface to the two functionalities of the application: browsing and placing advertisements.

2. The *Business Logic* tier encapsulates the business rules based on the concepts in the application domain (such as Customer, Ad, and Product). For exam-

ple, in figure 6.3 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. All business rules are stored in the *Business Logic* tier and not in the *Presentation* tier.

3. The *Database* tier provides persistent storage for the business data. In figure 6.3, the *Business Logic* and the *Database* tiers are clustered together based on the concepts that they support.

An *Infrastructure* layer supports by these three tiers. It provides support for the basic functionality needed by the various tiers, 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.

6.2.2 Concrete Architecture for Hopper News

The conceptual architecture shown in figure 6.3 was derived through reading the available documentation and using the application. On the other hand, the concrete architecture is based on the source code of the application. Using our extractors and the techniques described in chapter 5, we recovered the concrete architecture, as shown in figure 6.4. This figure is a screenshot from the landscape viewer. We note that the `UTIL`, `SYSTEMLIBS`, and `ASPOBJECTS` subsystems are used extensively

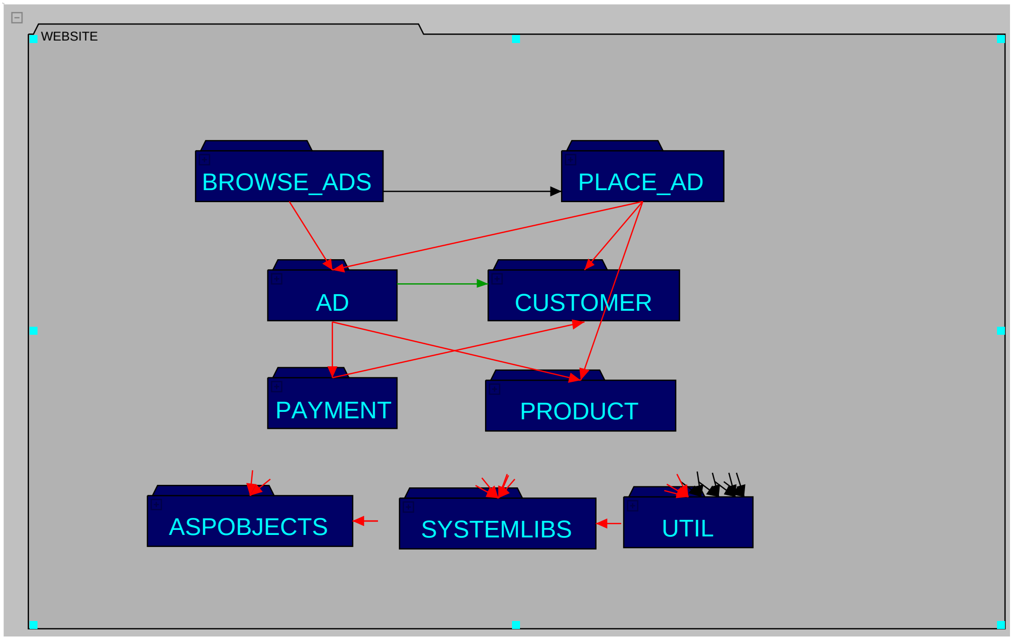


Figure 6.4: Concrete Architecture for Hopper News

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

Figure 6.5 shows the legend used by the landscape viewer. The landscape viewer presents the recovered concrete architecture and enables developers to analyze the architecture by using interactive queries such as "*What are all subsystems that use the Customer subsystem?*". The tool 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, the tool displays the actual type of relation. However, for the shown diagrams in this chapter we can only determine

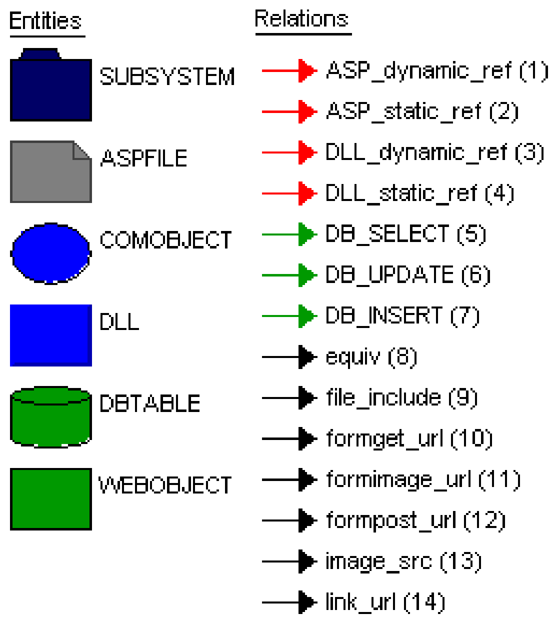


Figure 6.5: Legend for Landscape Viewer Diagrams

the type of the relation based on its color.

The viewer permits us to delve deeper into every subsystem. We simply double-click on the subsystem we need to investigate and the tool opens a new diagram showing the internals of the subsystem. As we go deeper into the recovered architecture diagrams, we get a closer look at the components that form the subsystem. Also we see the internal interaction between the components and the interactions between the components in the current subsystem and the other components that reside in other subsystems in the web application. Figure 6.6 shows the internals of the `CUSTOMER` subsystem. It shows the various databases that are part of the `CUSTOMER` subsystem and visualizes their interaction with the `DLL`s that encode the business rules of the web application.

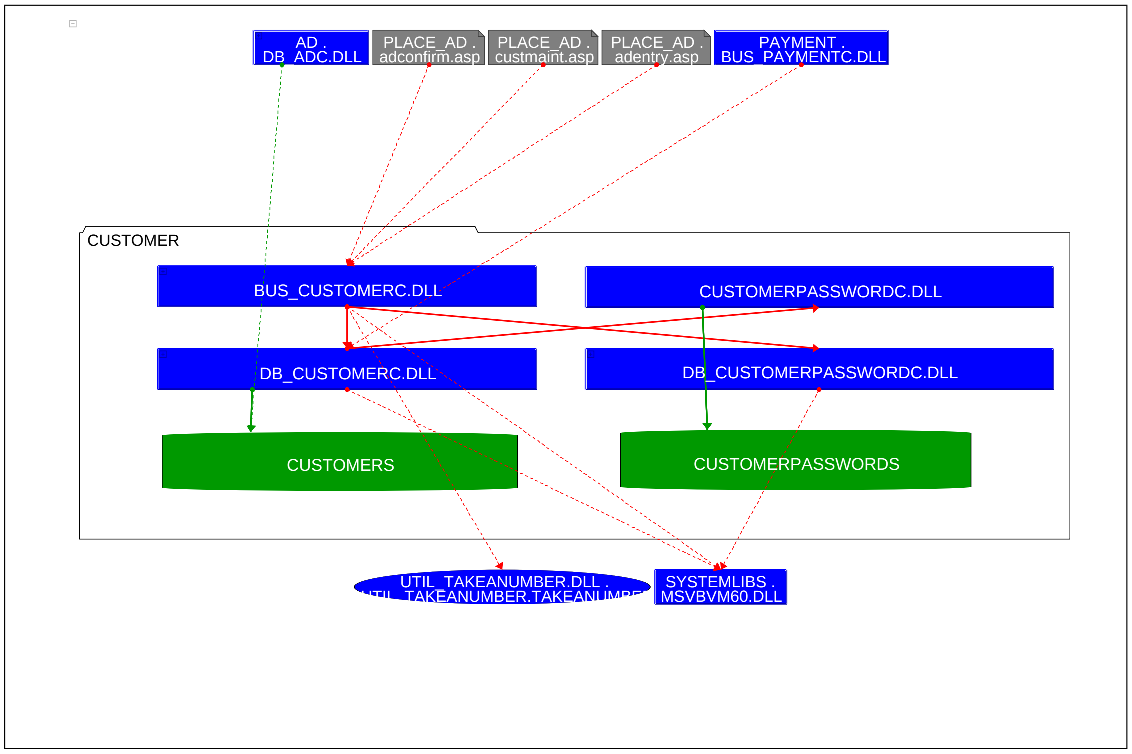


Figure 6.6: Hopper News Customer Subsystem

6.2.3 Comparing the Conceptual and Concrete Architectures

The recovered concrete architecture permits us to verify that the conceptual architecture matches the concrete architecture of the application. In many software systems, the conceptual architecture does not match the concrete architecture for many reasons such as implementation language constraints, or the lack of understanding of the conceptual architecture by new developers. For example, a developer working on a component in the *Presentation* tier may access directly a database in the *Database* tier instead of going through the appropriate components in the *Business*

Logic tier. Unfortunately, there exists no development tools to detect this violation in the architecture of a 3-tiered system. There are no tools to prevent a developer from inserting the offending code into the application's code base.

Comparing figures 6.3 and 6.4, we notice the following:

1. The conceptual architecture contains an extra dependency that does not exist in concrete architecture. In the concrete architecture, we do not have a dependency from the `CUSTOMER` subsystem to the `AD` subsystem. When we developed the conceptual architecture, we had envisioned the need for such a dependency to permit a customer to retrieve a listing of all her/his advertisements. This functionality was implemented differently than our expectations: instead the `AD` subsystem is responsible for implementing this functionality.

2. The concrete architecture contains an extra dependency that was not accounted for in the conceptual architecture. In the conceptual architecture, we do not have a dependency from the `PAYMENT` subsystem to the `CUSTOMER` subsystem. Examining the source code, this extra dependency exists to permit a customer to carry a balance. The `PAYMENT` subsystem updates the customer's balance every time a payment is made.

Using the landscape viewer, we can find and investigate mismatches between the conceptual and concrete architecture. Later, we can ask developer or examine the source code to explain these mismatches.

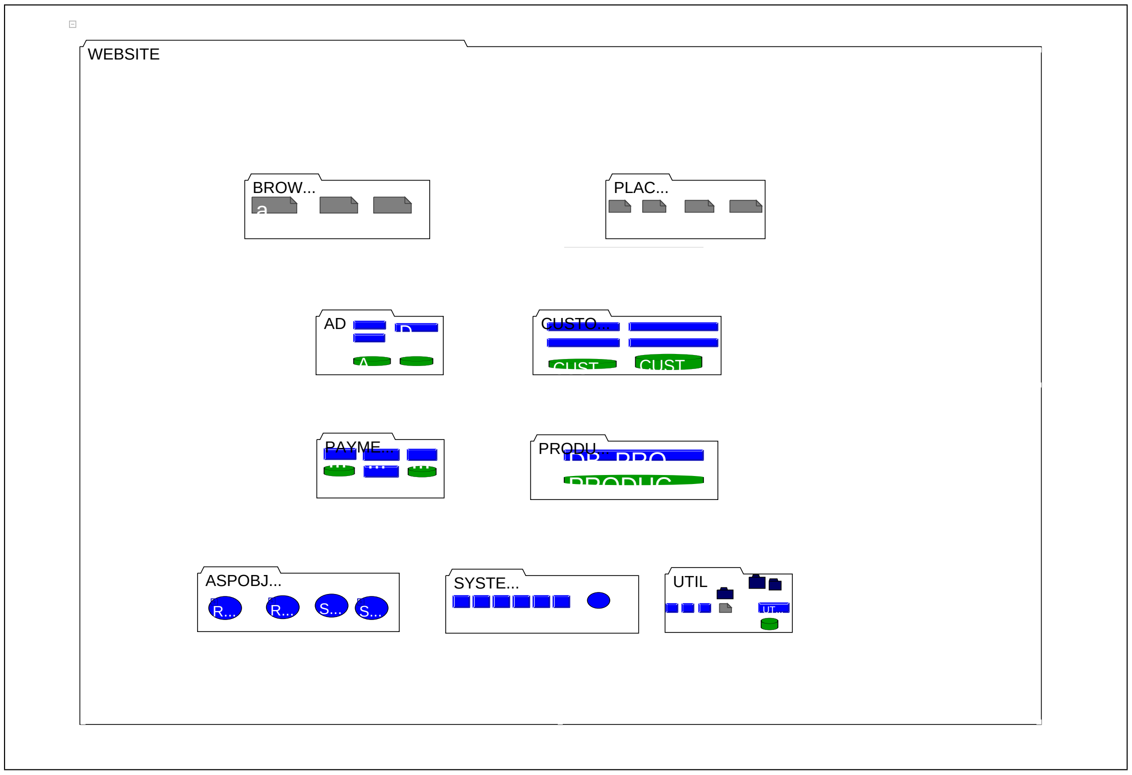


Figure 6.7: 3-Tiered Concrete Architecture for Hopper News

6.2.4 Verifying the Tiered Architecture

Viewing figure 6.7 and concentrating only on the colors in the figure, we can clearly recognize the three different tiers. Using the color scheme described in figure 6.5, we expect to see grey entities at the top for the *Presentation* tier, blue entities for the binaries that encode the business rules in the *Business Logic* tier, and green entities at the bottom for the *Database* tier.

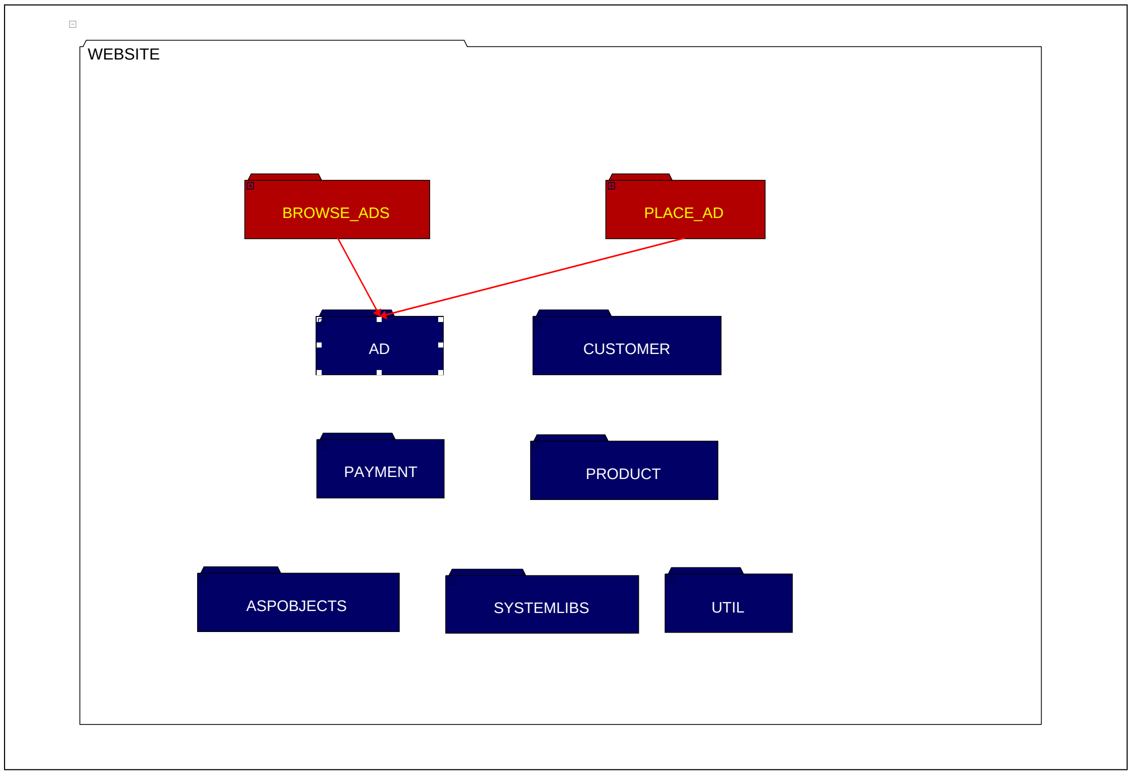


Figure 6.8: Backward-query for the `AD` Subsystem in Hopper News

6.2.5 Impact Analysis

The landscape viewer provides a simple query engine that permits software developers to perform impact analysis studies on the architecture of their system. For example, given a component, such as a database table, a developer can locate all the components that depend directly on this table. The impact analysis feature in the landscape viewer permits the developer to ask two types of questions:

1. *backward-query*: Given a component, what are the other components and subsystems that depend on it?

2. *forward-query*: Given a component, what are the other components and sub-

systems that it depends on?

Figure 6.8 and figure 6.9 show examples of the backward-query applied to the `AD` and `CUSTOMER` subsystem respectively[1]. Such results enable a team lead to determine less critical subsystems and assign them to junior developers. For example, the `CUSTOMER` subsystem is needed for only posting advertisements, but the `AD` subsystem is needed for posting and browsing advertisements. Thus, it may be prudent to assign the `CUSTOMER` subsystem to a junior developer.

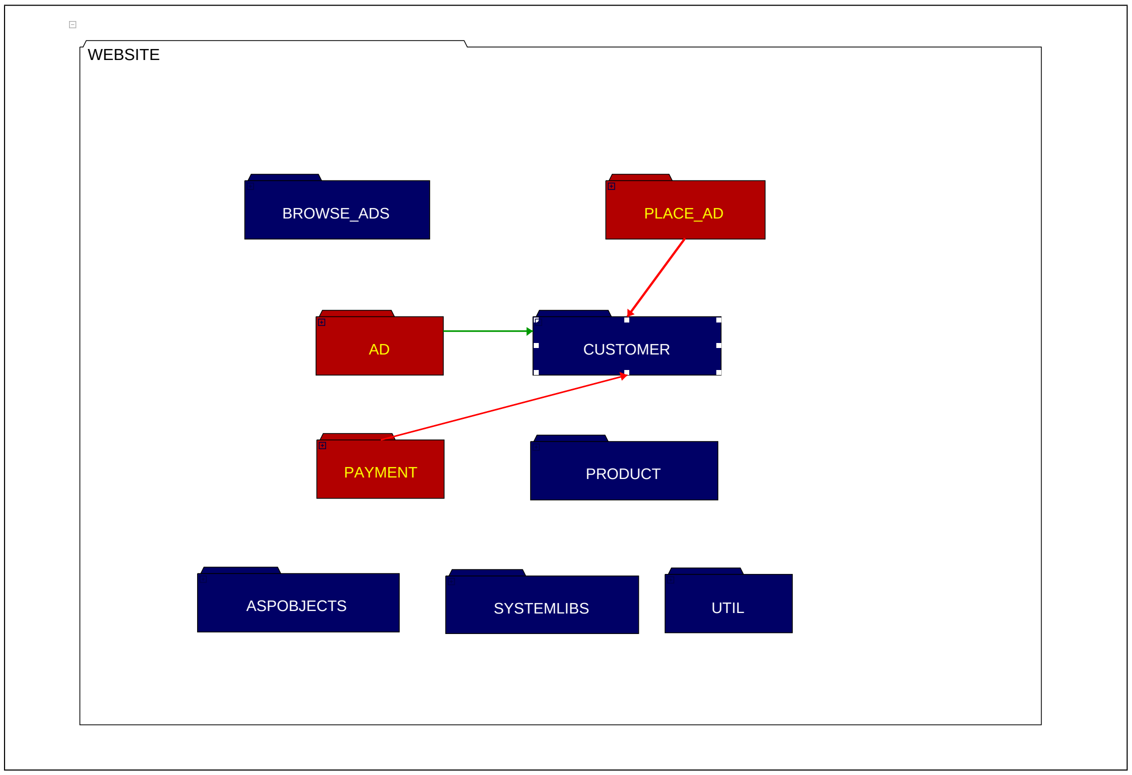


Figure 6.9: Backward-query for the `CUSTOMER` Subsystem in Hopper News

[1]notice in the figures the CUSTOMER and AD subsystems are highlighted to indicate the subsystem the query is on

6.2.6 Migration Cost Estimations

As a web application evolves, sometimes the need arises to migrate it to a different hardware or software platform. In other words, the *Infrastructure* layer of a web application may need to change during its lifetime. For example, a web application developed on the Microsoft Windows platform may be moved later to a Unix platform for many reasons, such as performance or scalability. Likewise the application may be moved to a different web server on the same platform.

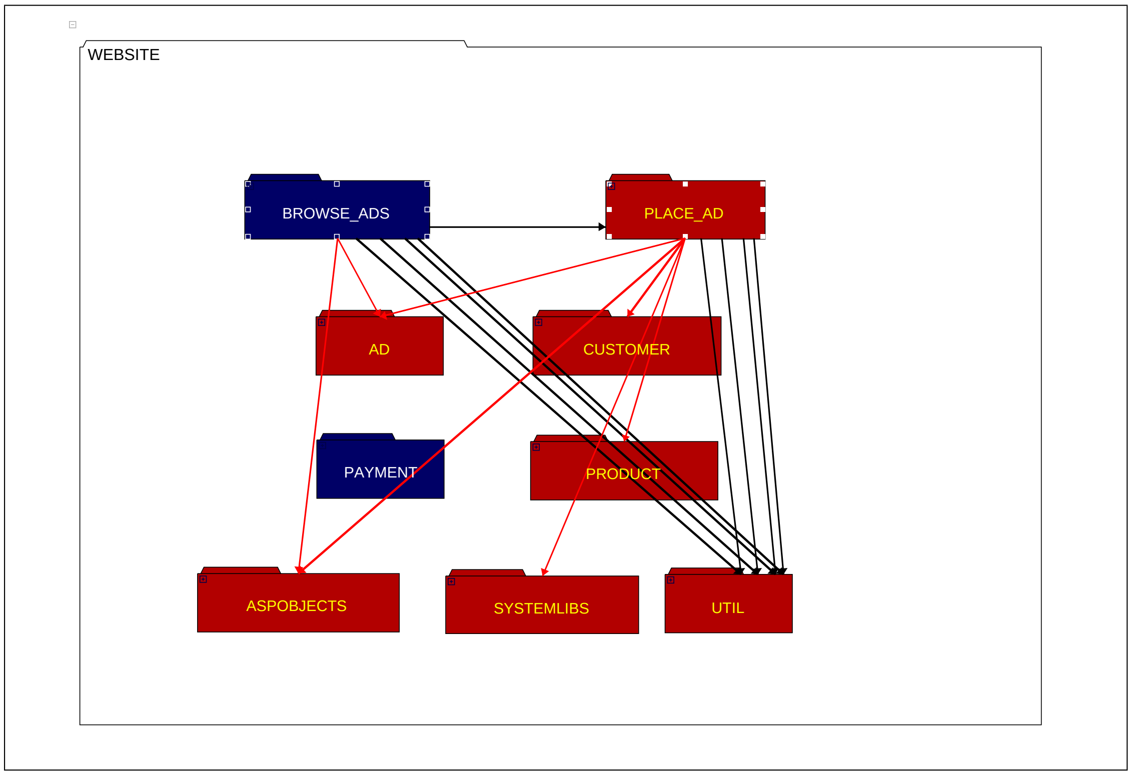


Figure 6.10: Forward-query for the *Presentation* tier in Hopper News

As migration engineers approach such a project, they need a tool to give them an idea of the amount of effort involved in the migration process. The migration engineers could use the displays of the architectures to get a better understanding of the structure of the system before they start working on it. Moreover, they can use the impact analysis feature in the landscape viewer to get an idea of the amount and type of dependencies between the web application and the *Infrastructure* layer.

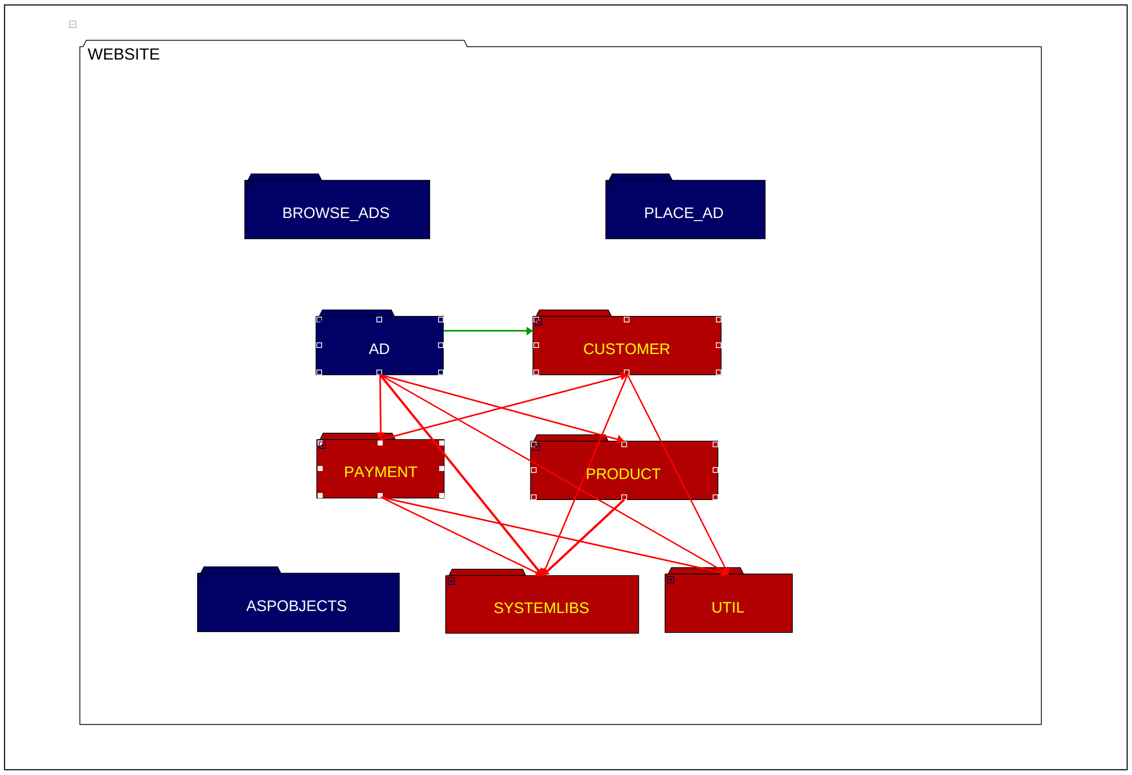


Figure 6.11: Forward-query for the *Business Logic* and *Database* tiers in Hopper News

To estimate the cost of migrating the web application, a migration engineer would examine the dependency of each subsystem or tier on the *Infrastructure* layer. The more depend the application is on the *Infrastructure* layer, the higher the migration cost. Figure 6.10 shows the dependency between the subsystems in the *Presentation* tier (`PLACE AD` and `BROWSE AD` subsystems) and the rest of the application including the *Infrastructure* layer. Figure 6.11 shows the results of a *forward-query* on the *Business Logic* and *Database* tiers in the Hopper News application. Studying both diagrams, we conclude that the application has a high dependency on the Microsoft platform — all subsystems depend heavily on the *Infrastructure* layer. This is expected because Hopper News is a sample application used to demonstrate the different technologies provided by Microsoft to develop web applications.

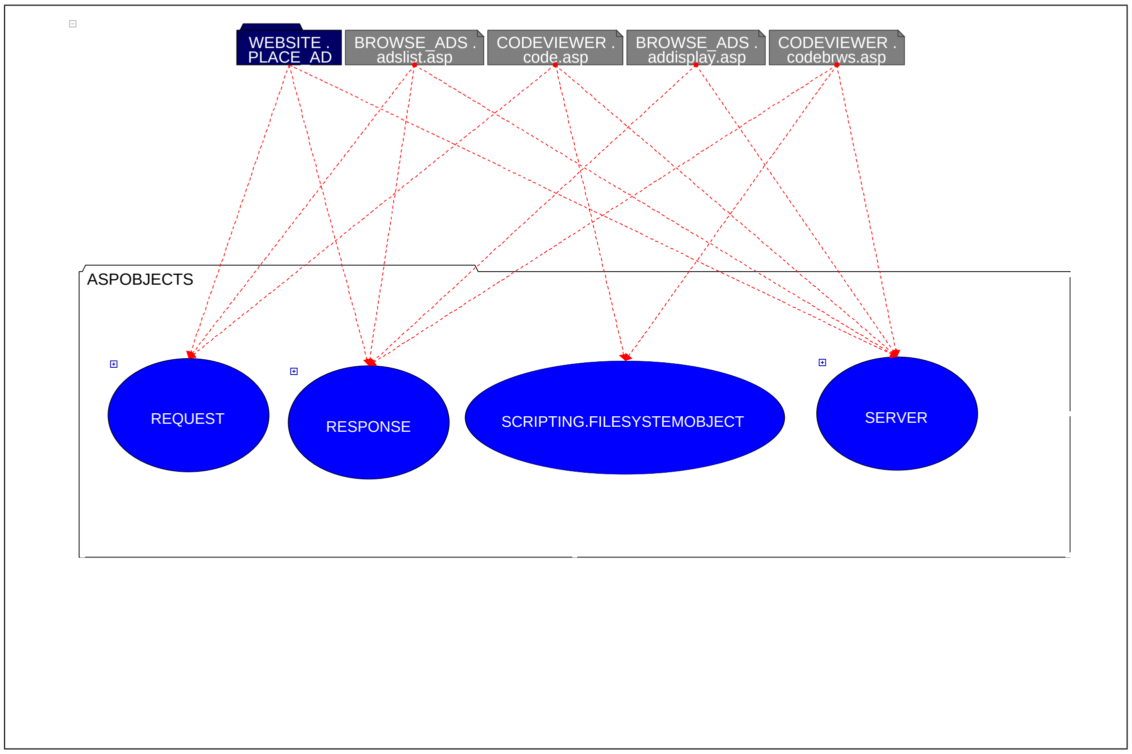


Figure 6.12: Hopper News ASP Objects Subsystem

Migration engineers can have a closer look inside subsystems in *Infrastructure* layer. Thus, they can determine the specific components that are causing the infrastructure dependencies. Examining figure 6.12 we can determine which specific platform features are used by the application based on the infrastructure dependencies.

6.3 Case Study: Wireless

The Wireless web application is one of the commercial applications we studied. The application is used by wireless service providers. It supports several features (sub-applications):

1. Users can send Short Messages (SMS) to a subscriber's wireless device through a web interface. (`SMS_PUSH` subsystem)
2. Administrators can manage their subscribers' base. They can search for subscribers, delete subcribers, or add new ones. Furthermore, they can modify the type of services available to every subscriber. (`ADMIN` subsystem)
3. Subscribers can manage some of their account information (such as their mailing address). (`SUB_UI` subsystem)
4. Subscribers can store their bookmarks online. Also, the bookmarks can be updated and accessed using a web browser or a WAP enabled wireless device. (`BOOKMARKS` subsystem)
5. Users can search for other users' directory entries. (`DIR_LOOKUP` subsystem)

Figure 6.13 shows the concrete architecture of the Wireless web application. The application is developed using Active Server Pages. Scripts inside of the application pages are written in both VBScript and JavaScript. To access and query

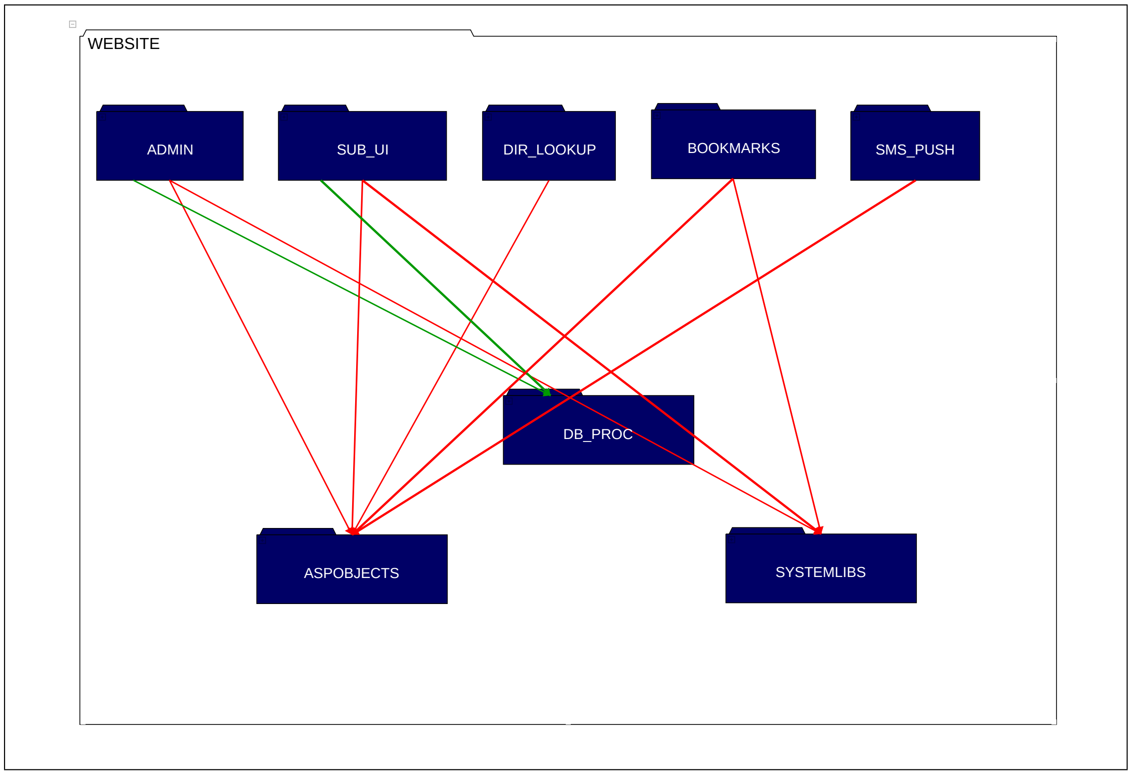


Figure 6.13: The Concrete Architecture of the Wireless Application - Highest Level

persistent storage, the application uses direct access through SQL command and stored procedures.

The main functionalities are clustered into five separate subsystems: `ADMIN`, `SUB_UI`, `DIR_LOOKUP`, `BOOKMARKS`, and `SMS_PUSH`. Data tables and stored procedures used between different sub-applications are placed in a separate subsystem (`DB_PROC`). For example, the subscriber related data tables are accessible from both the `SUB_UI` and `ADMIN` subsystems as both administrators and users need access for the tables. As usual, we have an *Infrastructure* layer that contains the `ASPOBJECTS` and `SYSTEMLIBS` subsystems.

6.3.1 Bookmarks Sub-Applications

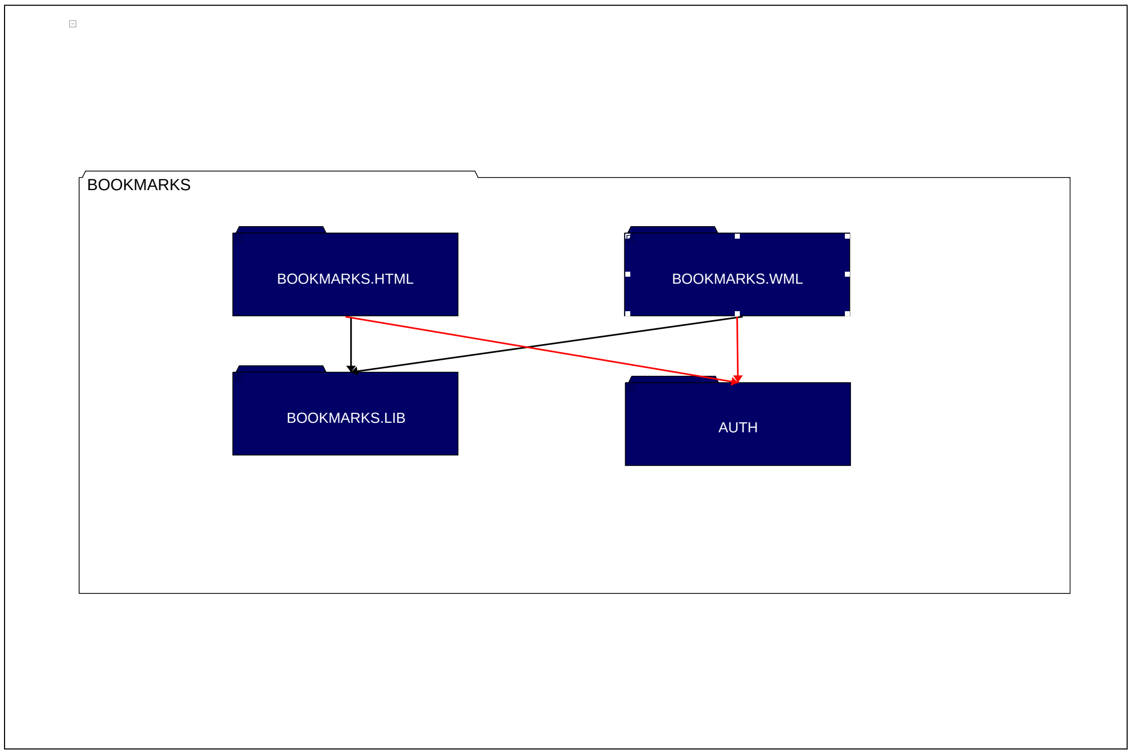


Figure 6.14: The Concrete Architecture of the Bookmarks Sub-Applications

The bookmarks application enables subscribers to store their bookmarks online. Furthermore, bookmarks can be updated and accessed using a web browser or a WAP enabled wireless device. Figure 6.14 shows the concrete architecture of the applications. It is composed of four main subsystems:

1. An HTML subsystem that provides the functionality to access the application through a traditional desktop browser. (`BOOKMARK.HTML` subsystem)

2. A WML[2] subsystem that provides the functionality to access the application

[2]WML is the Wireless Markup Language

through a WAP devices such as a wireless phone equipped with a browser. Because of the small screen size and the limited bandwidth of wireless devices, the HTML subsystem cannot be used. (`BOOKMARK.WML` subsystem)

3. A library subsystem that encapsulates all the code shared between the WML and the HTML subsystems. (`BOOKMARK.LIB` subsystem)

4. Finally, an authentication subsystem contains the logic needed to authenticate the online user. Once authenticated, the user can access her/his bookmarks and modify them. (`AUTH` subsystem)

The Bookmark HTML Subsystem

Figure 6.15 shows the Bookmark HTML subsystem. From the figure, we see that a `BOOKMARK.LOGON` subsystem authenticates the user using the `AUTH` subsystem. Once logged in, the user can add, delete; edit and create bookmarks. An `BOOKMARK.IMAGES` subsystem stores all the shared images between the different subsystems. We notice that the `BOOKMARK.ADD` and the `BOOKMARK.DEL` subsystems do not have any arrows to other subsystems. This is attributed to the fact that files in both subsystems generate links to files in other subsystems dynamically at runtime.

The Bookmark LIB Subsystem

Figure 6.16 shows the bookmark library which contains only one file that provides APIs to access the different data tables and stored procedures. Application pages

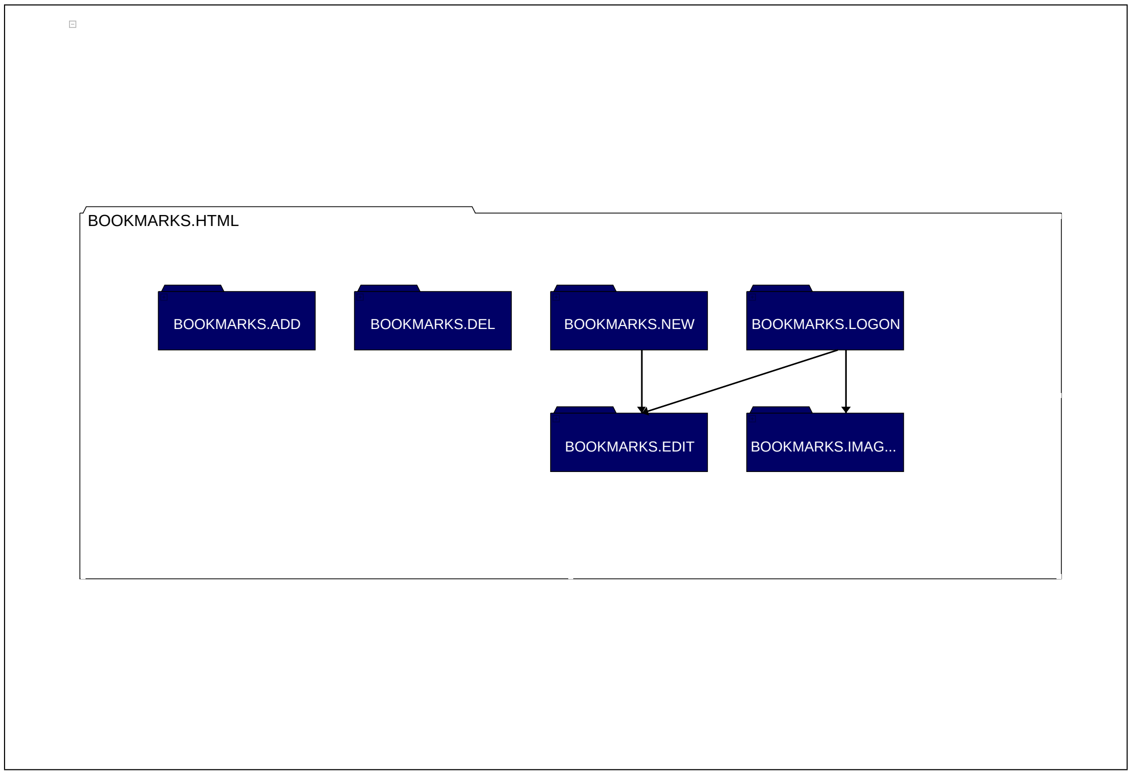


Figure 6.15: The Bookmark HTML Subsystem

have to include the `bm_service.asp` file to gain access to different functions used to access database tables.

The Bookmark AUTH Subsystem

Figure 6.17 shows the bookmark AUTH subsystem which is responsible for authenticating users so they can access their accounts. Although the subsystem provides an interface file (`utility.inc`), the file is not used. Instead, the authentication `DLL` is accessed directly. This could be considered as a violation of the system's design and could be repaired. If the system is being migrated, the migration engineers do not need to migrate the `utility.inc` as it is not used by other components.

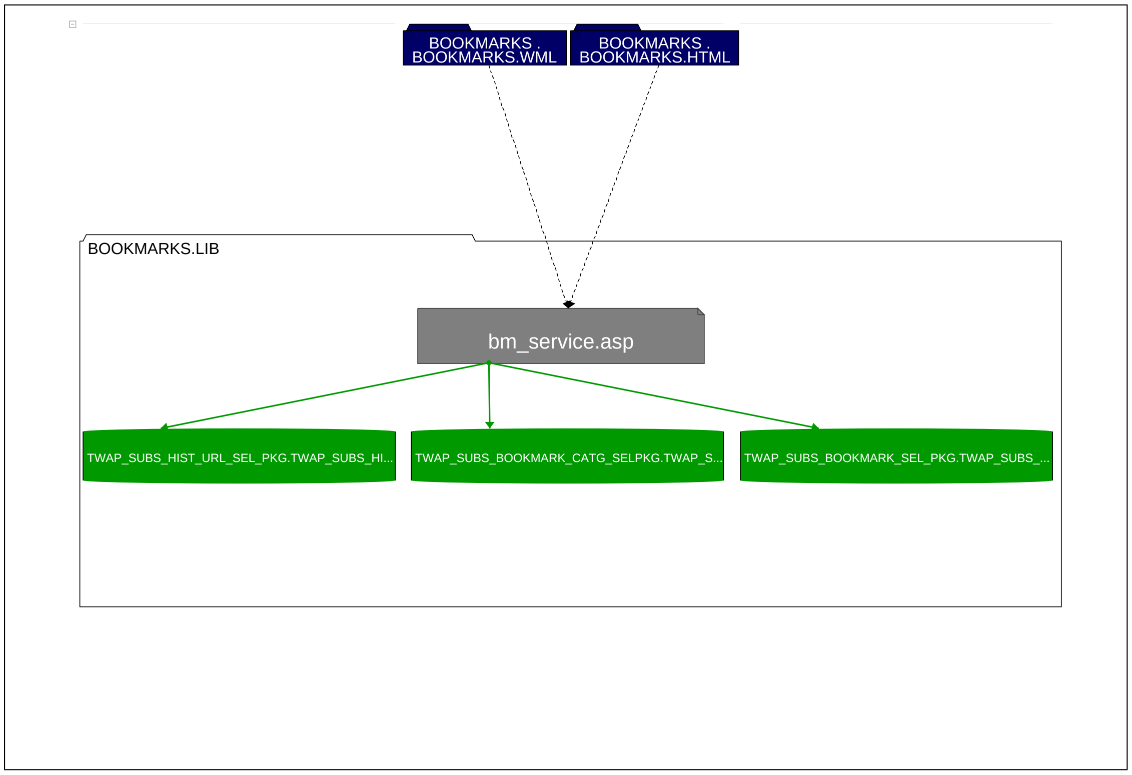

BOOKMARKS . BOOKMARKS.WML
BOOKMARKS . BOOKMARKS.HTML
BOOKMARKS.LIB
bm_service.asp
TWAP_SUBS_HIST_URL_SEL_PKG.TWAP_SUBS_HI...
TWAP_SUBS_BOOKMARK_CATG_SELPKG.TWAP_S...
TWAP_SUBS_BOOKMARK_SEL_PKG.TWAP_SUBS_...

Figure 6.16: The Bookmark LIB Subsystem

6.4 Summary of Case Studies

Our architecture recovery process has been successful in recovering the architecture of four web applications. Our developed extractors were able to gather facts about the studied web applications. Furthermore, the PBS landscape viewer was used to visualize the resulting diagrams.

The recovered architecture for some of the applications was shown to migration engineers who commented that the produced diagrams are very useful in assisting them in understanding the systems. Also, they explained that in many cases they produced similar diagrams by manually examining the code of various components.

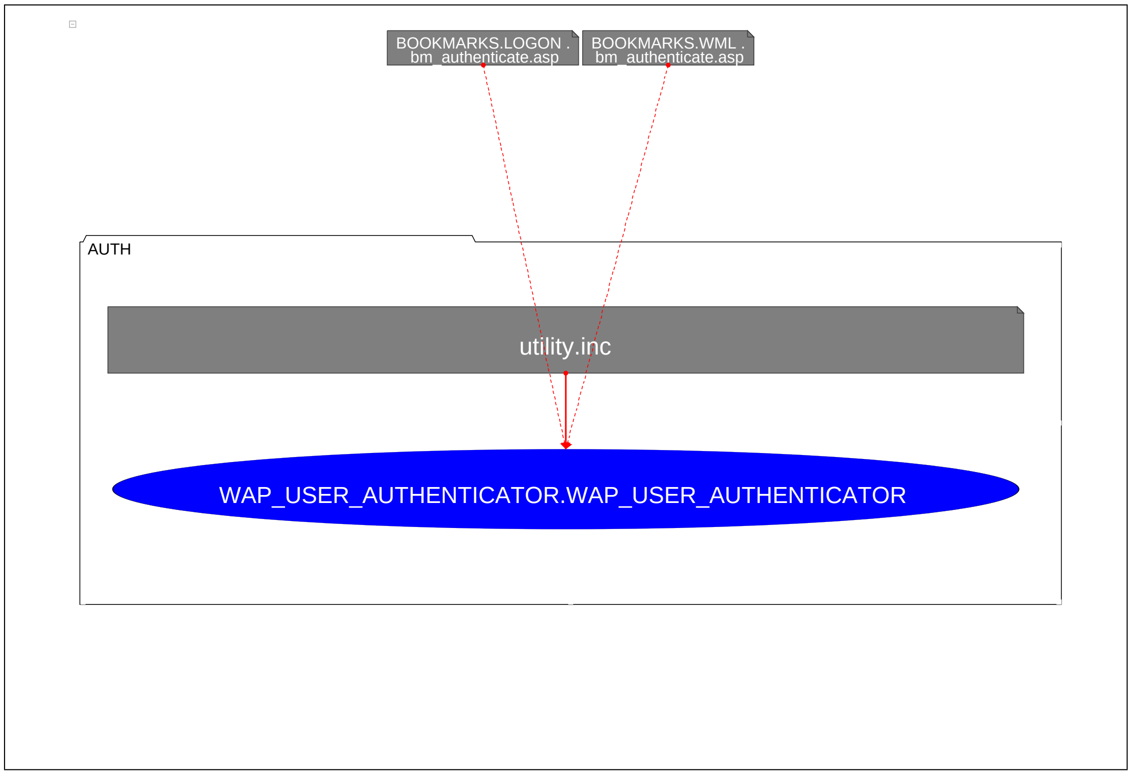

BOOKMARKS.LOGON . bm_authenticate.asp
BOOKMARKS.WML . bm_authenticate.asp
AUTH
utility.inc
WAP_USER_AUTHENTICATOR.WAP_USER_AUTHENTICATOR

Figure 6.17: The Bookmark AUTH Subsystem

To sum up, they considered the recovered architecture diagrams useful for their work and they would have used the tool, if it had been available to them.

Web applications are developed using a variety of languages and technologies. As researchers we have limited resources, so we choose to develop specialized extractors for commonly used languages such as VBScript and JavaScript. To study components developed in other languages, we choose to analyze the binaries. This is not the ideal solution, but we choose to extract as much information as possible given the tools we have and constraints we face.

Nowadays, many large open source systems are available for software engineering researchers to study. To our knowledge, no significant open source web applications exist. The web applications used for our case studies are commercial and non-commercial applications. The non-commercial applications are sample applications provided by Microsoft to showcase the different Windows technologies available to develop web applications. Fortunately, Sun Microsystems of Canada Inc. has provided us with commercial web applications to evaluate our work.

Most of the effort required for the architecture recovery of web application consisted of time spent on clustering the recovered components. For each web applications, the full recovery process took from four to fourteen hours. We believe that any developer with some knowledge of our tools should be able to recover the architecture of their web applications in a reasonable amount of time. Undoubtedly, we are more acquainted with our tools, yet developers working on the system will have an easier time clustering the components as they are more knowledgable of the studied system.

Chapter 7

Conclusions

This chapter summarizes the main ideas presented in this thesis. In addition, future research directions in the architecture analysis and recovery of web applications are addressed and explained.

7.1 Major Topics Addressed

Chapter 1 gives an overview of the history of the web from a document sharing medium to a platform for the development of large scale distributed applications. We explain the reluctance of web developers to follow software engineering practices, as a result of the fast pace and the uncertainty surrounding the development of web application. Web applications are the legacy application of the future. They are critical to the success and future of their organization. Current development tools for web applications are geared towards fast one time releases with no support for multiple releases. The need for tools that aid web developers in maintaining and developing large scale web application is imperative. Our research goal is to provide a tool that permits developers to visualize and analyze the architecture of web applications.

Chapter 2 surveys traditional software engineering research on program understanding, software architecture, architecture views and reverse engineering. We observe that much of the knowledge about a software system rests in the heads of its developers. Software Architecture is used as means for explaining and documenting the designs of large software systems. Because of the complexity of large software system, the use of a single software architecture diagram is discouraged. Instead, we use architecture views to simplify the complexity and improve the understanding of large software architecture documents. Each view addresses a specific area of concern in the design of the system. Unfortunately architecture diagrams are rarely up to date. Reverse engineering semi-automated techniques are used to recover the architecture of large system from the implementation and the developer's knowledge.

Chapter 3 explains the concepts introduced in Chapter 2 for web applications. We provide a concise definition of a web application and a classification of the different types of web applications based on their data and control complexity. We proceed to detail the different architecture views for web application (*Logical*, *Process*, *Physical*, and *Development* views). As a result of the distributed nature of web applications, we introduce a *Security* view to document the security mechanisms employed in large web applications. We also analyze the different available frameworks for developing web applications (such as Active Server Pages, Java Server

Pages, and Cold Fusion). Then, we present a common object model that combines all the common entities across the different frameworks.

Chapter 4 presents related research that attempts to apply traditional software engineering principles and techniques to the development of web applications. The web application domain is young compared to other development domain such as distributed systems. Two areas of active research in the web domain are web engineering and the specification of web applications. We present research in both areas and contrast the presented work to our research in architecture recovery.

Chapter 5 discusses our architecture recovery process for web applications — we present the different tools and techniques used to generate the architecture diagrams. The architecture recovery process of traditional web applications is explained. We highlight the differences between understanding web applications and traditional applications. We present our modifications to the extraction process. We also explain the operations of the tools we developed to recover the architecture of web applications.

Finally, Chapter 6 presents case studies to validate the effectiveness of our extraction and visualization techniques. We demonstrate how to use the architecture diagrams to gain a better understanding of large web applications for maintenance and new development purposes. The architecture of multiple commercial and experimental web applications are used to validate our work.

7.2 Major Thesis Contributions

Web Application developers face different challenges than traditional software developers. With a shorter release cycle, a "*no-documentation*" culture and high employment attrition rates, web development companies face many challenges to stay competitive in a highly volatile market. The need for tools to assist developers of web application is clear and justifiable.

In this thesis, we have shown that current research in software reverse engineering can be adapted and modified painlessly to reverse engineer web applications. We presented a taxonomy of web applications. We introduced the concept of a *Security view* for web applications and explained the different architecture views for web applications. We derived a *Domain Model* for web applications. We developed tools to recover the architecture of web applications based on our presented Domain Model. Furthermore, we have shown how to visualize the recovered architecture using simple box and arrow diagrams.

The recovered architecture is an essential tool to assist maintainers and developers of web applications in their work. The recovered architecture diagrams where shown to developers working on web application. They acknowledged the usefulness of the generated architecture diagrams. They commented that in many cases they derived the same diagrams by inspecting the source code manually. They would have used the tool, if it had been available.

7.3 Future Research

The future of software development is tied very closely to the Web. Web applications are becoming more and more common nowadays. Software engineering research must address the concerns of web developers. As a result of the nature of web development projects, many of the well-studied principles in software engineering are not applicable. Our research has shown that simple modification are needed to adapt many of the tools and techniques developed to study web application. This research is the first step in an emerging research field that addresses the needs of web application software developers. In the following sections, we point out some areas for future research that could extend the work presented in this thesis.

7.3.1 Dynamic Model for Web Applications

In our architecture recovery process, all the visualized information is based on static analysis of all code files for the web application. We do not attempt to profile the application to discover information that are only available at run-time. For example, Figure 7.1 shows the architecture of a bug tracking web application. We notice that the `LOGIN` subsystem does not have any links with the rest of the subsystems in the application. A closer look at the source code and manually tracing through it, we observe that many application pages in the `LOGIN` subsystem generate links to pages in other subsystems based on the associated privileges with the logged-in user. For example, if the user has administrator privileges, a link to the `ADMIN` subsystem would be generated at run-time. Such links are only available

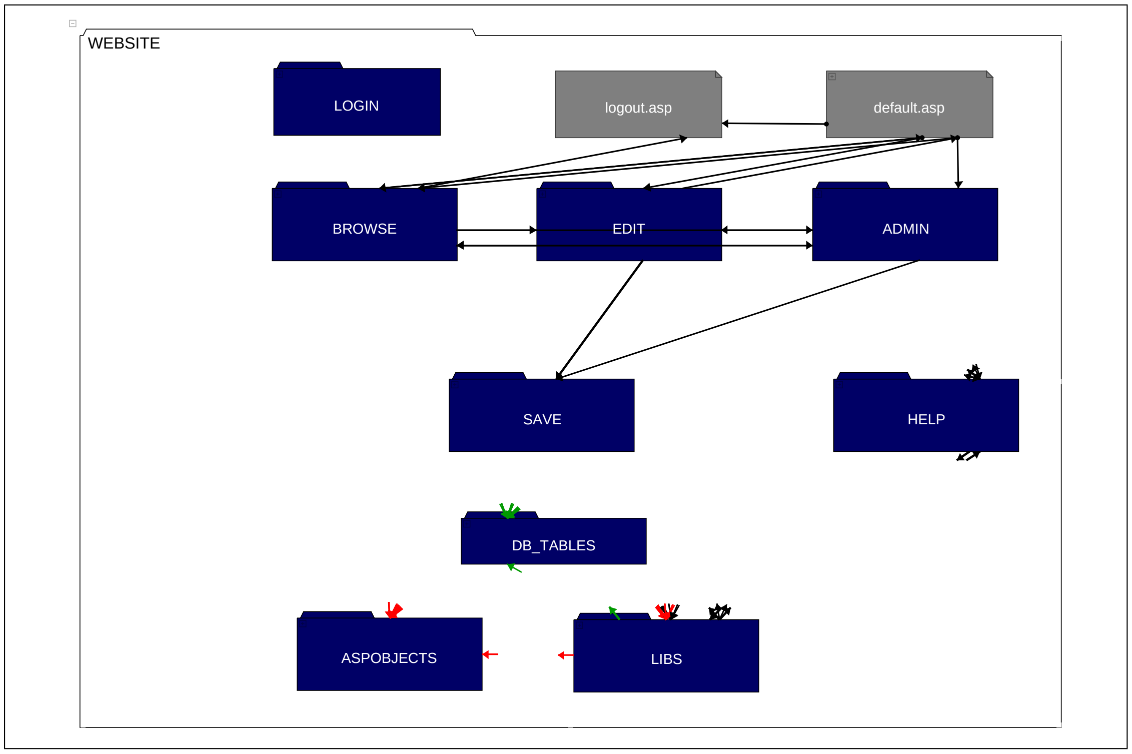


Figure 7.1: PBS Architecture View of a Bug Tracking Web Application

at run-time.

Many techniques are available to automate the recovery of these dynamic relations, examining the source code should be the last resort, for example:

1. The log file of the running web server and the application server hosting the web application can be examined and the run-time relations can be retrieved.

2. A network crawler could exhaustively examine all possible paths in the web applications. The crawler would start at a web page then visit all the other pages it links to, then does the same for the newly visited pages. Such technique would retrieve the largest amount of dynamic information, as the

crawler would parse the final output of the execution of the application pages on the server. Such crawler must contain a large amount of application specific logic to be able to navigate through the different application pages and fill in all the needed fields in each page to advance through the application pages. Although this technique provides the largest amount of dynamic information, overheads for specializing the crawler might be too high. A scripting language may be used to enable the reuse of the crawler logic for as many web applications as possible.

7.3.2 Better Fact Extractors and Architecture Repair

Currently, our extractors analyze the binaries and code files for all components of the web application and emit facts that are later used to generate the architecture diagrams. Unfortunately, we do not store any information in the generated facts to be able to go backwards. (*i.e.* we do not have enough information in the extracted facts to be able to determine given a fact which specific line in a specific file caused the fact to exist)

The fact extractors need to emit a full model of the source code. The full model of the source code will enable architects to perform architecture repair operation at the architecture level with tool support. Architects can simply remove an edge in thel recovered architecture. Then, the tools given the line number information can propagate the changes all the way down to the source code [Tra99].

7.3.3 More Experimentation

We have examined and recovered the architecture of multiple large commercial and experimental web applications. We concentrated on the extraction of web applications developed on the Microsoft Windows platform. Nevertheless, other web applications must be analyzed to gain better insight and validate our results. Furthermore, detailed empirical studies are needed to verify the benefits of our tools for web developers.

7.4 Commercialization

Using the PBS framework developers can recover the architecture of large traditional or web application, with a reasonable time commitment. The benefits of using these generated diagrams are tremendous, ranging from simple re-documentation to impact analysis tasks for future development. Researchers such as [Tra99, Bow99, HH00] have shown many uses of these recovered diagrams. They are mostly used for long term planning. The benefits of such planning are usually unknown in the rapid and volatile software development market.

Companies are reluctant to implement procedure and purchase products that will assist their development teams in the future. They are more concerned with the short run impact on their development cycle. Such mentality explains the quick adoption of RAD (Rapid Application Development) tools and hinders the adoption of architecture analysis tools similar to ours. Until companies begin planning beyond the next release and adopt more mature development cycles which need better planning and forecasting tools, the adoption of our tools in a commercial setting will be minimal. Although it is not clear when our tools will be fully adopted in a commercial setting, we believe that as the complexity of software systems increases, the need for our tools will become eminent. Researchers need to investigate the integration of our recovery tools into traditional application development tools.

Bibliography

[ACCL00] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia. Web Site Reenginnering using RMM. In *Proceedings of euroREF: 7th Reengineering Forum*, Zurich, Switzerland, March 2000.

[ACD95] A.E.Hatzimanikatis, C.T.Tsalidis, and D.Christodoulakis. Measuring the readability and maintainability of Hyperdocuments. *Software Maintenance: Research and Practice*, 7, 1995.

[BBH98] Pearl Brereton, David Budgen, and Geoff Hamilton. Hypertext: The Next Maintenance Mountain. *Computer*, 31(12), December 1998.

[BHB99] Ivan T. Bowman, Richard C. Holt, and Neil V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. In *IEEE 21st International Conference on Software Engineering*, Los Angeles, USA, May 1999.

[Bol00] Cornelia Boldyreff. Web Evolution: Theory and Practice, 2000. Available online at `<http://www.dur.ac.uk/cornelia.boldyreff/lect-1.ppt>`

[Boo00] Grady Booch. The architecture of Web Applications, 2000. Available online at `<http://www.developer.ibm.com/library/articles/ booch_web.html>`

[Bow99] Ivan T. Bowman. Architecture Recovery for Object Oriented Systems. Master's thesis, University of Waterloo, 1999.

[CC90] E. J. Chikofsky and J. H. II. Cross. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7, 1990.

[CD97] Surajit Chaudhuri and Umesh Dayal. An Overview of Data Warehousing and OLAP Technology. *ACM SIGMOD Record*, 26(1), March 1997.

[CFB00] Stefano Ceri, Piero Fraternali, and Aldo 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>`

[COM] The Component Object Model. Available online at `<http://www.microsoft.com/com/tech/com.asp>`

[Con99] Jim Conallen. *Building Web Applications with UML.* object technology. Addison-Wesley Longman, Reading, Massachusetts, USA, first edition, December 1999.

[Cor97] Microsoft Corporation. *Microsoft Portable Executable and Common Object File Format Specification.* Microsoft Press, Seattle, USA, fifth edition, October 1997.

[Cor98] Microsoft Corporation. *Microsoft Visual Basic 6.0 Reference Library.* Microsoft Press, Seattle, USA, first edition, August 1998.

[ECMa] ECMA - Standardizing Information and Communication Systems. Available online at `<http://www.ecma.ch>`

[ECMb] Standard ECMA-262: ECMAScript Language Specification . Available online at `<ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf>`

[Eco99] Computer Economics. e-business implementation by industy sector, 1999. Available online at `<http://ebusiness.mit.edu/cgi-bin/stats/catagorybrowser.cgi>`

[FHK+97] 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), 1997. Available online at `<http://www.almaden.ibm.com/journal/sj/364/finnigan.html>`

[GG99] Hans-W. Gellersen and Martin Gaedke. Object-Oriented Web Application Development. *IEEE Internet Computing*, January 1999.

[GJS96] J. Gosling, B. Joy, and G. Steele. *The Java Language Specification.* Sun Microsystems, 1996.

[GL00] Michael W. Godfrey and Eric H. S. Lee. Secrets from the Monster: Extracting Mozilla's Software Architecture. In *Proceedings of the Second International Symposium on Constructing Software Engineering Tools*, Limerick, Ireland, June 2000.

[Gla92] R. L. Glass. We have lost our way. *Systems and Software*, 18(3), March 1992.

[Gro99] The Object Management Group. *Unified Modeling Language Specification*. The Object Management Group, June 1999. Available online at `<http://www.rational.com/media/uml/post.pdf>`

[GS93] David Garlan and Mary Shaw. An Introduction to Software Architecture. In V. Ambriola and G. Tortora, editors, *Advances in Software Engineering and Knowledge Engineering*, Singapore, 1993. World Scientific Publishing Company.

[Hau00] 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.

[HH00] Ahmed E. Hassan and Richard C. Holt. A Reference Architecture for Web Servers. In *7th Working Conference on Reverse Engineering*, Brisbane, Queensland, Australia, November 2000.

[Hil96] Scot Hillier. *Inside Microsoft Visual Basic Scripting Edition.* Microsoft Press, Seattle, USA, October 1996.

[Hol96] Richard C. Holt. Binary Relational Algebra Applied to Software Architecture. CSRI Tech Report 345, University of Toronto, March 1996.

[Hol97] Richard C. Holt. *An Introduction to TA: the Tuple-Attribute Language*, March 1997. Available online at `<http://plg.uwaterloo.ca/~holt/papers/ta.html>`

[Hol98] Richard C. Holt. Structural manipulations of software architecture using Tarski relational algebra. In *Proceedings of WCRE'98*, October 1998.

[ITBH99] Michael W. Godfrey Ivan T. Bowman and Richard C. Holt. Extracting Source Models from Java Programs: Parse, Disassemble, or Profile?, 1999. Available online at `<http://plg.uwaterloo.ca/~itbowman/papers/javasrcmodel.html>`

[KLDM98] Gregory Knapen, Bruno Laguë, Michel Dagenais, and Ettore Merlo. Parsing C++ Despite Missing Declarations. In *Proceedings of the Seventh International Workshop on Program Comprehension.* IEEE, October 1998.

[KM98] Peter Kent and Kent Multer. *Official Netscape Javascript 1.2 Programmer's Reference : Windows, MacIntosh and Unix.* Ventana, first edition, January 1998.

[Kon00] Rachel Konrad. Tech employees jumping jobs faster, 2000. Available online at `<http://news.cnet.com/news/0-1007-202-2077961.html>`

[KR98] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice-Hall, Inc., Englewood Cliffs, NJ., USA, second edition, June 1998.

[Kru95] Philippe B. Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6), November 1995.

[LA97] Timothy C. Lethbridge and Nicolas 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.

[Lee00a] Eric H. S. Lee. Analyzing Mozilla, 2000. Available online at `<http://plg.uwaterloo.ca/~ehslee/pub/mozilla.ppt>`

[Lee00b] Eric H. S. Lee. Software Comprehension Across Levels of Abstraction. Master's thesis, University of Waterloo, 2000.

[LHGF98] L. D. Landis, P. M. Hyland, A. L. Gilbert, and A. J. Fine. Documentation in a software maintenance environment. In *Proceedings of the Conference on Software Maintenance*, October 1998.

[MMAC99] G. Mecca, P. Merialdo, P. Atzeni, and V. Crescenzi. The Araneus Guide to Web-Site Development - Araneus Project Working Report.

AWR-1-99, University of Roma Tre, March 1999. Available online at `<http://www.dia.uniroma3.it/Araneus/publications/ AWR-1-99.ps>`

[MNS95] Gail C. Murphy, David Notkin, and Kevin Sullivan. Software Reflexion Models: Bridging the Gap Between Source and High-Level Models. In *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, New York, NY, October 1995. ACM.

[MS] Microsoft Corporation. Available online at `<http://www.microsoft.com>`

[NS] Netscape Corporation. Available online at `<http://www.netscape.com>`

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

[Pen92] David A. Penny. *The Software Landscape: A Visual Formalism for Programming-in-the-Large.* PhD thesis, University of Toronto, 1992.

[Pre00] Roger S. Pressman. What a Tangled Web We Weave. *IEEE Software*, 17(1), January 2000.

[PW92] Dewayne E. Perry and Alexander L. Wolf. Foundations for the Study of Software Architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4), October 1992.

[RBP$^+$91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick

Eddy, and William Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Inc., Englewood Cliffs, NJ., USA, 1991.

[RT00] Filippo Ricca and Paolo Tonella. Visualization of Web Site History. In *Proceedings of euroREF: 7th Reengineering Forum*, Zurich, Switzerland, March 2000.

[SCH98] Susan E. Sim, Charles L. A. Clarke, and Richard C. Holt. Archetypal Source Code Searching: A Survey of Software Developers and Maintainers. In *Proceedings of International Workshop on Program Comprehension*, Ischia, Italy, June 1998.

[SCHC99] Susan E. Sim, Charles L. A. Clarke, Richard C. Holt, and Anthony M. Cox. Browsing and Searching Software Architectures. In *Proceedings of International Conference on Software Maintenance*, Oxford, England, 1999.

[Sim98] Susan 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>`

[SNH95] Dilip Soni, Robert L. Nord, and Christine Hofmeister. Software Architecture in Industrial Applications. In *IEEE 17th International Conference on Software Engineering*, 1995.

[Sta84] Thomas A. Standish. An essay on Software Reuse. *IEEE Transactions on Software Engeineering*, 10(5), 1984.

[Str97] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Longman, Reading, Massachusetts, USA, third edition, July 1997.

[TGLH00] John B. Tran, Michael W. Godfrey, Eric H. S. Lee, and Richard C. Holt. Architectural Repair of Open Source Software. In *Proceedings of International Workshop on Program Comprehension*, Limerick, Ireland, June 2000.

[TH96] Vassilios Tzerpos and Richard C. Holt. A Hybrid Process for Recovering Software Architecture. In *Proceedings of CASCON '96*, Toronto, Canada, November 1996.

[TH98] Vassilios Tzerpos and Richard C. Holt. Software botryology: Automatic clustering of software systems. In *Proceedings of the International Workshop on Large-Scale Software Composition*, 1998.

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

[tPOSIP93] Information technology Portable Operating System Interface (POSIX). *Portable Operating System Interface (POSIX) Part 2: Shell and Utilities (Volume 1)*. IEEE Computer Society, 345 E. 47th St, New York, NY 10017, USA, 1993.

[Tra99] John B. Tran. Software Architecture Repair as a Form of Preventive Maintenance. Master's thesis, University of Waterloo, 1999. Available online at `<http://plg.uwaterloo.ca/~j3tran/papers/thesis.html>`

[WCO00] Larry Wall, Tom Christiansen, and Jon Orwant. *Programming Perl.* O'Reilly and Associates, Inc., third edition, July 2000.

[Web99a] International Workshop for Web Engineering - The Eighth International World Wide Web Conference, 1999. Available online at `<http://budhi.uow.edu.au/web-engineering99/web_engineering.html>`

[Web99b] 1st International Workshop on Web Site Evolution, 1999. Available online at `<http://www.cs.ucr.edu/~stilley/wse99>`

[Web00] 2nd International Workshop on Web Site Evolution, 2000. Available online at `<http://www.cs.ucr.edu/~stilley/wse2000>`

[Zve83] Nicholas Zvegintzov. Nanotrends. *Datamation*, August 1983.