Extending a Web Authoring Tool for Web Site Reverse Engineering

by

Grace Qing Gui
B. Eng., Wuhan University, 1995

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Computer Science

We accept this thesis as conforming to the required standard

Supervisor: Dr. Hausi A. Müller

Abstract

Web site Reverse Engineering involves applying reverse engineering approaches to Web sites to facilitate Web site comprehension, maintenance, and evolution. Traditionally, reverse engineering functionality is implemented with stand-alone tools. During reverse engineering activities, software engineers typically have to switch between forward engineering tools and reverse engineering tools. Each of these tools has its own idiosyncratic user interface and interaction paradigm and therefore has a high learning curve. As a result, many reverse engineering tools fail to be adopted.

This thesis uses the ACRE (Adoption Centric Reverse Engineering) tool development approach to extend a forward engineering tool by seamlessly adding reverse engineering functionality to help software engineers and facilitate the adoption of reverse engineering functionality.

Following this approach, we present a tool prototype called REGoLive, which leverages the Web authoring tool Adobe GoLive by grafting Web site reverse engineering functionality on top of it. In particular, we show how to generate different perspectives of a Web site and establish mappings between them to expose the complex interrelationships of a Web site. We believe that allowing Web developers to generate different interactive, consistent, and integrated views with a Web authoring tool and establishing mappings between the views, facilitates Web site comprehension. The benefits and drawbacks of this approach from the tool user's as well as the tool builder's perspective are discussed.

Examiners:

Contents

List of Figures

List of Tables

Acknowledgments

Special thanks to my supervisor, Dr. Hausi Müller, for his patience, support, guidance and encouragement throughout this research. I really appreciate and cherish for giving me the opportunity to pursue graduate studies under his supervision.

I am also grateful to all the members of the Rigi research group for their contributions to this research project. In particular, I would like to acknowledge the help I received from Holger Kienle, Qin Zhu and Jun Ma, who provided me with valuable advice on the thesis, and Tony Lin, who helped me customize the SVG Editor.

Finally, I would like to thank all my friends in Victoria, and my family, for helping me through this long process with their care and love.

Dedication

To my parents

Chapter 1 Introduction

The Internet has been growing tremendously in recent years [23]. Web sites are becoming the major media through which industry and academia promote their products or ideas and share resources around the world. Various Web technologies including CGI, JSP, PHP, Servlet, CORBA, EJB, SOAP have been well invented to address the need of building more efficient, useful and sophisticated Web applications.

1.1 Motivation

The problem addressed in this thesis occurs when maintaining a complex Web site. Traditional software reverse engineering activities involve identifying components and their dependencies as well as extracting high level system abstractions. With the aid of reverse engineering tools [30, 31, 32], source code of different programming languages can be analyzed with corresponding parsers and artifacts can be extracted and manipulated automatically. Unlike traditional software systems, Web sites are complex heterogeneous systems that consist of various technologies and programming languages [29]. Also changes of Web sites occur more frequently and more radically than for traditional systems [36]. Moreover, although related development methodologies have been proposed in the literature for building a Web application, good software engineering principles usually are not applied in practice due to pressing market demand [33]. Most Web developers pay little attention to development methodologies and process, performance, maintainability and scalability. The development heavily relies on the knowledge and experience of individual developers and their individual development practices rather than standard practices [35]. As a result, these factors, the use of a

multitude of technologies, frequent changes, and lack of design principles, greatly exacerbate the difficulties of Web site maintenance. Thus, approaches that help understanding of Web sites are needed to ease development and maintenance activities.

Quite a few Web site reverse engineering tools have been developed to tackle the Web site maintenance problems [1, 2, 3]. Yet, generally, reverse engineering tools are difficult to learn (e.g. , Rigi [53] has a user manual with hundreds of pages). Users of Web authoring tools, most of them are neither software engineers nor computer scientists, are less likely to evaluate these tools that are hard to install, difficult to learn, and incompatible with their established work practices.

To provide Web site comprehension functionality that is accessible to these kinds of users, we follow a methodology called Adoption-Centric Software Engineering (ACSE) [34], which explores tool-building approaches to make software engineering tools more adoption-friendly by leveraging COTS (Commercial-Off-the-shelf Software) components and/or middleware technologies that are popular with targeted users. Integrating reverse engineering functionalities into such components promises to increase adoptability compared to offering these functionalities in stand-alone, idiosyncratic research tools.

Following this approach, this thesis discusses our experiences with a case study that utilizes a commercial Web authoring tool, Adobe GoLive, as a host component to provide reverse engineering capabilities for Web site comprehension. We hope that this new tool is compatible with existing users and thus will be adopted more readily.

1.2 Approaches

We first need to gain a fundamental understanding of Web Site Reverse Engineering, including process, tasks and requirements.

Next, we select the host component. Ideally the host component is a Web-authoring tool with a large user base, functioning as a major part of the user's workflow, with strong extensibility and visualization capabilities. It is also necessary to analyze the native reverse engineering capabilities the host tool provides, to determine what reverse engineering features can be added considering reverse engineering scenarios.

The added extension should be capable of collecting artifacts from the development environment, from the Web server, and from the client. The collected data can then be analysed and abstracted to give a visual presentation or other formats of output for further exploration.

The added new features should be integrated tightly with the native features. Data, control and presentation integration are required. Data integration can be achieved by introducing XML (eXtensible Markup Language), GXL (Graph eXchange Language) or SVG (Scalable Vector Graphs) [27] as the exchange file format, which are easily exported and shared between tools; control integration should be enabled for the host tool interacting with added functionalities such as calling the APIs in scripting languages and displaying the results; presentation integration is necessary so that the added tool is accessible from a consistent user-interface with a common look-and-feel as the host tool.

The new tool needs to be evaluated using reverse engineering tool requirements and compared with stand-alone reverse engineering tools.

1.3 Thesis Outline

This thesis is organized as follows. Chapter 2 provides the background on WSRE (Web Site Reverse Engineering), selected related research tools, and the ACRE approach. Chapter 3 discusses Reverse engineering tool requirements, GoLive reverse engineering

capabilities and compares GoLive with other Web authoring tools. Chapter 4 describes the design and implementation of our prototype, REGoLive. Chapter 5 evaluates the tool we developed. Chapter 6 summarizes the contributions of this thesis and proposes the future work.

Chapter 2 Background and Related Research

This chapter introduces and explains important terms and concepts underlying the Web site reverse engineering field. It also introduces some related work that inspired our research.

2.1 Terminology

2.1.1 Web Application

A web application is a software system where most of its functionality is delivered through the web [38]. A web site may contain multiple web applications. In the early days, Web sites were primarily static, i.e., composed of only static Web pages stored in some file system, linked together through hyperlinks. Today, the rise of new technologies (e.g., server and client scripting languages) has introduced the concept of computation in the Web application realm, thereby allowing novel and much more complex human-Web interactions. Nowadays, Web sites are complex and heterogeneous systems; most of them are dynamic Web sites containing mixes of programs that dynamically generate hyper-documents (dynamic Web pages) in response to some input from the user, and static hyper-documents.

Figure 2.1 depicts a typical generic Web application infrastructure. Web applications are based on the client/server model or 3-tier architectures. Many of them 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. A client sends a request of a Web page over a network to a Web server, which returns the requested page as response. Web pages can be static or dynamic. While the

content of a static page is fixed and stored in a repository, the content of a dynamic page is computed at run-time by the application server and may depend on the information provided by the user. The server programs that generate dynamic pages, run on the application server and can use information stored in databases and call back-end services.

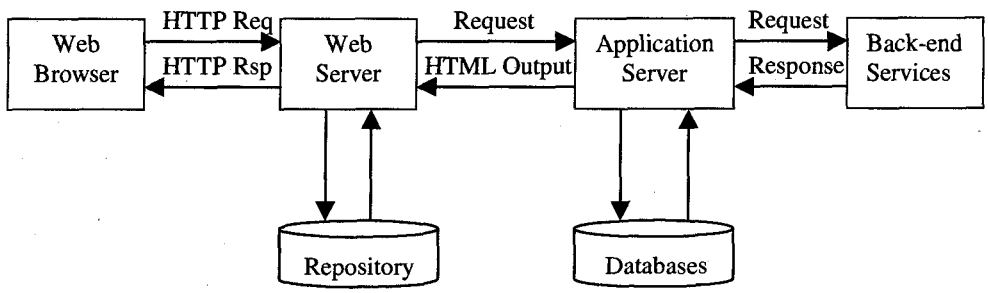

Figure 2.1 Web application infrastructure

The HTML code can activate the execution of a server program (e.g., JSP, ASP, PHP etc) by means of a SUBMIT input within an HTML element of type FORM or anchor and data propagated to a server program by means of form parameters (hidden parameters are constant values that are just transmitted to the server, while non hidden input parameters are gathered from the user). Data flows from a server program to the HTML code are achieved by embedding values of variables inside the HTML code, as the values of the attributes of some HTML elements. Server programs can exploit persistent storage devices (such as databases) to record values and to retrieve data necessary for the construction of the HTML page.

2.1.2 Web Engineering

Web engineering is the establishment and use of sound scientific, engineering and management principles and disciplined and systematic approaches to the successful development, deployment and maintenance of high quality Web-based systems and applications [39]. Web engineering adopts many software engineering principles as well as incorporates new approaches and guidelines to meet the unique requirements of Web-based systems. Building a complex Web application calls for knowledge and expertise from many different disciplines such as: software engineering, hypermedia and hypertext engineering, human-computer interaction, information engineering and user interface development [35].

Web engineering is a special sub area in software engineering. It is document-oriented containing static or dynamic web pages, focused on presentation and interface, content-driven, having diverse users, short development time, and developers with vastly varied skills. The distinguishing characteristics of the Web-based applications include: a relatively standard interface across applications and platforms, applications which disseminate information, the underlying principles of graphic design, issues of security, legal, social, and ethical ramifications, attention to site, document and link management, influences of hypertext and hypermedia, network and Web performance, and evolving standards, protocols, and tools [40].

Web engineering activities covers the whole Web life cycle from conception of an application to development and deployment, continual refinement and upgrade systems. The Web is dynamic and open, likewise, Web engineering needs to evolve and adapt to changes.

2.1.3 Reverse Engineering

Lehman's law of continuing change, which states that software systems that operate in the real world must be continually adapted else they become progressively less satisfactory, has been derived from observation of a variety of traditional software systems [41]. Usually, a system's maintainers are not its designers, so they must expend many resources to examine and learn about the system. Reverse engineering tools can facilitate this practice.

Chikofsky and Cross defined reverse engineering as "analyzing a subject system to identify its current components and their dependencies, and to extract and create system abstraction and design information." [37]. In forward engineering, the subject system is the result of the development process, whereas in reverse engineering, the subject system is generally the starting point of the practice. To identify components and their dependencies, we retrieve the low level artifacts such as call graphs, global variables, and data structures; we then further extract higher level information from the artifacts, to gain system abstraction and design information such as patterns, subsystems, architectures, and business rules. Reverse engineering processes have been proved to be useful in supporting the maintenance of traditional software systems.

Redocumentation and design recovery are two main subareas of reverse engineering. Redocumentation refers to the creation or revision of semantically equivalent representation within the same relative abstraction level. Design recovery goes beyond the information obtained directly by examining the system itself, by adding domain knowledge, external information and deduction reasoning to recreate design

abstractions. Design recovery thus deals with a far wider range of information than found in the conventional software engineering representations or code.

The extracted information should be understandable to and manageable by software engineers in order to facilitate the software maintenance, hence the information should be properly stored, manipulated, and in particular, visualized to facilitate human understanding. Visualization can be described as a mapping of data to visual form that supports human interaction for making visual sense [42]. The flow of data goes through a series of transformations to visual views. Software engineers may adjust these transformations, via user controls, to address the particular reverse engineering task.

2.1.4 Web Site Reverse Engineering

Reverse engineering processes have proved to be useful in supporting the maintenance of traditional software systems. Similarly, WSRE proposes to apply reverse engineering approaches to Web sites in order to reduce the effort required to comprehend existing Web sites and to support their maintenance and evolution. Thus, traditional reverse engineering approaches such as program analyses are being applied to Web sites.

Tonella classified server programs into two categories [8], one as state-independent which produces the output and generates a dynamic page whose structure and links are fixed, another one as state-dependent which provides different output pages when executed under different conditions according to the value of a hidden flag recording a previous user selection. In order to achieve a full comprehension of a Web application, a reverse engineering process should support the recovery of both the static

and dynamic aspects of the applications, and visualize the information with suitable representation models [43].

Static program analyses analyze the program to obtain information that is valid for all possible executions, whereas dynamic analyses instrument the program to collect information as it runs, the results are only valid for a specific execution. Static analyses can provide facts about the software system that the reverse engineer may rely upon; dynamic analysis is needed to obtain more precise information about the Web application behavior, such as generating pages on-the-fly depending on the user interaction.

The absence of implementing the well-known software engineering principles of modularity, encapsulation, and separation of concerns, make the comprehension of an existing Web application harder. Usually, script code implementations of business rules, presentation logic, as well as data management, are scattered within a same page, interleaved with HTML statements.

Some WSRE related tasks on data gathering, knowledge management and information exploration include: extracting and visualizing the Web site structure [1, 2] to identify the pages and the hyperlinks between; to identify their inner page components and associated relationship. Clustering techniques have been adopted to abstract artifacts represented by UML use case diagrams. Some research focus on collecting metrics [11, 12] and statistics of a web site such as its size, complexity, fan-in/out, lines of code, link density; number of in/out links a Web page has; number of pages using a same component; page download time, page access count and referral count. Usage pattern mining to facilitate Web site evolution; Some involves web site versioning—to measure

the rate and the degree of Web page change through server Log files [13] and to compute the differences [2].

The problem of defining techniques and tools similar to software engineering was investigated (e.g., Martin conducted experiments to use the software engineering tool Rigi for web application static analysis [18]). Some approaches for WSRE have been proposed to obtain the architecture that depicts components composing the Web site and the relationships at different degrees of detail [4, 5, 6].

Hassan proposed an approach to recover the architecture of Web applications to help developers gain a better understanding of the existing system and to assist in their maintenance [21, 22, 25]. The approach is based on a set of coarse-grained extractors, which examine the source code of the application such as HTML page, server-side JavaScript and VBScript, SQL database components, and Windows binaries. The extracted multi-language facts are abstracted and merged, and architecture diagrams representing the relations between Web application components are generated.

What is deployed on the Web server may not correspond to a physical file stored on the development environment (e.g., use of template). What the Web server sends to the client may not correspond to a physical file stored on the server (e.g., CGI bins, Servlets, ASP and JSP pages may generate pages on-the-fly). Mappings from pre-generation artifacts to post-generation artifacts need to be identified [7]. To the best of our knowledge, no existing tool or analysis explicitly identifies these different viewpoints or offers mappings between them. We believe that making these mappings explicit will potentially benefit Web site comprehension greatly.

2.2 Web site Reverse Engineering tools

We studied several related research tools to gain a solid understanding of WSRE requirements, its methodology, and its process. Most WSRE research tools have a similar structure. Figure 2.2 depicts the general components of WSRE tools.

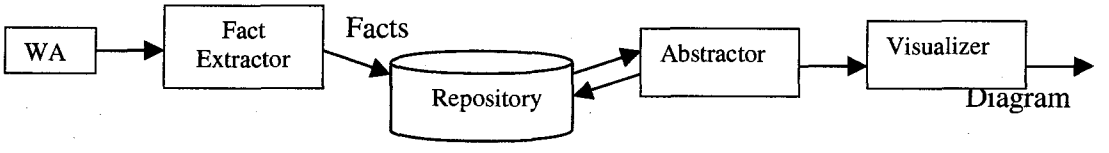

Figure 2.2 WSRE Tool General Architecture

A repository stores information that is needed for reverse engineering and program comprehension functionality. Examples of concrete implementations can range from a simple text file to a relational database. To enable information exchange between components, they must share a data schema. The fact extractor parses source code of a Web application (WA) and populates an intermediate representation of artifacts to the repository. Depending on the domain, there can be several extractors (e.g., extractors for HTML, JSP, ASP, and JavaScript might be necessary for parsing a web application). An abstractor performs certain analyses based on the facts stored in the repository; it recovers a conceptual model of the WA representing its components and the relations between them. The result of an analysis is stored back into the repository. A visualizer presents the extracted information and the results of analyses to the user in an appropriate visual form, typically in a graph editor.

2.2.1 ReWeb

Ricca and Tonella developed the ReWeb tool for Web site structure and evolution analysis [2]. ReWeb consists of a WebSpider, an analyzer and a viewer. The WebSpider downloads all pages of a target web site by sending the associated requests to the web server, providing the input where required. The spider contains an extractor, which recognizes HTML and JavaScript code fragments. The analyzer uses the UML model of the web site, interpreted as a graph, to perform structural and evolution analyses. The viewer reads the files representation of the views, generated from the analyzer, and produces the graphic representation of the structural and history views. Figure 2.3 depicts a sample structural view of a web site [6].

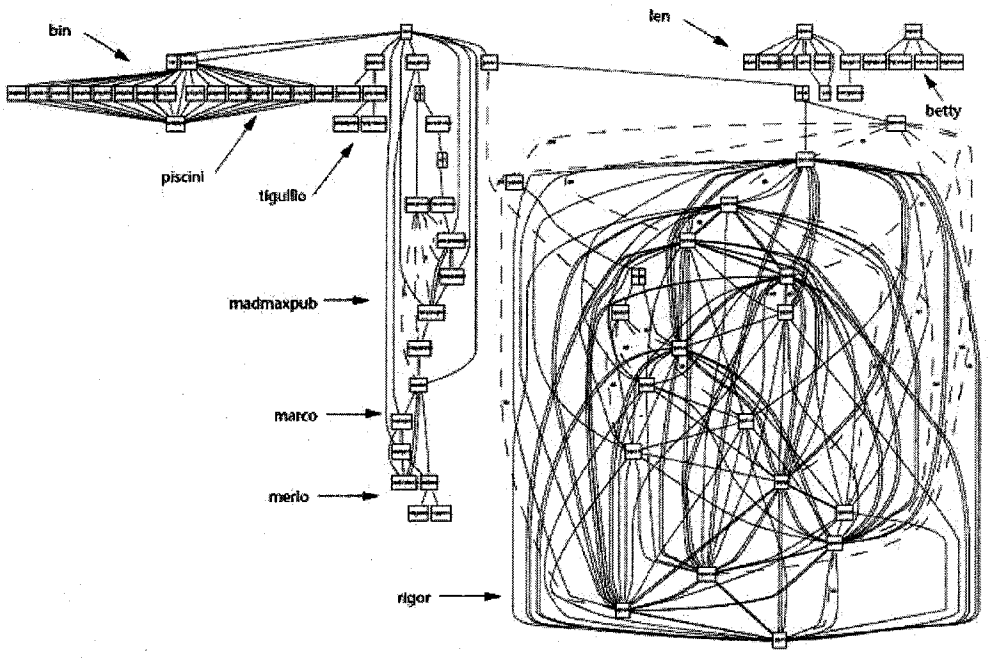

Figure 2.3 A sample view generated by ReWeb

During the structural analysis, the shortest path to each page in the site is computed to indicate potential costs for the user searching a given document; strongly connected components are identified to suggest regions with fully circular navigation facilities, which lead to previously visited pages to allow the user to explore alternative pages; structure patterns are also extracted to help understanding a Web application.

During the evolution analyses, it calculates the difference between each two successive versions of the site, aiming at determining which pages were added, modified, deleted or left unchanged, assuming that the page name is preserved. A range of colors is employed to represent how recent nodes are modified.

2.2.2 WARE

WARE uses UML diagrams to model a set of views that depict several aspects of a Web application at different abstraction levels [1].

The main components of WARE include an interface layer, a service layer, and a repository. The interface layer implements a user interface to provide access to the functions offered by the tool, and a visualization of recovered information and documentation, both in textual and graphical format. The service layer contains an extractor and an abstractor. The extractor parses WA source code and produces Intermediate Representation Form (IRF) of a WA, which are implemented with a set of tagged files, one for each source file. In the IRF files, each tag depicts a specific type of item (such as pages, page components, direct relations between the items, page parameters) and related attributes (such as code line number, form names, methods and actions associated with a form); The abstractor operates over IRF and recovers UML class diagram. The sub components of the abstractor are: a translator, a query executor,

and a UML diagram abstractor. The translator translates IRF into a relational database, the query executor executes predefined SQL queries for retrieving data about the application, such as the list of the page hyperlinks, page components, form parameters, etc. The UML diagram abstractor produces the class diagram of a WA. The IRF, the relational DB populated by the abstractor and the recovered diagrams are stored in the repository.

Structural views and behavioral views are recovered. In a structural view, at a coarse-grained level, server page (pages deployed on the web server) and client page (pages the web server sends back to the client requests) are distinguished and the hyperlink relationships are specified; at the finer-grained level, inner page components are identified and classified along with their interrelationships. As to the behavioral view, the collaborations and interactions between structural components are represented, including interactions triggered by events from code control flow or from user actions; the sequences of interactions are also identified.

WARE used extended UML diagrams to model the WA. The class diagram is used to model the architecture of the WA, which is made up of structural components and relationships among them. Sequence diagrams represent the dynamic interactions between pages, their inner components, and the users. A use case diagram provides a representation of the different behaviors exhibited by the WA. The tool WARE supports the recovery of these UML diagrams from WA source code. Figure 2.4 depicts a generated UML diagram where the classes corresponding to pages and forms have been represented. Each node represents a class, and different shapes are used to distinguish the

different classes: boxes are associated with static client pages, diamonds with server pages, trapezoids with dynamically built client pages, and triangles with forms [1].

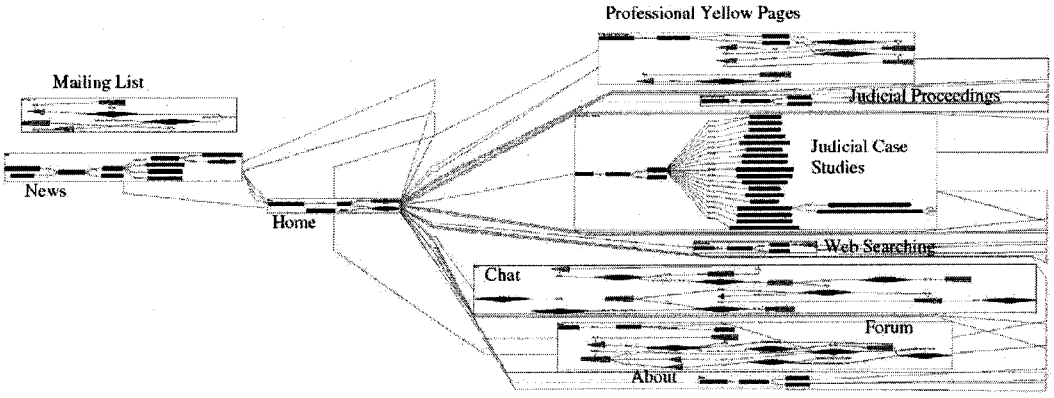

Figure 2.4 Sample UML Diagram Generated by WARE

2.3 Adoption Centric Reverse Engineering

Software engineering research tools are often not evaluated and fail to be adopted by industry due to their potential users' unfamiliarity with the tool, difficult installation, poor user interface, weak interoperability with existing development tools and practices, and the limited support for the complex work products required by industrial software development.

ACRE (Adoption Centric Reverse Engineering) approach hypotheses that in order for new tools to be adopted effectively, they must be compatible with both existing users and existing tools [15]. As mentioned in Section 1.2., our approach is to graft domain-specific functionality (such as support for Web sites) on top of highly customizable baseline tools. Thus, users can leverage the host component's existing (domain independent) functionality (e.g., search, copy/paste, and file save) and the cognitive

support (i.e., the principles and means by which cognitive software processes are supported or aided by software engineering tools) while seamlessly transition to the new (domain dependent) functionality.

Interoperability is another important aspect of the ACRE project suite. The interoperability of the new tools can also be improved by exploiting middleware technologies at various levels including data integration (e.g., XML standards), control integration (e.g., scripting languages and plug-in platforms) and presentation integration (e.g., consistent look and feel). Improving interoperability between forward and reverse engineering tools could facilitate reverse engineering tool adoption.

COTS products are designed to be easily installed and to operate with existing system software. COTS-based software development means integrating COTS components as part of the system being developed. A candidate for a baseline tool needs to be a (major) part of the user's workflow and programmatically customizable.

Various COTS platforms have been investigated in our research group, including IBM Lotus Notes, Microsoft Office, Microsoft Visio, and Adobe GoLive. The ACRE project expects that the new tools will have a higher adoption rate than stand-alone reverse engineering) tools. We also found that the Viewer of ReWeb is based on Dotty [26], a customizable graph Editor for drawing directed graphs developed at AT&T Bell laboratories. This is similar to our ACRE approach on that it also leverages an existing tool product.

2.4 Summary

This chapter presented the background on Web site reverse engineering research and described the main components of selected related research tools and their functionalities.

It also identified adoption problems research tools are facing and proposed a solution based on Adoption Centric Reverse Engineering approach, aiming at improving the tool adoptability and interoperability by grafting new functionalities on top of user familiar, highly customizable existing tools.

Chapter 3 Analysis

3.1 Reverse Engineering Tool Requirements

The reverse engineering tools we introduced in Chapter 2 consist of a few general components (Figure 2.2). In order to realize a reverse engineering environment, for each component, one can choose to reuse and customize an existing component, or to implement a component from scratch. Regardless, in order to be useful, reverse engineering tools have to meet certain requirements. Below we list a number of requirements that are independent of the tools' functionality and domain, and which have been repeatedly identified by researchers in the area.

Scalable: The evolution process is complicated by changing platforms, languages, tools, methodologies, hardware, and target users. One goal of software engineering research tools is to support long term software evolution in an environment of increasing complexity and diversity. Reverse engineering tools are required to handle the scale, complexity, and diversity of large software systems (Requirement 1 in [16]).

For instance, since the subject system can potentially get quite large (millions of lines of code), it is important that the performance of components and the information conveyed by the visualizer is able to scale up [45]. However, the necessary performance depends also on the granularity of the information model. For example, a schema that represents the subject system at a high level of abstraction allows the repository and visualization to worry less about scalability issues.

Extensible: Tilley states "it has been repeatedly shown that no matter how much designers and programmers try to anticipate and provide for users' needs, the effort will always fail short" [46]. Constantly arising new technologies mandate extensibility in RE

systems [47], for instance, to accommodate changing or new repository schemas or extractors [45]. To be successful, it is important to provide a mechanism through which users can extend the system's functionality. Making a tool user-programmable, through a scripting language, for example, can amplify the power of the environment by allowing users to write scripts to extend the tool's facilities. Other options include plug-ins and tailorable user interfaces.

Exploratory: Reverse engineering tools should provide interactive, consistent, and integrated views, with the user in control (Requirement 15 in [16]); it should integrate graphical and textual software views, where effective and appropriate (Requirement 20 in [16]). Information in the repository should be easy to query. Visualized information should be generated in different views, both textual and graphical, in little time. It should be possible to perform user-interactive actions on the views such as zooming, switching between different abstraction levels, deleting entities, grouping into logical clusters, etc. Moreover, the information presented should be interlinked (e.g., the environment should provide every entity with a direct linkage to its source code). Maintaining a history of views of all steps performed by the reengineer is also helpful as it allows returning to earlier states in the reengineering process [45].

Interoperable: Reverse engineering tools must be able to work together to combine diverse techniques effectively to meet software understanding needs [16, 48]. Tool integration can be distinguished at different levels: data integration, control integration, and presentation integration. Data integration involves the sharing of information among tool components, where components use standard data models and exchange formats and manage the data as a consistent whole; control integration entails

the coordination of tools to meet a goal, this can be achieved by enabling components to send messages to each other (e.g., Remote Procedure Call, Remote Method Invocation), one versatile technique for control integration is to use a scripting language; presentation integration involves concerns of user interface consistency, components have a common look-and-feel from the user's perspective, reducing cognitive load [49].

Language/Platform-Independent: If possible, tool functionality should be language independent in order to increase reuse of the components across various target systems.

Adoption-Friendly: A tool is only useful if it is actually used; it needs to address the practical issues underlying reverse engineering tool adoption (Requirement 12 in [16]). Intuitively, to encourage adoption, a new tool should be easy to install, have a steep learning curve, offer documentation and support, etc. Diffusion of innovation theory has identified a number of general characteristics significant to adoption [50]: relative advantage (the degree to which the new is perceived to be better than what it supersedes), compatibility (consistency with existing values and past experiences), complexity (difficulty of understanding and use), and trialability (the degree to which it can be experimented without committing to it).

3.2 Web Authoring Tools

Web Authoring Tool refers to a tool that generates and maintains Web pages. To choose a suitable host tool, we need to investigate the available web authoring tool's existing features and how to utilize them for reverse engineering and comprehension functionality.

Scott Tilley proposed using REEF (REverse Engineering Framework) to evaluate reverse engineering capabilities of Web tools [10]. REEF defines a descriptive model that categorizes important support mechanism features based on a hierarchy of attributes, which can be compared using a common vocabulary. It identifies reverse engineering tasks (e.g., program analysis and redocumentation) and defines three canonical reverse-engineering activities: data gathering, knowledge management and information exploration. Program analysis is syntactic pattern matching in the programming-language domain, such as control-flow analysis and slicing. Redocumentation is the process of retroactively providing documentation for an existing software system. Data gathering gathers the raw data about the system (i.e., artifacts and relationships between artifacts). Knowledge management structures the data into a conceptual model of the application domain; Information exploration analyzes and filters information with respect to domain-specific criteria. This task is the most important and most interactive of the three canonical activities. The software engineer, typically a maintenance programmer, gains a better understanding of the subject system with interactive exploration of the information that has been obtained by data gathering and knowledge management. Extensibility is also an important quality attribute included in the REEF model.

A recent survey conducted by SecuritySpace [9] showed that Microsoft FrontPage, Adobe GoLive and DreamWeaver occupied most of the market of Web site authoring tools. Based on the REEF framework, we evaluate the reverse engineering capabilities of these Web tools in the following sections.

3.2.1 Microsoft FrontPage

Microsoft FrontPage 2002 includes a visual editor. It supports CSS (Cascade Style Sheet), templates, browser plug-ins, database contents, applets, JavaScript, ActiveX controls, and Microsoft Visual Basic. For existing sites, FrontPage offers an import function. To facilitate management capabilities, FrontPage offers a view of navigational links, folders, and all files, as well as automatic hyperlink updates. Yet many developers consider FrontPage to be a low-end tool appropriate for less intricate projects. We evaluate its reverse engineering capabilities with the following attributes:

Program Analysis: the XML Formatting feature of FrontPage helps reformat HTML tags to make an HTML page XML-compliant, useful for interacting with an XML-based publishing system. The HTML Reformatting provides the capability to reformat an HTML page with the formatting preferences such as the number of indents before each tag, tag color, and whether or not to use optional tags.

Redocumentation: Share Point Team Services and Task Views record the tasks of the site development performed by the team. If used from the initial development stage, this documentation provides concise record of the development of the site.

Data Gathering: After publishing to a Web server with FrontPage extension, the Publishing Log File records when and what was published onto the web. The Usage Analysis Report can show the page hit statistics, slow or broken hyperlinks, the number of external and internal hyper links, unlinked files, and recently added or changed files. The Auto Filter shows the interested site report information such as oversized image files.

Knowledge Management: FrontPage provides a built-in mechanism for creating and managing site-wide navigation. A web page can be dragged from the page file Folder

List Window and dropped into the Navigation window to form a navigation path, this action causes the pages entered into the navigation records to be managed automatically. So when that page is selected in the Folder List window, the corresponding page in the Navigation window will be highlighted. But this action does not automatically create a corresponding hyperlink in the source page.

Information exploration: Multiple views of the Web site are provided: the Navigation View shows an overhead look at the structure of a web site, the Hyperlinks View presents a visual map of the hyperlinks to and from any page in the site.

Extensibility: FrontPage enables task automating with macros consisting of a series of commands and functions stored in a VB module.

3.2.2 Macromedia DreamWeaver

Macromedia DreamWeaver MX is a visual tool for building Web sites and RIA (Rich Internet Applications). It supports CSS, CSS-P (CSS positioning), Netscape Layers, JavaScript, XML, SVG, and various server technologies such as ASP.NET, ASP, JSP, PHP and ColdFusion. Some site management capabilities are also included.

Program Analysis: When importing a page generated from Microsoft Word, DreamWeaver can clean up the redundant and Word-specific HTML tags with the "Cleaning up Word HTML" feature; Dreamweaver highlights the invalid HTML in the Code View according to user-specified HTML version; Auto Tag Completion and Code Hints make the HTML coding more efficient; Code Navigation lists JavaScript and VBScript contained in a page opened in the Code View.

Plan Recognition: Library items are used for individual design elements, such as a site's copyright information or a logo; Templates controls a larger design area, a template

author designs a page and defines which areas of the age can accept either design or content edits. The Assets Panel feature of DreamWeaver provides access to these libraries and templates, so that editing a library item or template updates all documents in which they have been applied.

Redocumentation: DreamWeaver Design Notes are notes that a developer creates for a file, keeping track of associated information such as current design thoughts and status, which can be used to ease communication among development team members; the Workflow Report is used in a collaboration environment, displaying who has checked out a file and which ones have Design Notes associated with them. By consistently being kept up-to-date, Design Notes and Workflow Report can be taken as a form of redocumentation about the Web site development;

Data Gathering: DreamWeaver can report broken internal links, orphaned links, validate markup and XML, and check page accessibility. The Get command copies files from the remote site or testing server to the local site.

Knowledge Management: DreamWeaver Site Map can be used for laying out a site structure, or to add, modify, remove links; the Live Data Preview feature enables viewing and editing server site data in the workspace and making edits on the fly; Link Management automatically updates all links to a selected document when it is moved or renamed.

Information exploration: The Site Panel enables viewing of a site's local and remote files; the Site Map displays the site structure; and the Live Data window displays the web page using the testing server to generate the dynamic content.

Extensibility: Extensions can be built in HTML and JavaScript or DLL in C. DreamWeaver provides an HTML parser and JavaScript interpreter as well as APIs to facilitate extension. The DreamWeaver DOM (Document Object Model) represents tags and attributes as objects and properties and provides a way for documents and their components to be accessed and manipulated programmatically. DreamWeaver checks extensions during start up, and then compiles and executes procedures between opening and closing <script> tags. Some types of tasks that extensions typically perform are: automating changes to the document; interacting with the application to automatically open or close windows or documents; connecting to data sources; and inserting and managing blocks of server code in the current document.

3.2.3 Adobe GoLive

Compared to FrontPage, Adobe GoLive 6.0 is a heavyweight Web site design and development tool with a large user base. Compared to Dreamweaver, it has greater strengths in site planning, dynamic design, integration with other applications, data-driven publishing and site management. It supports CSS formatting control, JavaScript, XML, SVG, and server technologies ASP, JSP, and PHP. It is a consistent work environment with other Adobe tools including Photoshop, Illustrator, LiveMotion and Premiere. GoLive provides numerous controls that ease page development. The Layout Grid, for instance, generates an HTML table automatically while being dragged and dropped onto the pages.

We now evaluate GoLive's capabilities in reverse engineering related tasks and activities:

Program analysis: GoLive has Clean Up Site and Remove Unused commands to remove unused colors, font sets, links, etc. Fix Errors reports missing files and links. Check External Links tests whether external links are valid. The Syntax checker can parse the source code to verify if a document (HTML or XML files) meets standards of a particular browser version or at a particular DTD. Lastly, Site Report looks through the site for accessibility-related problems (e.g., missing ALT attributes).

Redocumentation: In GoLive, developers can use design diagrams to record their initial design of the Web site structure. If this diagram exists, it can be a good starting point for redocumentation. A design diagram shows pages, (potential) navigations between pages, and hierarchical relationships between pages. Similar to UML diagrams, design diagrams can contain annotations. Navigations that are proposed in the design diagram, but have not been realized yet, are shown in the Pending Links view. The Navigation view shows the hierarchical structure of the site. Thus, these views can be used to assess differences between the proposed design and the actual site.

Site data gathering: GoLive can import a Web page from the internet, including associated components (e.g., image files, CSS files and script library file). It can also import sites from ftp or http servers, including Web pages and related components, external links; the Site Report mechanism enables the query of site information based on file size, estimated download time, date of creation, html errors and usage of protocols.

Knowledge management: Typically, the conceptual model of a Web site shows pages and navigations between pages. GoLive has several views to summarize, navigate, and manipulate this information. The Navigation view shows the hierarchical organization of pages. The In & Out Links view is a link management tool that

graphically shows the links to or from a selected file. Similarly, the Links view shows the recursive link structure starting from a certain file. The Revision Management feature compares different versions of a file through Work Group Server. It also lists full details of who made certain changes to what, along with the time that the changes were entered.

Information exploration: GoLive represents information with views. There are a large number of views, showing various properties of the Web site. The Files view lists the files (e.g., pages, images, and scripts) belonging to a Web site. Some views focus on a single page (e.g., Source Code Editor and Layout Preview), while others show relationships between pages (e.g., In & Out Links and Navigation). Various interactions between views can aid the software engineer during exploration. For example, selecting an icon in one view, can trigger the display of more detailed information about the selection in another view. The Split Source view simultaneously shows a page's layout along with its underlying HTML source code. Changes and selections in either view are immediately reflected in the other. While GoLive offers information exploration with views, it has no graph visualization, which is now the preferred visualization of most program comprehension tools. As a result, information in GoLive is dispersed over several views. A complementary graph visualization providing a unified view of a Web site along sophisticated manipulations such as hierarchy building would be desirable.

Extensibility: An extension can obtain content and services from the GoLive design environment, from other extensions, and from resources on local and remote file systems. JavaScript DOM in GoLive provides access to the markup elements which enables programmatic editing of files written in HTML, XML, ASP, JSP and other markup languages. An extension can also create/customize GoLive user interfaces (e.g.,

Menus, Dialogs, Palettes, Inspectors, Site window, Document windows, Site reports), user settings (e.g. global style sheets, preferences), and automations/macros (e.g., applying automated edits to every file in a site, or generating entire sites programmatically).

3.2.4 Comparison

Table 3.1 lists the reverse engineering capabilities of each web authoring tool:

	FrontPage	DreamWeaver	GoLive
Program Analysis	HTML reformat, XML formatting	HTML Validation Code navigation	Syntax check Site Clean Site report
Redocumentation	Task view	Design notes Workflow report	Design diagram
Data Gathering	Publish log Usage analysis report	Get file from remote site Broken link report	Import web page from internet and ftp/http server; Site report; Document scanning
Knowledge Management	Navigation management	Site map Live data preview Link management	Navigation view In/out links view Revision management
Information Exploration	Navigation view Hyperlink view	Site Panel Site map Live data window	Files view Source code/layout view In/out link/navigation view
Extensibility	Only supports Macros	End-user programmable in HTML, JavaScript, C	End-user programmable in JavaScript and external library in C/C++ Supports automation and macros

Table 3.1 Tool Reverse Engineering Capabilities

Based on the reverse engineering features of these tools, we found that GoLive has relatively strong capabilities with respect to data gathering, knowledge management, information exploration and extensibility. Compared with the other two, GoLive is our preferred Host Tool.

3.3 GoLive Customization

3.3.1 Customization Options

The GoLive SDK (Software Development Kit) provides numerous JavaScript objects and methods to perform tasks on a document, on a site, in the GoLive environment, on local and remote file systems, and on DAV (Distributed Authoring and Versioning) servers. It allows users to programmatically:

- Add a menu bar, define a menu and menu items, implement the menuSignal function
- Define Modal Dialog windows and modeless Palettes
- Create Custom Elements, implement Custom Element event-handling functions (The created Custom Element, represented as an icon, can be dragged to a page from the Objects palette to add predefined HTML elements to a page or site.)
- Edit markup documents with JavaScript and DOM, retrieve and modify Markup elements to manipulate the contents of pages and sites
- Create files/folders, open existing files, read file content, retrieve content of a folder, delete, copy and move file/folder, and save documents
- Manipulate a Web site, including the files, selections, and custom column content in an open site window; generate custom report about a site.
- Connect to the WebDAV (Web Distributed Authoring and Versioning) server, retrieve site resources, and get Metadata of resource, upload/download files to server.
- Communicate with other extensions

3.3.2 Customization Methods

The GoLive SDK enables customization via so-called Extend Scripts [24]. Building an Extend Script extension includes creating a Main.html file, which contains JavaScripts and special GoLive SDK supplied tags (identified with the prefix jsx). The JavaScript code contained in a <script> element in the Main.html file consists of user-defined functions and implementations of GoLive Event-Handling Functions. The special tags declaratively define menus, dialogs, palettes, inspectors and custom tools in the GoLive design environment.

The extension file needs be placed in a subfolder of the GoLive Extend Scripts folder. At startup time, GoLive interprets these tags and scripts, and loads an extension into the GoLive environment. Depending on the extension, the Extent Script has to implement a number of JavaScript call-back functions, which are invoked by GoLive to signal events. When an event is triggered, GoLive calls the event-handling function. For example, when the user interacts with an extension's custom menu, dialog, or palette, GoLive calls the appropriate event-handling function. If the extension provides that function, GoLive executes it; otherwise, the extension ignores the call to that function.

At application start-up, GoLive calls each extension's initializeModule() function. To give a flavor what Extent Scripts look like, here is a "Hello World" example:

```
<html><body>
<jsxmodule name="MyExtension">
<script>
function initializeModule()
{ alert ("Hello, World!") }
</script>
</body></html>
```

GoLive's SDK provides numerous JavaScript objects and methods to programmatically manipulate files and folders as well as the content of documents written in HTML, XML, JSP, etc. The document content that has been read into memory is made available in GoLive through a DOM, which allows it to query and to manipulate markup elements. Thus, batch-processing of changes to an entire Web site can be easily accomplished. Notebly, since Extent Scripts are essentially HTML/XML documents, they can be easily edited in GoLive itself.

3.3.3 JavaScript programming

There are two kinds of high level languages: System Programming Language and Scripting Language. A system programming language (application language) is typed and allows arbitrary complex data structures. Programs written in them are compiled, and are meant to operate largely independently of other programs. A scripting language is weakly typed or untyped, and has little or no provision for complex data structures. Programs in them are interpreted. Scripts need to interact either with other programs (often as glue) or with a set of functions provided by the interpreter.

JavaScript is a lightweight interpreted programming language with rudimentary object-oriented capabilities. The general-purpose core of the language has been embedded in Netscape Navigator and other Web browsers and extended for Web programming with the addition of objects that represent the Web browser window and its contents. The JavaScript Document objects, and the objects they contain, allow programs to read, and sometimes interact with, portions of the document.

When GoLive SDK interprets the markup elements in an extension, it creates objects, and the attributes of elements are interpreted as properties of JavaScript objects. Objects that represent the content of html pages are available from the markup tree that the page's Document Object provides.

3.4 Summary

This chapter elaborated the general requirements of reverse engineering tools. It analyzed the reverse engineering capabilities of some web authoring tools based on the REEF framework. After comparing several tools, GoLive was selected as the host tool for our case study. Hence, its customization options and methods were investigated and its related scripting language was studied.

Chapter 4 Design and Implementation of REGoLive

This chapter discusses the design rationale and implementation methodology of REGoLive. We begin by analyzing which features of Adobe GoLive can be leveraged during reverse engineering tasks and which functionality can be used to build extensions on top of it. In Sections 4.2 and 4.3 we then present the design and implementation process. Section 4.4 documents our development experiences.

4.1 Requirements

Our design is based on the analysis of the cognitive support GoLive provides and the Web Site Reverse Engineering (WSRE) functionality software engineers require. Our ultimate goal is to leverage those features that help satisfy these requirements.

4.1.1 Supportive Features

In order to leverage the cognitive support provided by GoLive, we specifically investigated the functionality inherent in reverse engineering tools. In particular, we concentrated on analyzing GoLive's parsing, interoperability and visualization capabilities. We studied the documentation of GoLive 6.0 SDK [24] and found a number of useful GoLive objects. Tables 4.1, 4.2, and 4.3 list some of the selected object properties and functions as well as their potential applications.

Parsing is essential in WSRE processes. GoLive provides parsing through the DOM API which allows programmatic access to the document structure. GoLive DOM enables editing of Markup documents programmatically through its document object model by retrieving and modifying Markup elements. This provides the capability to parse a Web site and its documents to retrieve artifacts that are of interest to Web site

maintainers. GoLive SDK supplies various JavaScript objects. The Web site object manipulates the site that is currently open in the GoLive design environment. It provides a Site Reference iterator to access all files in the site; each Site Reference object represents one file along with its outgoing and incoming references. Moreover, the Markup object enables a programmer to retrieve element attributes for a particular document in a Web site. The File object has the capability to download a file from a remote URL, which is useful for Web crawling.

GoLive Object	Properties	Functions	Description
Web site	root(SiteReference)	selectedfiles()	selects specified files in the site window
	homepage(SiteReference)	getSelectedFiles()	returns references to all selected files in the site window
SiteReference	name(String)	getIncoming() / getOutgoing()	retrieves SiteReference objects that link to/referenced by this siteReferenceObj page
	url(String)		
	type(String)		
	siteDoc(Document)	open()	opens this siteReferenceObj page in GoLive, returns documentObj
siteRefIterator	[index]	first() / next()	Returns the first / next element of the siteReference collection
document	element(Markup) homepage(SiteReference)	selectElement()	Selects the specified element
Markup	elementType(String)	getInnerHTML() / getOuterHTML()	retrieves the markupObj element's HTML representation, excluding/including the outmost tag delimiters
	subElements(collection)		
	tagName(String)	getSubElementCount()	retrieves the count of a specified tag elements found among this element's subelements
		getSubElement()	retrieves a subelement of the markupObj element by name, index or type
		getAttribute()	returns a string of the value of a specified attribute

Table 4.1 GoLive Objects Useful for Parsing

Tool interoperability is necessary to combine diverse techniques effectively to meet software comprehension needs including data, control, and presentation integration. Existing tools have a variety of capabilities. It is useful to combine various facilities to provide useful general services.

With the help of the GoLive Application, Document and JSXFile objects, we can create files, open existing files and save documents. Using these objects, we can generate text files in various formats including RSF (Rigi Standard Format) [17] and XML formats (e.g., GXL or SVG) to share data among tools to fulfill the requirement of data integration. Hence, we can use the local file system as a repository to store software artifacts.

GoLive Object	Properties	Functions
File	Name Path url	Open() openDocument() openMarkup() read()/readln() write()/writeln() copy() get()/put() remove() createFolder()
App		launchURL() openDocument()

Table 4.2 GoLive File Object and App Object

GoLive's scriptability enables programmatic interoperability with other tools. Operational interoperability can be achieved by invoking a Web service via GoLive's Application object which launches a URL on the server.

Presentation integration and visualization capabilities can be achieved by programmatically customizing the GUI and its views with GoLive-provided widget

objects such as menus, modal dialog windows, palettes (modeless dialog or floating window), and custom controls. Functions of the Draw object allow us to draw primitive graphics, text and images. These capabilities can be used for visualization tasks and to obtain the same look-and-feel as the GoLive development environment. Using the File object, it is also possible to generate SVG files from GoLive.

GoLive Object	**properties**	**functions**
Menu	Name Selection title	addItem() removeItem()
MenuItem	dynamic	menuSignal() event
Draw		drawstring() fillOval() fillRect() lineTo() moveTo() setColor() penSize()
Dialog	Name(string) Title(string) Visible(boolean)	exitModal() runModal()
Palette	Name(string) Title(string) Width(integer) Height(integer)	
CustomControl	PosX(integer) Posy(integer) Width(integer) Height(integer)	mouseControl(x, y, mode) beginDraw() endDraw() refresh()

Table 4.3 GoLive Objects Useful for Visualization

4.1.2 Selected Reverse Engineering Tasks

A Web application is usually developed with the help of a Web authoring tool. Using these tools, developers visually layout text, images, anchors and many other objects as

well as apply structural, graphical or interactive attributes to the contents on the Web page with little hand-coding. As shown in Figure 4.1, pages deployed on the Web server are static or active pages. Static pages contain HTML code and embedded scripts that execute on Web browsers without server-side preprocessing; active pages such as JSP (Java Server Pages) and ASP (Active Server Page), however, require preprocessing on the application server, which may integrate data from Web objects or databases to generate the final HTML page.

To aid a Web site maintainer in the tasks of structuring and understanding large Web sites, and to improve the maintainability and evolution of such sites, different views need to be extracted [7]. A Web site is often represented by means of a developer view, a server view and a client view as depicted in Figure 4.1. The developer view is what a developer sees in the development environment where the Web site is built. The server view is what a maintainer sees on the Web server where the Web site is deployed. The client view is what a user sees on the Internet, typically using a Web browser.

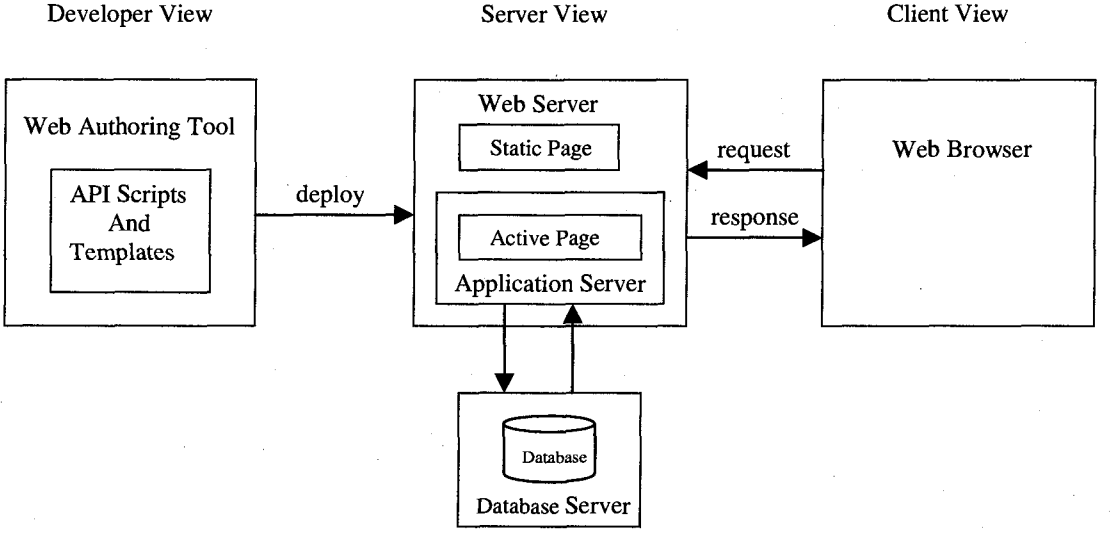

Figure 4.1 Structure of a Web Application

The server view represents a Web site's resources on the server, such as HTML pages, JSP, configuration files, and databases. It shows a Web site structure at different levels of abstraction. At the higher level, it identifies Web pages and their hyperlink relationships to give the developer a quick overview of the site structure and its complexity; at the lower level, it identifies inner page components and their associated relationships.

The developer view can be very different from the server view when the generative techniques of Web authoring tools are exploited. Templates and smart objects in GoLive, for example, can generate target code when publishing a Web site to the server. If certain maintenance and reengineering tasks are to be performed (e.g., migrating a Web site from GoLive to Dreamweaver), we want to identify the GoLive-specific generative components in the developer view, so as to estimate the potential costs of the migration process.

The client view differs from the server view in that dynamic, server-side technology, such as JSP and servlets, generate target code on the fly for the client view. Often the server program exhibits different behaviors based on a condition or user input—different computations may be performed to produce different Web pages for the client. The Web server hosting pages and server programs are treated as a black box in this case and only its rendered output pages are accessible. Some Web authoring tools are capable of detecting dead links in the development view, but not usually in the client view. However, only the dead links in a client view reflects the real dead links.

Web authoring tools such as GoLive or Dreamweaver often only provide the developer view and lack the server view and/or the client view. The developer view often

offers limited user interaction exploration operations. We do not know of any WSRE tool that generates all of these views. We believe that with the user in control, generating different interactive, consistent, and integrated views from GoLive and establishing mappings among them can facilitate Web site understanding.

4.2 Design

After analyzing GoLive's supportive features and selected reverse engineering tasks that potentially aid program understanding, we started the design of REGoLive. With the requirements in mind as well as considering cognitive support and constraints, we designed the REGoLive infrastructure, functionality and visual metaphor.

4.2.1 Infrastructure

The major components of REGoLive were designed by leveraging the cognitive support of GoLive with respect to data collection, analysis and visualization.

Figure 4.2 shows the basic structure of REGoLive. To tackle the required reverse engineering tasks, we first need to retrieve artifacts from the subject system—the development environment, the server, and the client. From a developer's perspective, the subject system includes the Web pages developed as well as the various objects provided in the Web authoring tool. From the server's perspective, it includes all the source code of the Web site, the application objects, the Web server logs, and databases. From a client's perspective, the server is treated as a black box and the subject system consists of the pages generated and sent from the server and displayed in the Web browser. To extract software artifacts from all of these resources, different extractors are needed. This extraction process involves parsers that automatically traverse the source code with the

support of GoLive. The interested software artifacts express site structure, page components, etc. and should be detailed such as HTML page "p1" links to JSP page "p2", or at a lower level such as form "f" contains input "name". The extracted artifacts can be represented as entities, relationships, and attributes stored in XML or RSF files.

Next, the Data Analysis Engine manages the extracted information through reformatting and computing. It builds multiple, layered hierarchies as higher-level abstractions to reduce the complexity of understanding large Web applications. It produces a spatial site structure, at various levels of details, identifies and builds the mapping between the development environment, the Web server and the client.

Finally, the Visualization engine gives a graphical presentation of the site structure. We could either customize the GoLive visualization capability, or generate an independent graph using SVG as the visual representation of the Web application, or both, to allow exploration of the generated views with spatial and visual operations. The interaction between SVG views and GoLive views has to be enabled to improve tool interoperability.

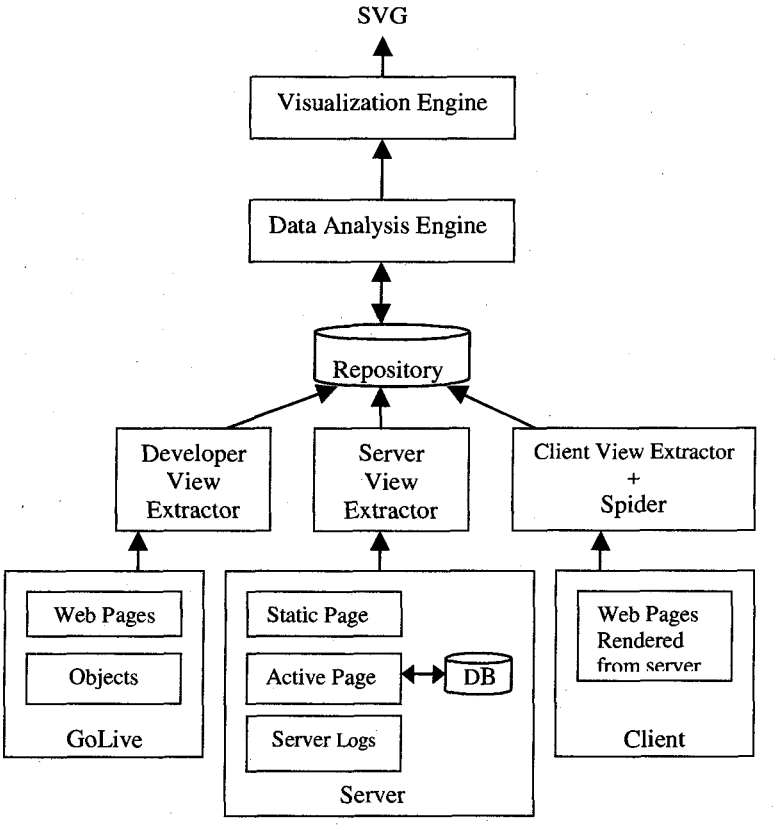

Figure 4.2 Architecture of REGoLive

4.2.2 Functionality

To facilitate the understanding of Web site structure for maintenance tasks, we feel that generating multiple views representing the Web site is helpful in the comprehension process, as outlined in Section 4.1.2.

To obtain the Web site structural information at a coarse-grained level, pages and frame sets are identified. We distinguish between the server pages, static client pages, and dynamic client pages. The relationships among pages or page components include static and dynamic hyper links, links between server pages to other systems such as a DBMS. At a finer-grained level, inner page components are identified including text,

image, input/output form, text box, multimedia object, active component such as script and applets. Submit relationships between a form and the server page, redirect relationships between a script and a Web page, and include relationships between a client script and client module relationships are also specified.

RSF [17] is an input format for the RigiEdit software reverse engineering tool as well as several other tools as data import/export format. (e.g., Martin built a Web2Rsf parser and transformation tool to generate an RSF file from a Web site [18]). To recover the structure and navigation schemes of a Web site, we chose unstructured RSF as the intermediate data format for future processing. To construct RSF triples, we need to define the node, arc and attribute types.

To obtain the collaboration and interaction information between structural components, interactions triggered by events from code control flow and user actions along with their sequences need to be identified.

4.2.3 Visual Metaphor

The purpose of Web site information visualization is to help a maintainer or developer understand and explore the information space. Using nodes and arcs to represent entities and relationships is widely accepted in the research community. In our case, nodes represent Web pages and components, arcs represent navigational or composition relationships. We use different colors to differentiate between types of entities and relationships.

We could either customize the GoLive GUI or extend the visualization capability with SVG. With GoLive GUI widget objects and Draw objects, we can perform basic drawing operations. The problem with this approach is the lack of programmatic mouse

control over the graph generated within GoLive dialog/palette since the related APIs have not been released by Adobe.

To solve the user interaction problem, we use SVG as the visualization format. SVG is an XML grammar for Web graphics from the W3C [27]. SVG combined with ECMA Script can produce rich interactive graphics for live documents. Because of its XML contents, it can also be linked to back-end processes, databases and other valuable sources of real-time information. By generating SVG dynamically from GoLive and integrating with the ACRE SVG Visualization Engine (ASVE) [19], which is a graph visualization engine for exploring and annotating software artifacts developed at the University of Victoria, we gain live, explorable and interoperable presentations.

Web pages and components can be represented as nodes, and the relationships are represented as arcs in the SVG documents. Using ECMAScript, we add additional graph manipulation functionality on top of SVG for further exploration.

4.3 Implementation

This section illustrates the implementation of our design, the prototype REGoLive. We first introduce how to collect interesting artifacts from the subject system on the server, the development environment, and the client; then present the information processing and visualization capabilities, and finally demonstrate how to use this prototype to facilitate selected Web application reverse engineering tasks.

4.3.1 Data Extraction

To obtain the site structure at the Web page level, the REGoLive extractor starts at a specified initial URL such as the home page and first determines the MIME type of the

URL. If it is an HTML page, it extracts all links to other URLs, adds them to the processing queue and subsequently analyses these in the same manner as the initial URL. The Site Reference object of GoLive provides access to all outgoing URLs using a site reference iterator, which is very useful in the parsing process. Figure 4.3 shows part of the sample source code for this extraction process.

```
function processURL(siteRefObj)
{
      …
      var siteRefIterator=siteRefObj.getOutgoing();
      var siteRef=siteRefIterator.first();
      while(siteRef != null){
            isHTML=getType(siteRef.url);
            if(isHTML){
               for ( var j=0; j< i; j++){
                  if (pageQueue[j].siteRefObj == siteRef){
                        break;
                  }
               }
               if(j == i){
                  pageQueue[i] = new
      SiteRef(siteRef,level,false,null,null,null);
                  i++;
               }
            }
            siteRef=siteRefIterator.next();
      }
      …
      while( indexPop < i){
            processURL(pageQueue[indexPop++]);
      }
      return;
}
```

Figure 4.3 Sample Source Code for Page Data Extraction

To obtain the inner structure of a Web page, the extractor first obtains the selected files from the Web site object, and then associates a Document Object with each file; we then get the markup tree of each document, and finally obtain all subelements of the markup tree which is kept in the element Collection object, as depicted in Figure 4.4.

```
var file=selectedFiles.first();
var document = file.open();
var mkupTree = document.element;
var elemCollect=mkupTree.subElements;
...
var elemtObj=new ElemtObj(eleCollect[0], 1, true,null,null,1);
processInnerCom(elemtObj);
...
function processInnerCom(elemtObj)
{
      ...
      elemt=elemtObj.elemt;
      if(elemt.subElements == null){
                  return;
      }
      var elemtIterator=elemt.subElements;
      var siteRef;
      for(var j =0; j<elemtIterator.length; j++){
            elmt=elemtIterator[j];
            if(chkElemType(elmt.tagName))
            {
                  comQueue[i] = new
                  ElemtObj(siteRef,level,false,null,null,null);
                  i++;
                  ...
            }
      }
      while( idxPop < i){
            processInnerCom(comQueue[idxPop++]);
      }
      return;
}
```

Figure 4.4 Sample Source Code for Inner Page Component Extraction

To extract attributes of an HTML tag (e.g., the "action" attribute of "form" tags), we may use the following source code:

```
var elmCount=doc.element.getSubElementCount("form");
for(var j=0;j<elmCount;j++){
      var elmt = doc.element.getSubElement("form",j);
      var att=elmt.getAttribute("action");
      ...
}
```

As intermediate results, corresponding RSF and XML files are generated at the end of the extraction process, which mainly captures the relationships between pages and inner page components.

The server view extractor works on a server side copy of the Web site, which can be obtained from the GoLive FTP browser of the FTP server connection through which we deployed the Web site.

The developer view extractor works on a Web site residing in the GoLive environment. It differs from the server view extractor in that it mainly identifies the GoLive generated components, such as templates and smart objects, which may generate source code when deploying the Web site onto the Web server. Using templates increases development efficiency; it brings consistent style to the pages and avoids inadvertently corrupting code. Templates are often used in banner and navigation bars of Web pages in a site. In GoLive, page templates are stored under subfolder ".data/templates" of a site's directory. To identify a template, our REGoLive programmatically accesses all page objects in the list, and for each page object, searches for the references to any templates. If any templates are found, it then retrieves the source code representation of the template along with its location information. This process also produces intermediate results: RSF and XML files. Some common smart objects include smart Photoshop, smart Illustrator, smart LiveMotion, Component, and browser switch. The Browser switch object, for example, allows the user to add a browser-switch script to the head section of a Web page that detects the Web browser loading the page and automatically redirects viewers to an alternate page based on their browsers. To identify the usage of the Browser Switch object, we can extract the csbrowser markup object from the document and its generated JavaScript for page redirecting by calling the getInnerHTML() method of the markup element.

Client side data extraction is a little more complicated. To obtain artifacts of a Web site on the client side, the spider of REGoLive crawls the Web site starting from the home page URL on the server. It then downloads all the linked pages as depicted in the sample source code of Figure 4.5. If a linked page is a server program (e.g., through a Form submit action or a stand-alone hyperlink), which might generate dynamic pages, the source page will then be launched in a Web browser (e.g., in Figure 4.6, REGoLive opens a Web browser to launch a page containing a form request to a JSP program) and users will be prompted to input values where it affects the server program behavior (e.g., Figures 4.7 and 4.8 show result pages based on user inputs). Input values are embedded in request URLs stored in server log files (e.g., Figure 4.9 shows the affected entries in Web server logs). This process is repeated a number of times until the input values cover all the relevant navigations. The URLs containing the input data can then be extracted from the Web server log, the crawler then downloads the resulting URLs in the log file (Figure 4.10 shows the resultant URLs from Figure 4.9, the downloading procedure is listed in Appendix A). The downloaded pages are merged if they represent the same behavior of a server program.

```
var dlURL=siteRefObj.url.substring(rootIndex-1);
var filename = siteRefObj.name;
var filename2=dlURL.substring(1);
var filename3=filename2.replace(/\//g, '\\');
var idx=filename3.indexOf(filename);
var foldername = rootDir + filename3.substring(0,idx);
var subfolder=filename3.substring(0,idx-1);
var idx2= subfolder.indexOf("\\");
var curfolder=rootDir;
var remain=subfolder;
if(idx2 == -1){
      exePage = JSXFile(foldername);
      exePage.createFolder();
}
else{
      var folderAry=subfolder.split("\\");
```

```
        for(var i=0; i< folderAry.length; i++){
                curfolder=curfolder + folderAry[i] +"\\";
                exePage = JSXFile(curfolder);
                exePage.createFolder();
        }
}
exePage = JSXFile(rootDir + filename3);
exePage.get(WebSvrURL + dlURL);
```

Figure 4.5 Sample Source Code for Static Page Downloading

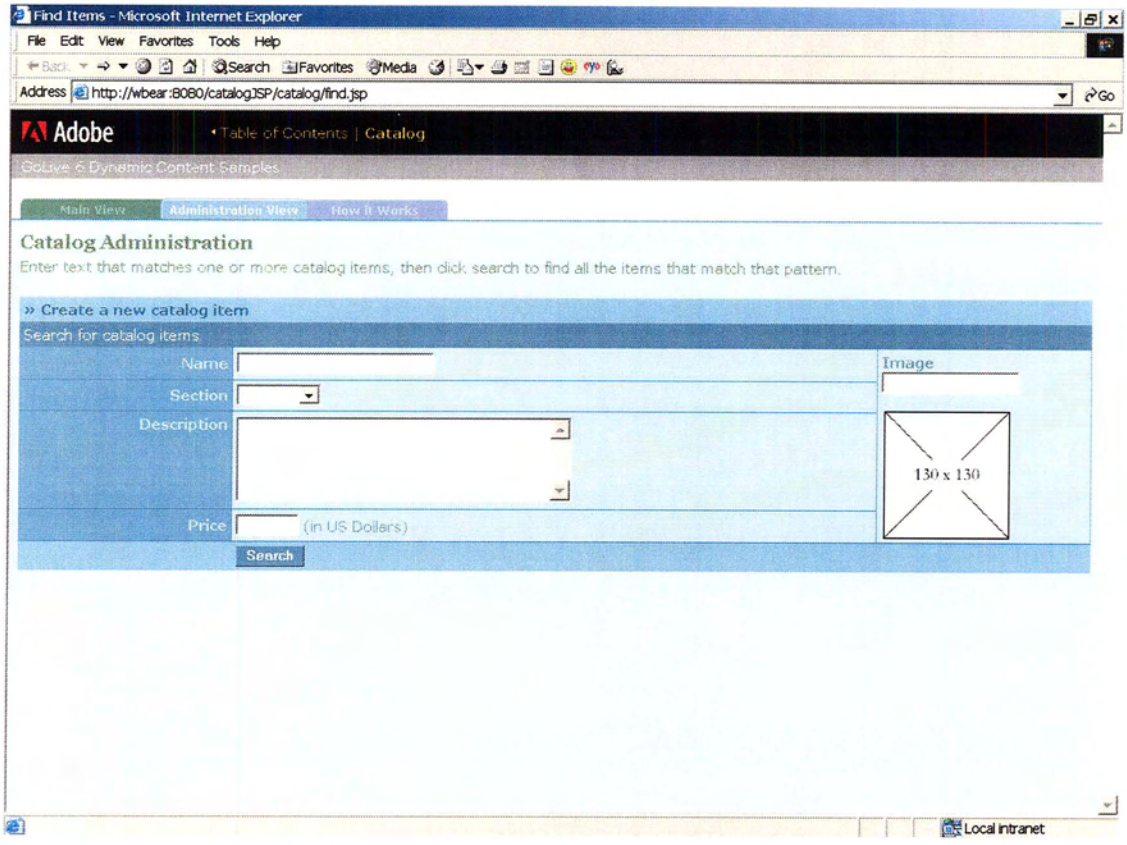

Figure 4.6 Screenshot of Page "find.jsp"

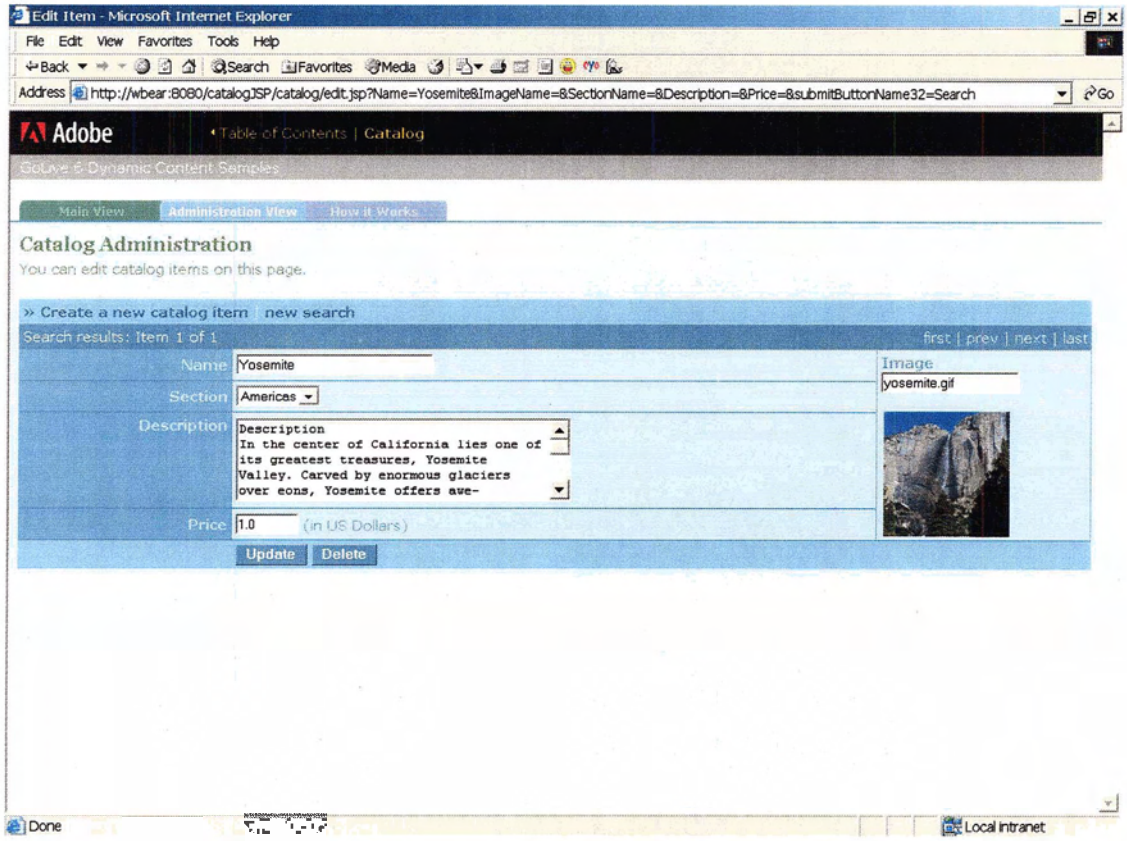

Figure 4.7 Screenshot of a Successful Query Result Page

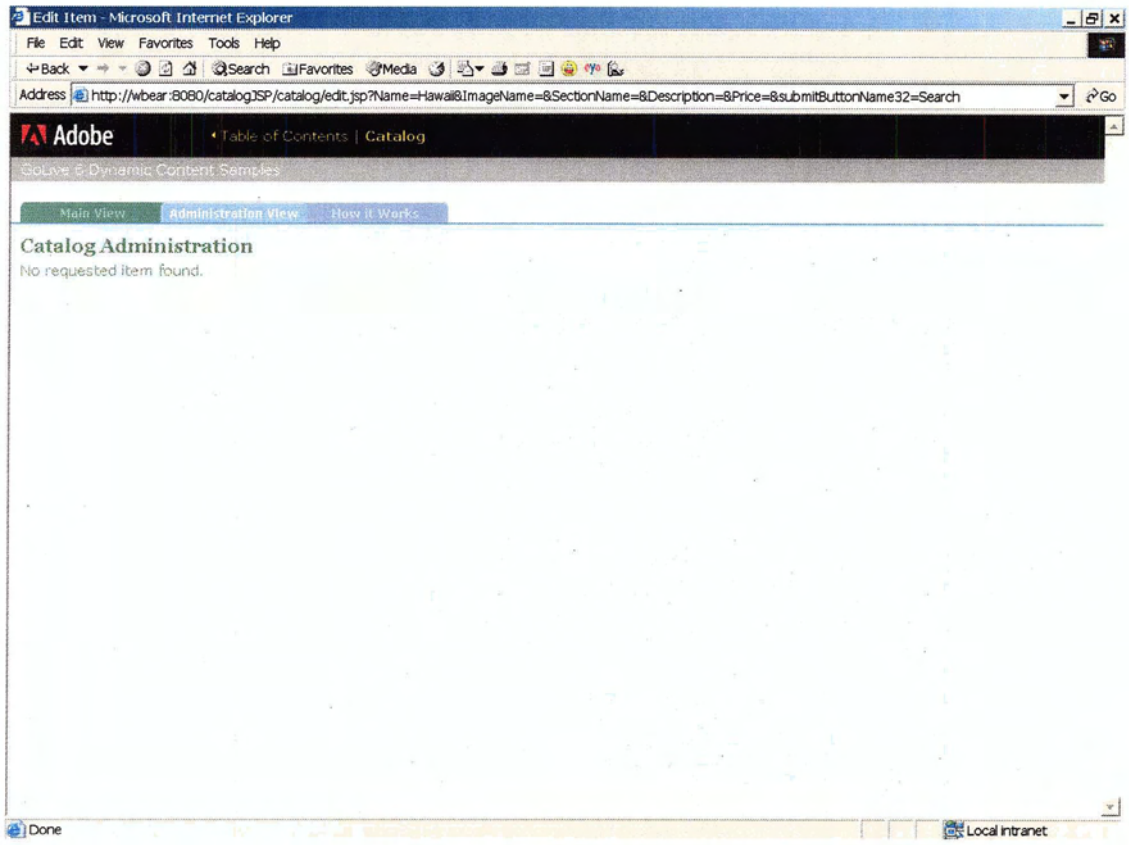

Figure 4.8 Screenshot of an Unsuccessful Query Result Page

142.104.107.79 - - [16/Nov/2004:21:50:08 -0800] "GET /catalogJSP/catalog/edit.jsp?Name=Hawaii&ImageName=&SectionName=&Description=&Price=&submitButtonName32=Search HTTP/1.1" 200 4079

142.104.107.79 - - [16/Nov/2004:21:59:24 -0800] "GET /catalogJSP/catalog/edit.jsp?Name=Yosemite&ImageName=&SectionName=&Description=&Price=&submitButtonName32=Search HTTP/1.1" 200 7207

Figure 4.9 Affected Entries in Server Log "catalina_access_log.2004-11-16.txt"

http://wbear:8080/catalogJSP/catalog/edit.jsp?Name=Hawaii&ImageName=&SectionName=&Description=&Price=&submitButtonName32=Search

http://wbear:8080/catalogJSP/catalog/edit.jsp?Name=Yosemite&ImageName=&SectionName=&Description=&Price=&submitButtonName32=Search

Figure 4.10 Resulting URLs from Figure 4.9

4.3.2 Data Abstraction

After collecting the raw data, it is necessary to analyze and reformat them to elicit useful information. In our case, data are categorized into entities, relationships and associated attributes. In general, entities are Web pages or components within pages. The relationships among them are navigation or containment. For instance, "database.html" links to "browse.jsp"; "table2" contains "image 3"; server program "calendar.jsp" called by "date.html" with parameter "date" and "time" in form "form1" submit.

For the developer view, entities also include GoLive generated components such as templates and smart objects, and corresponding usage relations (e.g. "edit.htm" uses template "topbar").

For the client view, entities also include copies of the server program generated pages (e.g., "find.jsp" links to server program "edit.jsp", which generates three different pages "edit1jsp.htm", "edit2jsp.htm" and "edit3jsp.htm", each of them representing a different computing behavior of the server program "edit.jsp"). A merge process is applied to remove redundant data since some generated pages may actually represent the same behavior of the server program. Dead links are detected through the site reference status (e.g. "browse.htm" links to dead node "edit.htm").

Figure 4.11 shows the type definition of nodes and arcs at the Web page level. For the inner page component, node types are HTML tags, arcs represent containment relationships. We use nodes and arcs to represent entities and relationships, respectively. Attributes of nodes include "id", "type", "name", "source file" and "x", "y" coordinates, etc; attributes of arcs include "id", "type", "from" and "to" indicating starting and ending nodes of the relation.

- Node Type
  - HTML document
  - Script file(JSP/ASP/PHP)
  - Template
  - Dynamic page
  - Dead node
  - Image file
  - Text document
  - Email
  - CSS file
- Arc Type
  - LinksTo
  - runsJSP/PHP/ASP
  - usesStyle
  - forwardsTo
  - emailsTo
- Triple definition
  - linksTo "/help/feature_index.htm" "/help/databases.htm"
  - Type "/images/spacer.gif" "image"

Figure 4.11 Type Definition of Nodes and Arcs

4.3.3 Data Structure

We modeled the Web application to describe its composite pages as well as navigation links and inner page components. Linked lists of Web page objects and their inner page components are implemented to store the abstracted site data for further processing.

The GoLive Site Reference object encapsulates a reference to a URL; it contains information about the referenced resource, such as its location, file transfer protocol, file type, size, and a list of its anchors. It also provides access to incoming Site Reference

objects that link to the page it represents, and outgoing Site Reference objects that it references. It can identify the type of the resource it represents as "html", "image", or "mail". To describe a Web page, we extended the Site Reference object to create the SiteRef object, which expands the definition of the "type" property to also cover page types such as dynamically generated page and dead link page (as specified in the domain definition shown in Figure 4.1), and added spatial information, id, and inner components such as templates, forms with parameters, and frames. It can also obtain a collection of its outgoing page objects by calling the getOutgoing() method. Figure 4.12 depicts the data structure of a Web page represented as "SiteRef" and its inner components. Similarly, the interaction among the components (e.g., "submit" of a "form" to a Web page) can be modeled using corresponding use case and sequence diagrams.

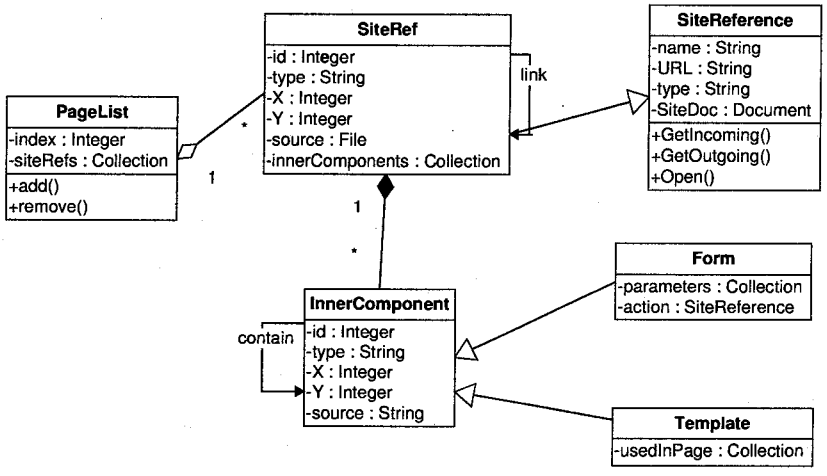

Figure 4.12 Data Structure of a Web Page and its Inner Components

A Web page represented by a SiteRef object can be a static or dynamic page which is indicated by the "type" attribute. While the static page contents are fixed, the contents of a dynamic page are computed at run time by the server. A Web server

program may behave differently according to the user interaction. Figure 4.13 shows an example of this feature of server program behavior. File "find.html" contains the initial static page connected to two replications of the sever program "edit.jsp", denoted as "edit1.jsp" and "edit2.jsp", respectively. Correspondingly, two different dynamic pages are built, "DYN1" and "DYN2". The "find.html" page accepts user input to search for a tour catalog. The input variable is passed to the server program with the value provided by the user through a form submit. The input of "Yosemite" results "edit1.jsp" triggered and "DYN1" page built representing a successful query result whereas the input of "Hawaii" results "edit2.jsp" triggered and "DYN2" page built representing a warning page of unsuccessful query.

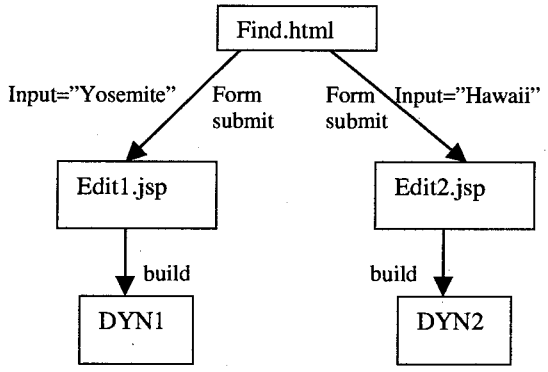

Figure 4.13 Example of Dynamic Page

The list of SiteRef objects is constructed starting from the home page represented by a SiteReference object, a corresponding SiteRef object is constructed and added to the list each time. The list constructor then iterates through all the SiteReference object's outgoing references by calling its getOutgoing() method and repeats this process. To

avoid cycles caused by multiple paths reaching the same URL, a Visited Page list is built and maintained—a Visited Page's outgoing reference will not be re-added to the Link list.

The list of inner components of a Web page is constructed starting from the "<html>" element, and then iterating all sub elements of the markup tree, which is kept in the element Collection object. This step is repeated until all elements have been visited. The position info (e.g., line number) is computed and added to the inner component data structure.

Data processing is performed with the constructed data structure. To gain a better overview of the Web site structure, we used tree layout to represent the site structure. The tree is constructed and spatial information of nodes and arcs are computed. Here nodes represent Web pages and arcs represent navigational relationships.

We use breadth-first traversal to build the tree. Starting from an initial SiteRef object, we added its outgoing SiteRef objects to a queue, and then retrieve an object from the queue in the FIFO sequence to repeat the processing until the queue is empty.

During this process, an array of counters is maintained to keep the number of nodes at each level of the tree, as well as the number of children of each node. This information is further used to compute the x and y coordinates of each node in the tree, and all corresponding SiteRef object are updated with the spatial information.

The tree of the inner page components is also constructed in the same manner. Here nodes represent components and arcs represent composition relationship.

4.3.4 Visualization

As discussed in Section 4.2.3, we can create an embedded graph in GoLive or generate an SVG graph for visual presentation and interactive exploration.

The GoLive Draw object provides methods that GoLive extensions can call to perform basic drawing operations. We obtain the Draw object from the drawControl() function, a global callback function called when a custom control should be redrawn. The functionality of the Draw object is to draw primitive graphics, text and images, as depicted in the source code of Figure 4.14. Here, "canvas" is the name of a custom control contained in a palette dialog. All drawing commands are created and stored in the "drawCmd" array. Nodes and arcs are drawn as ovals and lines, respectively.

```
function drawControl(control,draw)
{
      if (control.name == "canvas"){
            draw.setColor("white");
            draw.fillRect(10,10,780,580);//canvas background
      }
      drawGraph(draw);
}
function drawGraph(drawObj){
      var str, itemArray,name,shape,color,x1,y1,x2,y2;
      for(var i=0;i<indexCmd;i++){
            str=drawCmd[i];
            itemArray = str.split(":");
            name = itemArray[0];
            shape = itemArray[1];
            color = itemArray[2];
            x1 = itemArray[3];
            y1 = itemArray[4];
            if(shape=="line"){
                  x2 = itemArray[5];
                  y2 = itemArray[6];
                  drawObj.setColor(color);
                  drawObj.moveTo(x1,y1);
                  drawObj.lineTo(x2,y2);
            }
            else if(shape=="oval"){
                  drawObj.setColor(color);
                  drawObj.fillOval(x1,y1,18,18);
            }
      }
}
```

Figure 4.14 Sample Code Using GoLive Draw Object

We found that computing and rendering of a graph with a thousand nodes on the Palette Dialog is fairly slow especially when repainting the graph. Moreover, its lack of mouse control programmability over the drawn graph can not satisfy out visualization needs.

SVG is a logical middle ground between disparate domains [28]; it provides a presentation integration platform. The ACRE SVG engine [19], developed with SVG and ECMAScript, is a graph visualization engine for exploring and annotating software artifacts. It is embeddable into host applications such as Web browsers and office tools (e.g., PowerPoint, Word). Figure 4.15 is a screenshot of ACRE SVG Engine.

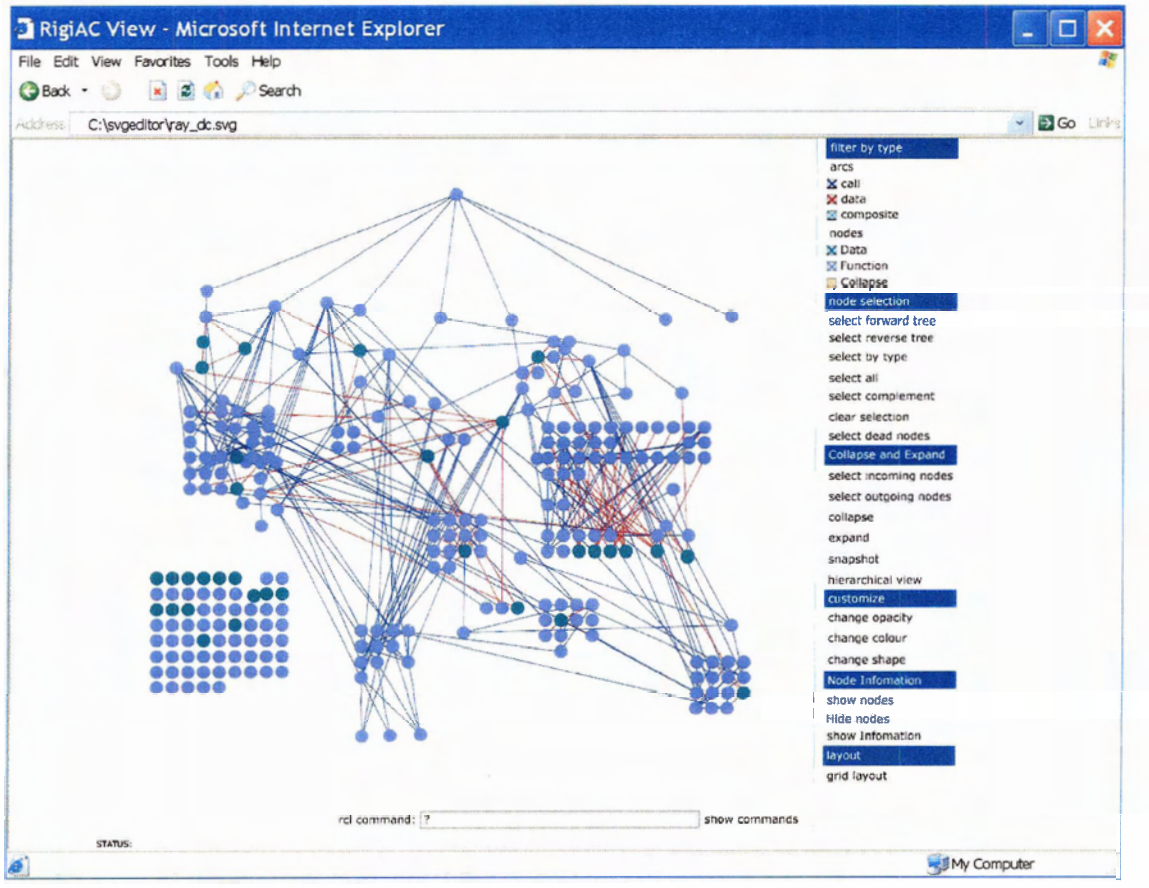

Figure 4.15 Screenshot of the ACRE SVG Engine

In order to gain better user control over the Web site structure graph, we export the node structure and the layout information into an SVG template file, which fits into the ACRE SVG engine. The domain definition of the ACRE SVG engine was expanded to include new node and arc types, node and arc attributes used in REGoLive, and corresponding ECMA Script functions were built to enable user interactions on these new nodes and arcs. Appendix B lists the sample source code of the ReGoLive visualization engine for generating SVG.

The problem with generating stand-alone SVG documents is losing the GoLive cognitive support if the presentation leaves the COTS product. GoLive users would likely prefer to stay within the GoLive development environment they are familiar with. Because of that, we need SVG to interact with GoLive to gain view and control integration back after generating the SVG view out of the GoLive environment. For security reasons, JavaScripts within SVG are restricted from accessing the local file system. This prevents us from sharing files between GoLive and SVG. Hence, we built a Web service where SVG can call a JSP program on a Web server, which reads/writes a message file, and ReGoLive opens a network socket to interact with the server. Appendix C lists the sample source code implementing this functionality.

We may also further customize the SVG engine; to improve presentation integration and thus to gain the cognitive support we lost by escaping to SVG back.

4.3.5 Demo of Prototype

To validate the feasibility and effectiveness of our approach, we developed the prototype—REGoLive. This section employs sample scenarios to illustrate how REGoLive helps in the analysis of a Web application.

This prototype is an extension built on top of the GoLive host environment. It takes the form of a "Main.html" file (as explained in Section 3.3.2) that resides in its own uniquely named folder called "RETool". To install REGoLive, one simply places "RETool" into the "GoLive/Modules/Extend Scripts" folder.

Web application slicing [20] is a decomposition technique that produces a reduced Web application, which behaves as the original one with respect to some information of interest; it helps in understanding the Web site's internal system structure. The Web site we use here to demonstrate REGoLive is a slice taken from the GoLive dynamic Content sample Web site. The original Web site consists of several samples with about 400 files where each sample exhibits similar structure and uses similar techniques. The sample we took is an online tour catalog containing about 200 files of Web pages and server programs. A user of this Web site can search for a tour package from the Americas or Europe categories through a JSP JDBC connection to the database server. We deployed the tour catalog Web site to a Tomcat 5.0 Web server and set up a MySQL 4.0 database server which stores the golive_samples database consisting of 11 tables. Other samples can be analyzed with REGoLive in the same manner by exploiting technologies such as ASP or PHP.

Figure 4.16 shows a screenshot of the GoLive development environment with the subject Web site open in the site window. The widgets circled in red are our custom REGoLive menu items.

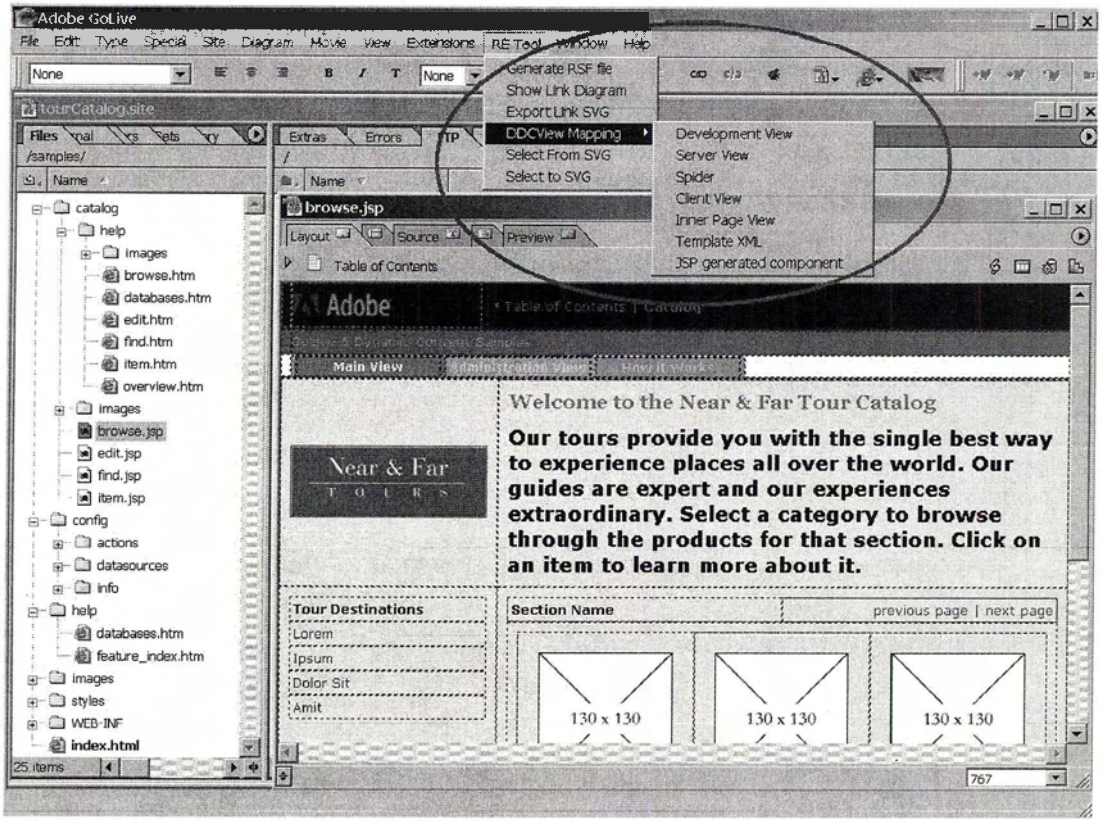

Figure 4.16 Screenshot of GoLive with ReGoLive Menu

To accomplish the Web application comprehension tasks specified in Section 4.1.2, we generated the server view, the developer view, and the client view of the subject system from GoLive and identified the differences between these views. The main differences stem from generated components (e.g., templates, smart objects) and server program generated dynamic codes.

4.3.5.1 Server View

Suppose a developer is tasked to upgrade a component on a Web application. This component resides in a JSP server program referenced by some Web pages of this site. He or she needs to know which Web pages call this server program and hence might be

affected if the server program changes. The developer then opens the Web site in GoLive with REGoLive loaded and selects the "Server view" submenu item from the drop down menu "RE Tool".

As shown in Figure 4.17, the generated SVG graph embedded in an Internet browser represents the server view. It helps the developer obtain a quick overview of the Web site's navigational structure and complexity. Blue nodes denote HTML pages and green nodes denote JSP server programs. Pages other than HTML and JSP are filtered to provide a simplified structural view.

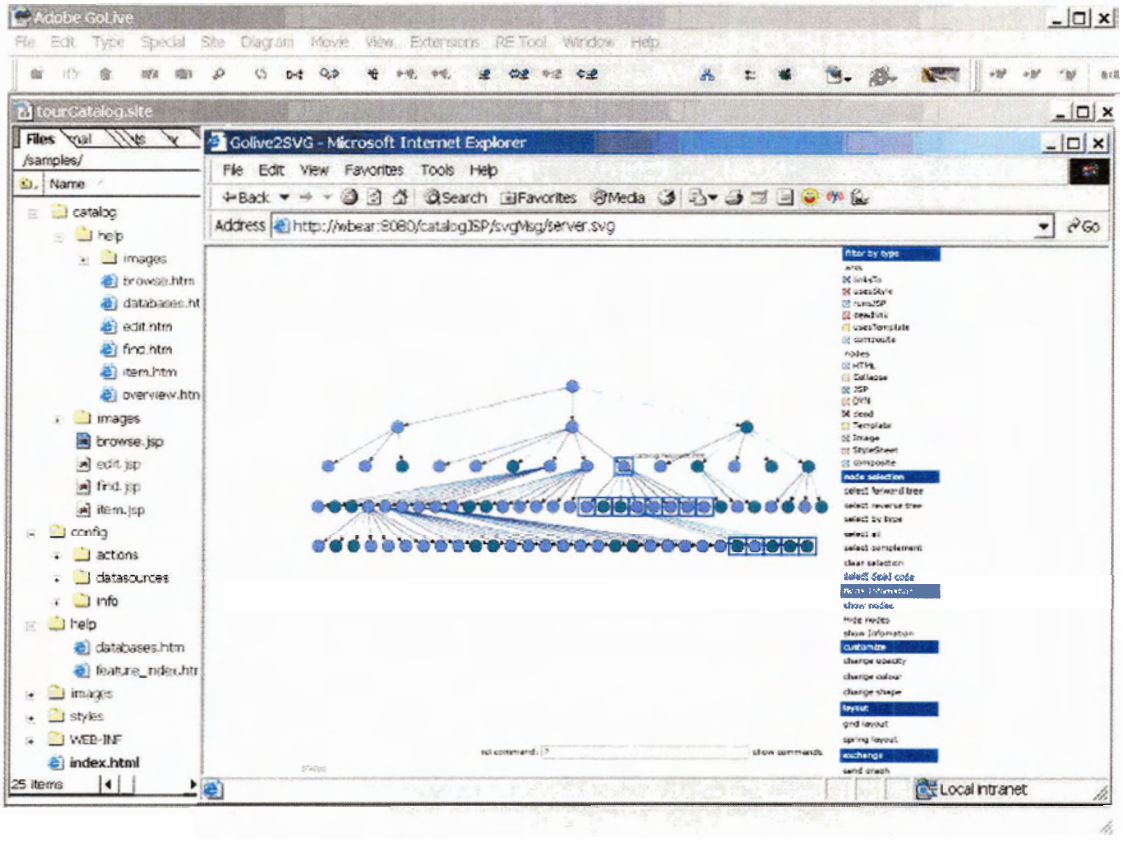

Figure 4.17 Server View

The developer can then further explore the graph, select the interested server program, highlight all nodes representing the server program and select the incoming

nodes representing the pages that link to the server program. He or she can also select an interested node in the generated SVG graph by clicking on "select to GoLive" in the popup menu, that will open the corresponding document in GoLive, and vice versa. The nodes' detailed information and their source files are accessible through the popup menu in the SVG graph.

4.3.5.2 Developer View

Suppose the development team decides to migrate a Web application to another platform—for example from Adobe GoLive to Macromedia Dreamweaver (e.g., due to the availability of a great new feature provided in Dreamweaver's new release). It is useful to estimate the costs and the risks of such a migration project. In particular, developers would like to know which components used in the site are GoLive specific and might thus are not recognized by Dreamweaver.

Template is one such example of a GoLive specific component. As depicted in Figure 4.18, the template nodes are colored in yellow in the generated developer view. From this graph, developers can learn quickly which pages exploit a template and thus need additional migration work. Other generated components can also be identified.

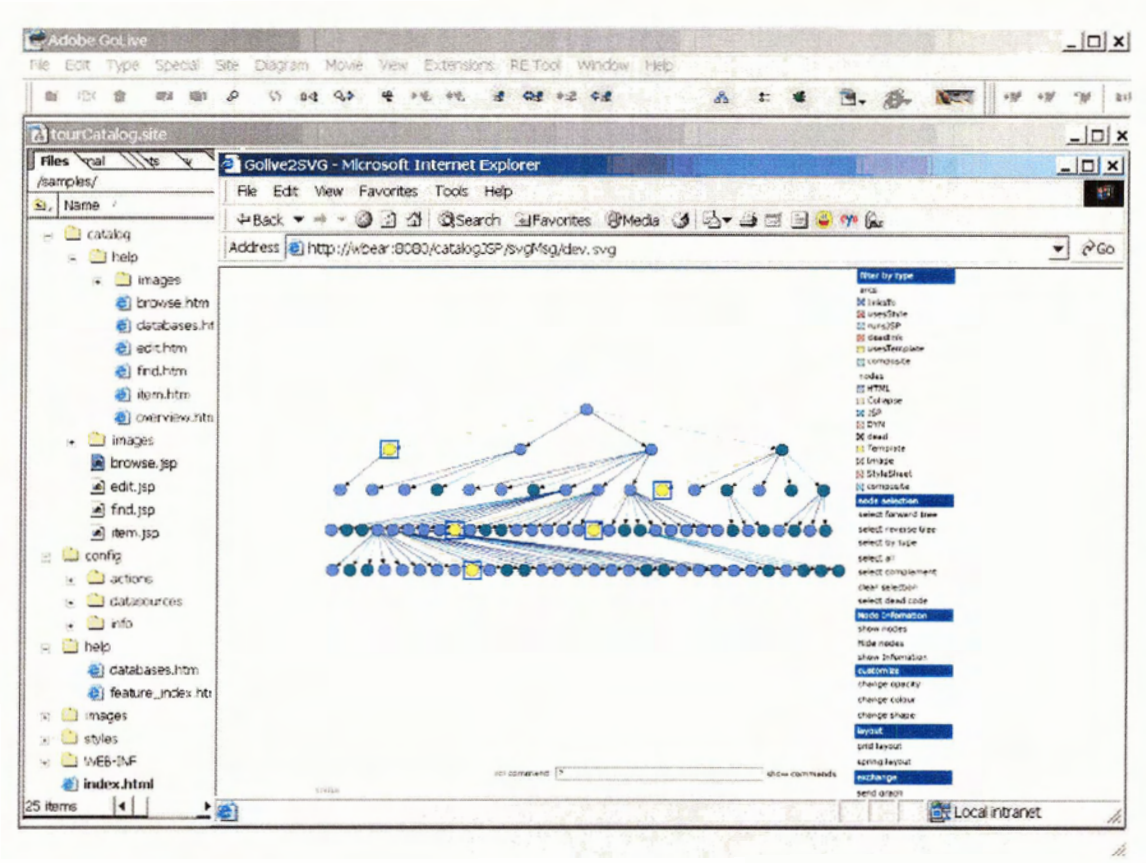

Figure 4.18 Developer View

Besides the generated SVG documents, we also generated an XML file to capture the template components information. Figure 4.19 exhibits the template usage identification in the generated XML file. The "<code>" portion contains the generated template source code from GoLive.

```
<?xml version="1.0" ?>
- <templates>
- <template file="../Templates/topbar.html">
- <referencedBy>
    <page id="0" lineNo="3" >/index.html</page>
    <page id="1" lineNo="8" >/catalog/browse.jsp</page>
    <page id="2" lineNo="3" >/catalog/help/browse.htm</page>
    <page id="3" lineNo="3" >/catalog/help/edit.htm</page>
    <page id="4" lineNo="5" >/catalog/help/resources.html</page>
  </referencedBy>
    -<code>
    +<![CDATA[ ]]>
```

```
    </code>
    </template>
  + <template file="../Templates/bottom_nav.html">
</templates>
```

Figure 4.19 Sample template identification in XML form

4.3.5.3 Client View

In the developer view, the developer only sees one copy of a server program (e.g., a JSP file), but a client can see different dynamic pages generated by that JSP from different computations based on client side user input or certain conditions.

Suppose the developer wants to change one of the functions of a JSP program "browse.jsp" without affecting its other functions.

At this stage, a client side copy of the Web site has already been obtained through dynamic analysis, as discussed in Section 4.3.1. This obtained client copy of the Web site is the subject system of the client view. With this client copy of the Web site opened in GoLive, the developer selects the "client view" menu item to generate the graph shown in Figure 4.20. Green nodes represent JSP generated HTML Web pages and red nodes represent duplicated JSP generated pages. Each red node represents a different computing path of the same JSP server program. Compared with the developer view in Figure 4.18, the Web site in the developer view only contains a single copy of the "browse.jsp" file under the "catalog" folder, whereas in the client side copy of the Web site in the client view, there are "browse1.jsp" and "browser2.jsp" denoted by red nodes, which represent two pages generated from the same "browse.jsp" server program through different execution path.

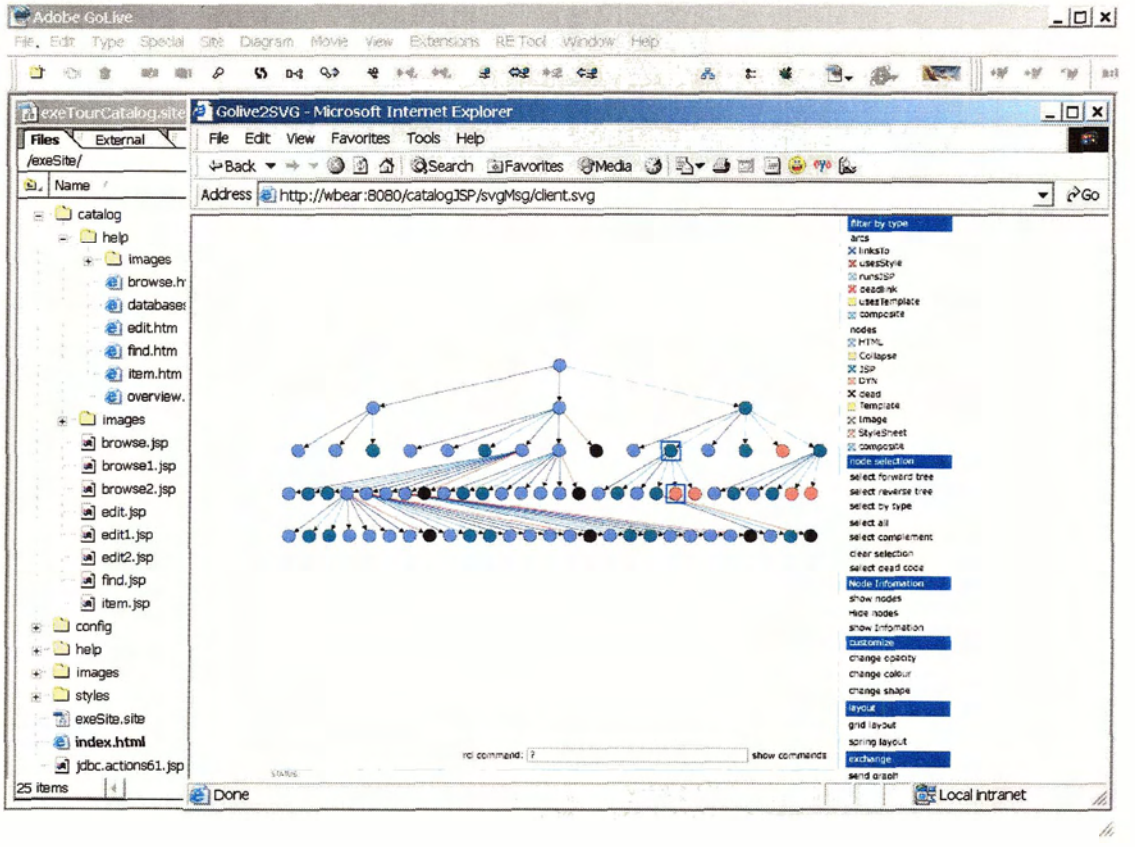

Figure 4.20 Client view

To show the differences between these two pages more explicitly, the developer can further exhibit the inner structure of these two pages, as depicted in Figures 4.21 and 4.22. In these diagrams each node represents a markup tag. As shown in Figure 4.21, the table represented by a yellow node circled in red contains no contents whereas the corresponding node in Figure 4.22 contains rows of images. The reason for this is that these pages are created on-the-fly and their contents is populated depending on and supplied by a server-side database. The first page is the result of an unsuccessful query to the database performed by the "browse.jsp" (as shown in Figure 4.8), and the second page is a successful query with results displayed (as shown in Figure 4.7).

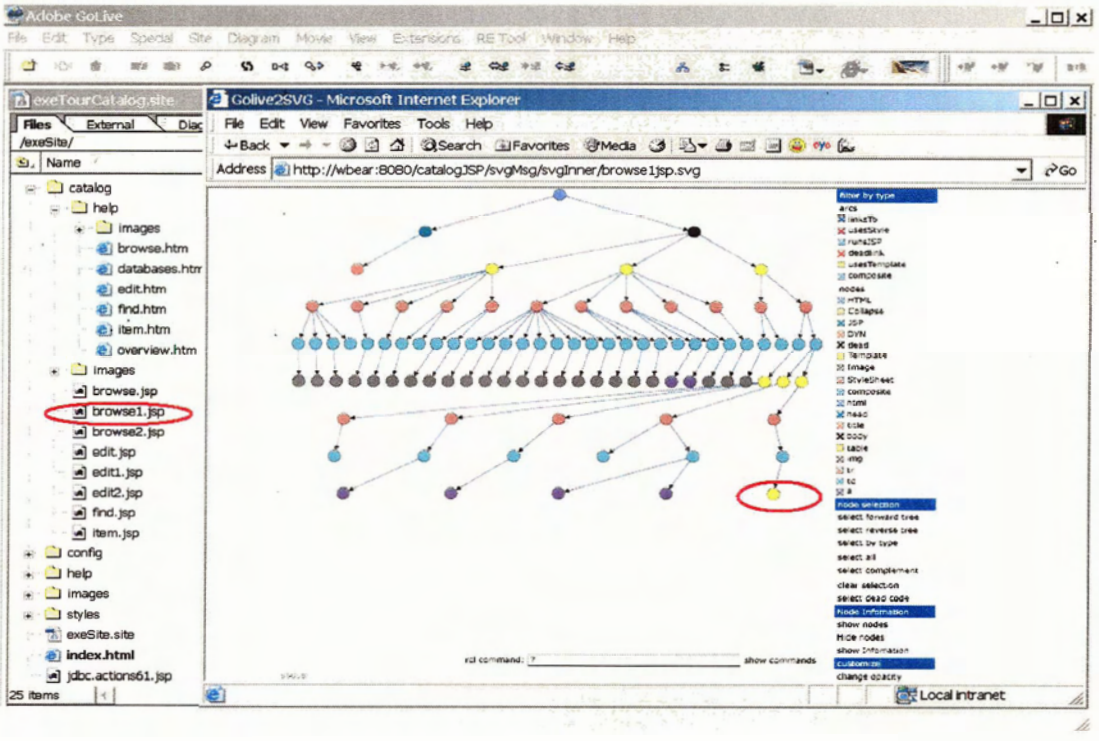

Figure 4.21 Generated Inner Page Structure for Page 1

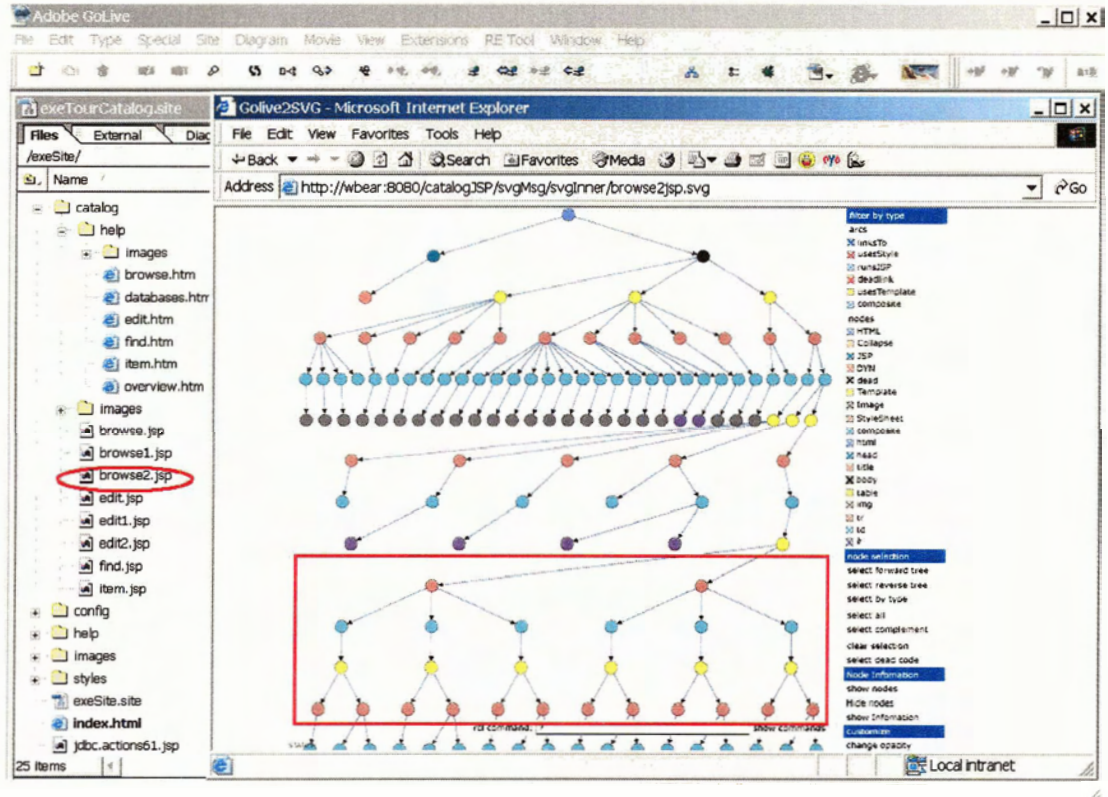

Figure 4.22 Generated Inner Page Structure for Page 2

GoLive provides mechanisms to identify broken links in the development view for the client view, however, since links might be accidentally removed from the Web server, we also need to detect broken links on the client side. The downloaded client side copy of the site is parsed and the missing nodes and links are denoted in black in the generated SVG graph depicted in Figure 4.20.

The details of JSP dynamic generated contents are also captured in XML files shown in Figure 4.23. All JSP calls are identified with attributes such as the referencing files, line number, a type of hyperlink or form submit, along with associated parameters passed to the JSP files. All JSP files within a site are parsed to obtain the generated output (i.e., value insertion expressions and print statements).

```
<?xml version="1.0" ?>
- <JSPs>
- <jsp file="cal1.jsp">
- <requestedBy>
    + <req id="0" from="cal1.jsp" lineNo="38" type="hyperlink">
    + <req id="1" from="cal1.jsp" lineNo="40" type="hyperlink">
    - <req id="2" from="cal2.jsp" lineNo="36" type="submit">
          <parameter id="0">date</parameter>
          <parameter id="1">time</parameter>
      </req>
    + <req id="3" from="login.html" lineNo="29" type="submit">
  </requestedBy>
-<generates>
  - <code id="0" lineNo="39" type="insert">
  - <![CDATA[
          table.getDate()
   ]]>
   </code>
  + <code id="1" lineNo="57" type="insert">

  -<code id="2" lineNo="61" type="print">
    - <![CDATA[
      <% out.print(util.HTMLFilter.filter(entr.getDescription())); %>
     ]]>
   </code>
  + <code id="3" lineNo="74" type="print">

  + <code id="4" lineNo="75" type="print">
```

```
    </generates>
    </jsp>
+<jsp file="cal2.jsp">

    </JSPs>
```

Figure 4.23 Sample JSP Code Generating Identification Output

4.4 Summary

This chapter introduced the design rationale and implementation details of our REGoLive prototype. Based on the supporting features of the host tool GoLive and the selected reverse engineering tasks we targeted, we designed the new tool's infrastructure, functionality and visual metaphor. The implementation section follows the flow of reverse engineering process, elaborates how artifacts are extracted and abstracted, what data structures are used to manipulate the acquired information, and what visualization techniques are exploited to present the results. At last, we demonstrated how to use the new tool under selected reverse engineering scenarios.

Chapter 5 Evaluation

The REGoLive tool has been built following the adoption-centric tool-building methodology outlined in Section 2.3. We address the specific benefits and drawbacks of this approach from both the tool-user's and tool-builder's perspective. Good practices and lessons learned are also documented.

5.1 Evaluating Using RE Tool Requirements

Following the tool requirements introduced in Section 3.1, a reverse engineering tool should be:

Scalable: We tested REGoLive with several sample Web applications provided with GoLive. One of these consists of about 400 files. Extraction and analysis of the sites do not cause performance problems, even though all files have to be programmatically opened in order to parse them. Generally, the added REGoLive functionality does not negatively impact GoLive's performance. The SVG visualization can handle graphs with hundreds of nodes adequately. Furthermore, the performance of the visualization engine can be further tailored by displaying graphs at various levels of abstraction. Potentially, REGoLive is scalable to explore larger information space.

Extensible: GoLive itself can be extensively customized via JavaScript to integrate new functionality seamlessly into the existing environment. Since the extensions are available in source text form in GoLive's plug-in folder, developers can adapt REGoLive to their own needs. However, customizations based on scripting languages are challenging to manage and maintain from a software engineering point of view.

Exploratory: Views in GoLive are well suited for interactive exploration. Extensions for program comprehension can take advantage of this and add their own information to existing or new views. This approach benefits tool adoption since the users are already familiar with the concept of GoLive's views and can smoothly switch between the built-in and the custom views.

Interoperable: Since GoLive's API allows programmatic file manipulation, data interoperability via reading and writing of files can be easily achieved. Using RSF, XML and SVG as data exchange facilitates, information sharing between REGoLive components and other tools can readily be accomplished.

We extended REGoLive to output SVG files, which can then visualized in a Web browser. For security reasons, SVG's JavaScript code is prohibited to access the local file system. After some experimentation, we found that GoLive can be used to access Web services by launching a URL on a server, which enables messaging between REGoLive and SVG to achieve control integration. For example, selecting a graph node in SVG sends a message to GoLive to select the corresponding entities in the views within GoLive environment, and vice versa.

Customizing GoLive's menu bar constitutes presentation integration. However, our SVG visualization does not achieve presentation integration, since the graph is not visualized using GoLive's native drawing capabilities. Performance problems and limitations of GoLive's API are both responsible for this break down in presentation integration, which, from an adoption perspective, is a significant drawback. For example, how the user interacts with the SVG visualization engine is quite different from how a user interacts with the GoLive's user interface. Our research group, currently, is

investigating how to design and implement a framework that generates SVG GUI widgets tailored to the specific look and feel of a host environment such as GoLive.

Language-independent: REGoLive's analysis capabilities are both language and host-tool dependent. However, REGoLive can take advantage of GoLive's existing parsers, which support a broad range of popular Web technologies. If necessary, customizations can use GoLive's Translator objects to gain access to files before they are passed on to GoLive's parsers.

Adoption-friendly: REGoLive is easily installed by copying its Main.html to GoLive's module-extension directory, it can be easily uninstalled similarly. REGoLive creates its own pull-down menu in GoLive and is thus conveniently accessible. Users of REGoLive can inspect the JavaScript source and make changes to better suit their needs; they can also easily redistribute these changes to other users (see Appendix D for more details). Third-party contributions such as REGoLive can be submitted to the Adobe GoLive Developer Knowledgebase and Adobe Studio Exchange [52].

5.2 Quality Comparison

Compared with other WSRE research tools described in Section 2.2, REGoLive realizes limited reverse engineering functionality. However, as a proof-of-concept prototype, we were able to generate different views (developer, server, and client) of Web sites and map between these perspectives to aid Web site comprehension. We further comparing REGoLive to stand-alone research tools with respect to non-functional requirements and found that REGoLive has advantages as follows:

Easy of Use: REGoLive has more potential users compared to stand-alone research tools since the host tool already has a large user base; moreover, it can be easily

installed/uninstalled by simply copying/removing the extension folder; the integrated pull-down menu in the host tool environment also provides easy access to the new tool's functionality.

Development effort: The development effort to extend a COTS product is significantly less than building stand-alone tool. It took one developer several months of development time to build REGoLive. By leveraging the existing features of the host tool (e.g., parsing), we only need to concentrate on realizing the additional reverse engineering functionality to be provided. The JavaScript implementation of REGoLive is realized with about 2,500 LOC (Lines of Code). The SVG visualization engine that we employed consists of about 7,000 lines of JavaScript code. Although learning GoLive functionalities and supportive features takes time and effort, the experiences and knowledge we gained can be reused when working on similar platforms such as DreamWeaver.

Interoperability: REGoLive support data interchange with other reverse engineering tools via RSF, XML and SVG formats. The generated output can easily integrate into other SVG-compatible tools. Control integration between GoLive and the SVG visualization engine is achieved by means of a Web service which serves as the communication carrier. Thus the generated output can easily controlled by Web service-enabled tools.

5.3 Experience and Lessons Learned

From the developer's point of view, building on top of a component that offers many functionalities with potential for reuse has greatly reduced our development effort and facilitated rapid prototyping.

GoLive facilitates rapid prototyping with a built-in JavaScript editor and debugger. New or changed JavaScript code can be immediately executed and tested. While the amount of code to be written is greatly reduced, time has to be spent up front understanding GoLive's architecture, customization mechanisms, and API. GoLive's API is quite large and described in about 600 pages of documentation [24]. Developers can deepen their understanding by taking advantage of discussion forums (i.e., GoLive SDK User to User Forums) and sample extensions (i.e., Adobe Studio Exchange).

Furthermore, before customizing GoLive it is also necessary to first master its functionality in order to better understand its existing (program comprehension) capabilities and suitable places for customizations. GoLive is a sizeable and sophisticated application; the developer book that we used has about 900 pages to describe all of its functionality [51]. It took the author of this thesis about four months to familiarize herself with GoLive's functionality and customization capabilities, but the development time of REGoLive was significantly shortened as a result.

Fortunately, GoLive's SDK is stable and we did not encounter any bugs. However, one future problem we envision is difficulties due to different SDK versions. GoLive has evolved rapidly with each new version, each introducing significant API changes. Not all of these changes are backward compatible. Thus, switching to a newer version of GoLive will likely break the code of REGoLive.

Limitations exist when using scripts to access lower-level resources such as network sockets. An external library built in C/C++ is required in this case.

5.4 Summary

Given our experience designing, implementing and using REGoLive, we are pleased to attest that the tool is able to satisfy most of the functional and non-functional requirements identified at the beginning of Chapter 3. It is scalable, extensible, exploratory, and adoption-friendly to a certain degree. The new tool has many advantages over a stand-alone research tool with respect to leveraging cognitive support, interoperability, adoptability and, of course, all the advantages stemming from being an industrial-strength tool. Data and control interoperability have been achieved; however, the SVG graph visualization needs further work regarding presentation integration with the GoLive user interface. Other students in our group are working on a general presentation integration framework for the SVG engine with several COTS product platforms.

Chapter 6 Conclusions

6.1 Summary

Web based systems tend to evolve frequently due to new technological and commercial opportunities, as well as feedback from users. As a consequence, Web site reverse engineering tools are emerging to facilitate Web site comprehension and maintenance. Traditional software research tools often have difficulty being adopted and deployed in industry because of their unfamiliarity with users and poor usability.

This thesis presents a reverse engineering tool REGoLive, based on the ACRE approach, aiming at increasing tool adoptability by implementing reverse engineering functionality on top of a Web authoring tool with a large user base. By leveraging the supportive features of the host tool, development time and effort is saved. The new tool is more likely to be evaluated and adopted by potential users.

6.2 Contributions

In this thesis, we have introduced a tool-building approach that strives to make reverse engineering tools more adoptable by building new reverse engineering functionality on top of existing, popular components. To select a proper host tool, we analyzed and compared the reverse engineering capabilities of some popular web authoring tools including FrontPage, DreamWeaver, and GoLive.

In order to show the feasibility of this approach, we have developed REGoLive, which we built on top of the Adobe GoLive Web authoring tool. REGoLive augments GoLive to support Web site comprehension. Specifically, REGoLive offers a graph

visualization of a Web site's three viewpoints (i.e., developer, server, and client view), exposing dependencies between them. Stand-alone Web site comprehension tools can only address the client and server view, whereas REGoLive supports GoLive's developer view. To the best of our knowledge, no other tool or analysis explicitly identifies the three viewpoints or offers mappings between them. We believe that making these mappings explicit potentially benefits Web site comprehension greatly.

Through building REGoLive, we also obtained experience on extending host tools. We investigated the supportive features of REGoLive in detail, which might be helpful for other developers who wish to build extensions on GoLive or similar Web authoring tool such as Dreamweaver.

When implementing REGoLive, we were able to leverage GoLive's existing capabilities (e.g., for parsing), but customizing GoLive for sophisticated graph visualization could not be realized. This is an example of a typical trade-off when using (black-box) components. If the component does not support the implementation of a certain requirement, then the requirement has to be dropped, adapted, or implemented outside of the component.

In order to assess REGoLive's effectiveness as a reverse engineering tool, we collected tool requirements identified by researchers in the area and then assessed REGoLive with them. The results of this assessment suggest that REGoLive can meet most requirements and can compete with stand-alone research tools. Thus, REGoLive serves as a case study to show the feasibility of component-based building of reverse engineering tools.

Furthermore, we believe that REGoLive is more adoptable with users that are already familiar with GoLive, compared to implementing a stand-alone tool. Since REGoLive is integrated within GoLive, Web developers and maintainers can smoothly transition between GoLive's basic program-comprehension capabilities and REGoLive's advanced views. However, these hypotheses need further evaluation, for example, with user studies. However, REGoLive constitutes a useful case study platform for future ACRE research.

6.3 Future work

A user study is needed in order to further evaluate the tool. The ideal participants are GoLive users with similar skills. One group of users would be assigned a set of Web site comprehension tasks using only the original GoLive, while another group would perform the same tasks with the help of REGoLive. We could then observe, collect and compare the feedback from each group and find out how REGoLive facilitates Web site reverse engineering.

To achieve better interoperability, we may customize the SVG engine to regenerate the user interface with the same look-and-feel as the host tool, such as GoLive, to improve presentation integration.

To achieve better control integration, we may connect GoLive to the ACRE engine to improve interoperability, by importing and exporting data, consuming and providing operations and services.

There is still plenty of opportunity for improving the functionality of REGoLive. We may detect the defects within a Web application where the violation of software

engineering principles occurs; we may analyze the Web site evolution aspects leveraging the version control feature of GoLive; we may conduct Web site usage analysis by generating the access log using scripts; we may build client side JavaScript parsers and Server side parsers to further analyze the control flow and data flow of a Web application. We may also implement an advanced repository with a relational database to query Web site metrics.

Bibliography

[1] D. Lucca, A. Fasolino, F. Pace, P. Tramontana and U. Carlini, "WARE: A tool for the reverse engineering of Web applications," *Proceedings the 6th European Conference on Software Maintenance and Reengineering (CSMR 2002)*, pp. 241-250, Budapest, Hungary, March 2002.

[2] F. Ricca and P. Tonella, "Web site analysis: structure and evolution," *Proceedings International Conference on Software Maintenance (ICSM 2000)*, pp. 76-86, San Jose, USA, October 2000.

[3] F. Estiévenart, A. François, J. Henrard and J. Hainaut, "A tool-supported method to extract data and schema from Web sites," *Proceedings 5th IEEE International Workshop on Web Site Evolution (WSE 2003)*, pp. 3-11, Amsterdam, The Netherlands, September 2003.

[4] S. Chung and Y.S. Lee, "Reverse software engineering with UML for Web site maintenance," *Proceedings 1st International Conference on Web Information Systems Engineering (WISE 2000)*, pp. 157-161, Hong Kong, June 2000.

[5] D. Lucca, A. Fasolino and U. Carlini, "Recovering Class Diagrams from Data-Intensive Legacy Systems," *Proceedings IEEE International Conference on Software Maintenance (ICSM 2000),* pp. 52- 63, San Jose, USA, October 2000.

[6] F. Ricca and P. Tonella, "Understanding and Restructuring Web Sites with ReWeb," *IEEE Multimedia,* Vol.8, pp. 40-51, 2001.

[7] H. Kienle, A. Weber, J. Martin and H. A. Müller, "Development and maintenance of a Web site for a Bachelor program," *Proceedings 5th IEEE International Workshop on Web Site Evolution (WSE 2003),* pp. 20-29, Amsterdam, The Netherlands, September 2003.

[8] P. Tonella and F. Ricca, "Dynamic model extraction and statistical analysis of Web applications," *Proceedings 4th IEEE International Workshop on Web Site Evolution (WSE 2002)*, pp. 43-52, Montréal, Canada, October 2002

[9] Securityspace.com, http://www.securityspace.com/s_survey/data/man.200403/Webauth.html, April 2004.

[10] S. Tilley and S. Huang, "Evaluating the reverse engineering capabilities of Web tools for understanding site content and structure: a case study," *Proceedings 23rd IEEE/ACM International Conference on Software Engineering (ICSE 2001)*, pp. 514-523, Toronto, Canada, May 2001.

[11] P. Warren, C. Gaskell and C. Boldyreff, "Preparing the ground for Web site metrics research," *Proceedings 3rd IEEE International Workshop on Web Site Evolution (WSE 2001)*, pp. 78-85, Florence, Italy, November 2001.

[12] E. Mendes, N. Mosley and S. Counsell, "Web metrics—estimating design and authoring effort," *IEEE Multimedia*, Vol. 8 , pp. 50-57, 2001.

[13] D. Fetterly, M. Manasse, M. Najork and J. Wierner, "A Large-Scale Study of the Evolution of Web Pages," *Proceedings 12th International World Wide Web Conference (WWW 2003)*, pp. 669-678, Budapest, Hungary, May 2003.

[14] P. Tonella, F. Ricca, E. Pianta and C. Girardi, "Using keyword extraction for Web site clustering," *Proceedings 5th IEEE International Workshop on Web Site Evolution (WSE 2003)*, pp. 41-48, Amsterdam, The Netherlands, September 2003.

[15] H. A. Müller, A. Weber and K. Wong, "Leveraging cognitive support and modern platforms for adoption-centric reverse engineering," *Proceedings 3rd International Workshop on Adoption-Centric Software Engineering (ACSE 2003)*, pp. 30-35, Portland, Oregon, USA, May 2003.

[16] K. Wong, "The Reverse Engineering Notebook," PhD Thesis, University of Victoria, Victoria, British Columbia, Canada, 1999.

[17] S. R. Tilley, "Domain-Retargetable Reverse Engineering," PhD Thesis, University of Victoria, Victoria, British Columbia, Canada, 1995.

[18] J. Martin and L. Martin, "Web Site Maintenance with Software-Engineering Tools," *Proceedings 3rd IEEE International Workshop on Web Site Evolution (WSE 2001)*, pp. 126-131, Florence, Italy, November 2001.

[19] H. A. Müller, A. Weber, and H. Kienle, "Leveraging Cognitive Support in SVG Applications From The Host COTS Product," *SVG Open*, Vancouver, Canada, 2003.

[20] F. Ricca and P. Tonella, "Web Application Slicing," *Proceedings IEEE International Conference on Software Maintenance (ICSM 2001)*, pp. 148-157, Florence, Italy, November 2001.

[21] A. E. Hassan and R. C. Holt. Architecture Recovery of Web Applications. *Proceedings IEEE International Conference on Software Engineering (ICSE 2002)*, Orlando, Florida, 19-25 May 2002.

[22] A. E. Hassan and R. C. Holt, "Towards a Better Understanding of Web Applications," *Proceedings 3rd IEEE International Workshop on Web Site Evolution (WSE 2001)*, pp. 112-116, Florence, Italy, November 2001

[23] Internetworldstats.com, http://www.internetworldstats.com/stats.htm, November 2004.

[24] Adobe, GoLive 6.0 Extend Script SDK Programmer's Reference Manual.

[25] A. E. Hassan and R. C. Holt, "A Visual Architectural Approach to Maintaining Web Applications," *Annals of Software Engineering*, Vol. 16, 2003.

[26] F. Ricca, "Analysis, Testing and Re-structuring of Web Applications," PhD Dissertation, DISI, University di Genova, Genova, Italy, 2003.

[27] http://www.w3.org/TR/SVG12/ November 2004.

[28] http://www.w3.org/Graphics/SVG/ December 2004.

[29] J. Offutt, "Quality Attributes of Web Software Applications," IEEE Software, Vol. 19, pp. 25-32, March-April 2002.

[30] H.A Müller and K. Klashinsky, "Rigi—A System for Programming-in-the-large," *Proceedings the10th International Conference on Software Engineering (ICSE 1988)*, pp. 80-86, Raffles City, Singapore, April 1988.

[31] M.-A. Storey, C. Best, J. Michaud, D. Rayside, M. Litoiu and M. Musen, "SHriMP views: an interactive environment for information visualization and navigation," *Proceedings Conference Extended Abstracts on Human Factors in Computer Systems (CHI 2002)*, pp. 520-521, Minneapolis, Minnesota, USA, April 2002.

[32] N. Mansurov and D. Campara, "*Extracting High-level Architecture from Existing Code With Summary Models*," http://www.klocwork.com/products/insight.asp, December 2004.

[33] R.S. Pressman, "What a tangled Web we weave," *IEEE Software*, Vol. 17, pp. 18-21 , January-February 2000.

[34] R. Balzer, J.-H. Jahnke, M. Litoiu, H.A. Müller, D.B. Smith, M.-A. Storey, S.R. Tilley and K.Wong, *Proceedings 3rd International Workshop on Adoption-Centric Software Engineering (ACSE 2003), Workshop at 25th IEEE/ACM International Conference on Software Engineering (ICSE 2003)*, pp. 789-790, Portland, OR, May 2003.

[35] A. Ginige and S. Murugesan, "Web engineering: An introduction," *IEEE Multimidia*, Vol. 8, pp. 14 – 18, April-June 2001.

[36] D. Fetterly, M. Manasse, M. Najork, and J. Wiener, "A large-scale study of the evolution of Web pages," *Software-Practice and Experience*, Vol. 34, pp. 213-237, 2004.

[37] E. J. Chikofsky and J. H. II. Cross, "Reverse engineering and design recovery: A taxonomy," *IEEE Software*, Vol. 7, pp. 13-17, 1990.

[38] J. Conallen, *Building Web Applications with UML,* Object Technology, Addison-Wesley Longman, Reading, Massachusetts, USA, first edition, December 1999.

[39] S. Murugesan, Y. Deshpande, S. Hansen and A.Ginige, "Web Engineering: A New Discipline for Web-Based System Development," *Proceedings 1st ICSE Workshop on Web Engineering (ICSE 1999)*, Los Angeles, June 1999.

[40] Y. Deshpande and S. Hansen, "Web Engineering: Creating a Discipline among Disciplines," *IEEE Multimedia*, Vol. 8, pp. 82- 87, April-June 2001.

[41] M. Lehman, D. Perry and J. Ramil, "Implications of Evolution Metrics on Software Maintenance," *Proceedings International Conference on Software Maintenance* (ICSM 1998), pp. 208-217, 1998.

[42] S. K. Card, J. Mackinlay and B. Shneiderman, "Readings in Information Visualization Using Vision to Think," San Francisco, CA, Morgan Kaufmann, 1999.

[43] G. Lucca, A. Fasolino and P. Tramontana, "Towards a better Comprehensibility of Web Applications: Lessons learned from Reverse Engineering Experiments," *Proceedings 4th IEEE International Workshop on Web Site Evolution (WSE 2002),* pp. 33-42, Oct. 2002

[44] D. Jackson and M. Rinard, "Software analysis: A roadmap," *Proceedings 22nd International Conference on The Future of Software Engineering*, pp. 135-145, June 2000.

[45] S. Ducasse, M. Lanza and S. Tichelaar, "Moose: an extensible language-independent environment for reengineering object-oriented systems," *Proceedings 2nd International Symposium on Constructing Software Engineering Tools (COSET 2000)*, June 2000.

[46] S. Tilley, "Domain-Retargetable Reverse Engineering," PhD thesis, Department of Computer Science, University of Victoria, 1995.

[47] A. Alvaro, D.Lucredio and V. Garcia, "Component-based software reengineering environment," *Proceedings 10th IEEE Working Conference on Reverse Engineering (WCRE 2003)*, pp. 248-259, November 2003.

[48] D. Jin, J. R. Cordy and T. R. Dean, "Transparent reverse engineering tool integration using a conceptual transaction adapter," *Proceedings 7th European Conference on Software Maintenance and Reengineering (CSMR 2003)*, pp. 399-408, March 2003.

[49] I. Thomas and B. A. Nejmeh, "Definitions of tool integration for environments," *IEEE Software*, Vol. 9, pp.29-35, March 1992.

[50] E. M. Rogers, "Diffusion of Innovations," The Free Press, fourth edition, 1995.

[51] J. Carlson and G. Fleishman, "Real World Adobe GoLive 6," Peachpit Press, 2003.

[52] Adobe Studio Exchange http://share.studio.adobe.com/, March 2005.

[53] M. Story and H. Müller, "Rigi: A Visualization Environment for Reverse Engineering," *Proceedings International Conference on Software Engineering (ICSE-97)*, pp. 606-607, Boston, Massachusetts, May 1997

Appendix A Source Code for Dynamic Page Downloading

```
function extractDynURL(siteRefObj,childSiteRef,count)
{
        var link;
        var filename = childSiteRef.name;
        var now = new Date();
        var day   = now.getDate();
        var month = now.getMonth();
        month=month+1; //starts from 0
        var year  = now.getYear();
        if(year < 2000) { year = year + 1900; }        //Y2K
        if(month<10)      month="0"+month;
        if(day<10)        day="0"+day;
        var date = year +"-"+month+"-"+day;

        var srcURL=siteRefObj.url.substring(rootIndex-1);
        writeln("launch " + WebSvrURL+srcURL);
        app.launchURL(WebSvrURL+srcURL);

        logfile =new JSXFile("d:\\tomcat5.0\\logs\\catalina_access_log."+date+".txt");
        if(!logfile.exists){
                writeln("file does not exist");
        }

        var o=logfile.open("w");
        if(!o){
                writeln("failed to open file");
        }
        var c=logfile.close();
        var dstURL=  childSiteRef.url.substring(rootIndex-1);
        alert("please access "+ dstURL +" then press ok");
        var o=logfile.open("r");
        if(!o){
                writeln("failed to open file");
        }
        while(!logfile.eof){
                link=logfile.readln();
                var index = link.indexOf(childSiteRef.name);
                if(index != -1){
                        break;
                }
        }
        var c=logfile.close();
        var index1=link.indexOf("GET");
        var index2=link.indexOf("HTTP/1.1");
        var index3=link.indexOf("?");
        var dynURL= link.substring(index1+4,index2-1);
        var dlURL=link.substring(index1+4,index3);

        var splitAry= dlURL.split("\/");
        dlURL="";
        for(var j=2; j<splitAry.length; j++){
                dlURL+= "\/"+splitAry[j];
```

```
        }

        splitAry= dynURL.split("\/");
        dynURL="";
        for(var j=2; j<splitAry.length; j++){
                dynURL+= "\/"+splitAry[j];
        }
        var filename2=dlURL.substring(1);
        var filename3=filename2.replace(/\//g, '\\');
        var ind=filename3.indexOf(filename);
        var foldername = rootDir + filename3.substring(0,ind);
        var subfolder=filename3.substring(0,ind-1);
        var idx2= subfolder.indexOf("\\");
        var curfolder=rootDir;
        var remain=subfolder;
        if(idx2 == -1){
                exePage = JSXFile(foldername);
                exePage.createFolder();
        }
        else{
                var folderAry=subfolder.split("\\");
                for(var i=0; i< folderAry.length; i++){
                        curfolder=curfolder + folderAry[i] +"\\";
                        exePage = JSXFile(curfolder);
                        exePage.createFolder();
                }
        }
        var idx3=filename3.indexOf(filename);
        filename3=filename3.substring(0,idx3);
        var splAry=filename.split(".");
        filename=splAry[0];
        for(var j=1;j<splAry.length-1;j++){
                filename+= "."+splAry[j];
        }
        filename += count+ "." + splAry[splAry.length-1];
        filename3 += filename;
        exePage = JSXFile(rootDir + filename3);
        exePage.get(WebSvrURL + dynURL);
        writeln("file:"+rootDir + filename3);
        writeln("url:"+WebSvrURL + dynURL);
}
```

Appendix B Source Code for Generating SVG

```
function generateSVG(page)
{
        var filename="dev.svg";
        if(view == "server")
                filename="server.svg";
        var svgTmpl=JSXFile("c:\\redoc\\svg\\svgTmpl");
        svgfile=JSXFile("c:\\redoc\\svg\\"+ filename);
        svgfile.remove();
        svgTmpl.copy("c:\\redoc\\svg\\"+ filename);
        svgTmpl.close();
        svgfile=JSXFile("c:\\redoc\\svg\\"+ filename);
        if(svgfile!=null){
                var o=svgfile.open("a");
                if(!o){
                        writeln("failed to open file");
                }
                if(!svgfile.exists){
                        writeln("file does not exist");
                }
        }
        else{
                writeln("rsf fileobj is null");
        }
                initialize();
                if(page)  isPage = true;
                prepareGraph();

        //feed into node & arc info
        for(var i=0;i<nodeAry.length;i++){
                var id=nodeAry[i].id;
                var type=nodeAry[i].type;
                var name=nodeAry[i].name;
                var srcfile=nodeAry[i].srcfile;

                svgfile.writeln("node"+id+" = model.createNode(\"" + id + "\",\"" + type +"\");" );
                svgfile.writeln("Node.setAttribute(node"+id+",\"name\", \""+name+"\");");
                svgfile.writeln("Node.setAttribute(node"+id+",\"sourcefile\", \""+srcfile+"\");");
        }
        for(var i=0;i<arcAry.length;i++){
                var id=arcAry[i].id;
                var type=arcAry[i].type;
                var from=arcAry[i].from;
                var to=arcAry[i].to;
                svgfile.writeln("arc"+id+"= model.createArc(\""+id+"\", \""+type+"\",
node"+from+",node"+to+");");
        }
        svgfile.writeln("graph.setBounds(1, 1, 1000, 768);");
        svgfile.writeln("var view = graph.view;");
        svgfile.writeln("view.viewattrs.setAttr(\"NODE_RADIUS\", 9);");
        svgfile.writeln("view.viewattrs.setAttr(\"GRAPH_STYLE\", \"directed\");");
        svgfile.writeln("view.viewattrs.setAttr(\"LABEL_BEHAVIOUR\", \"tooltip\");");
        svgfile.writeln("view.realize();");
```

```
        svgfile.writeln("view.setViewbox(90, -51, 1030, 614, \"xMinYMin meet\");");
        svgfile.writeln("view.importModel();");
        svgfile.writeln("var nodeview = view.nodeview;");
        for(var i=0;i<nodeAry.length;i++){
                        var id=nodeAry[i].id;
                        var x=nodeAry[i].xPos;
                        var y=nodeAry[i].yPos;
                        svgfile.writeln("nodeview.setInitialLocation(\""+id+"\", "+x +", "+y+");" );
        }
        svgfile.writeln("view.redraw();");
        svgfile.writeln("var labels = view.labels;");
        svgfile.writeln("labels.createLabels(view.nodes)");

        for(var i=1;i<=nodeAry.length;i++){
                svgfile.writeln("labels.setDefaultVisibility([\""+ i +"\"], false);" );
        }

        var svgTmpl2=JSXFile("c:\\redoc\\svg\\svgTmpl2");
        var r=svgTmpl2.open("r");
        var str=svgTmpl2.read();
        svgfile.write(str);
        var c=svgfile.close();
        if(!c){
                writeln("failed to close file");
        }
        var c=svgTmpl2.close();
        finalize();
        app.launchURL("c:\\redoc\\svg\\"+ filename);
}
```

Appendix C Source Code for Communication between GoLive and SVG

```
// JavaScript in REGoLive

function select2SVG()
{
        var selectedfiles=Website.getSelectedFiles();
        var siteref=selectedfiles.first();

        var urlhome=Website.homePage.url;
        var namehome=Website.homePage.name;
        var rootindex=urlhome.indexOf(namehome);

        var nodeName="";

        while(siteref!=null){
                writeln(siteref.url.substring(rootindex-1));
                nodeName=nodeName+siteref.url.substring(rootindex-1) + ";";
                siteref=selectedfiles.next();
        }
        app.launchURL("http://wbear:8080/catalogJSP/wrMsg.jsp?nodeName="+nodeName);
}

function selectFromSVG()
{
                var msgfile = new JSXFile("C:\\redoc\\msg.txt");
                var i;
                var newmsg;
                var msg=" ";
                msgfile.get("http://wbear:8080/catalogJSP/msg.txt");
                msgfile.open("r");
                newmsg = msgfile.readln();
                msgfile.close();
                Website.deselectAllFiles();
                Website.selectFiles(newmsg);
                var selectedfiles=Website.getSelectedFiles();
                var siteref=selectedfiles.first();
                var document=siteref.open();
}



// "wrMsg.jsp" on Web server
<HTML>
<BODY>
<%@ page import="java.io.*" %>
<% String msg = request.getParameter("nodeName");%>
<% System.out.println("jsp received msg: "+ msg);%>
<%
        try {
            BufferedWriter bw = new BufferedWriter(new FileWriter("./Webapps/catalogJSP/msg.txt"));
            bw.write(msg);
```

```
            bw.close();
        } catch (IOException e) {
            System.out.println(e.getMessage());
        }
%>
</BODY>
</HTML>

//ECMAScript added to ACRE SVG Engine
////////////////////////////////////
//      select2GoLiveCommand
////////////////////////////////////
inherit(Command, select2GoLiveCommand);
select2GoLiveCommand.prototype.desc = "select corresponding nodes in GoLive";
function select2GoLiveCommand (graph) {
  if (arguments.length > 0) {
      this.init(graph);
  }
}
select2GoLiveCommand.prototype.init = function (graph)
{
  this.graph = graph;
};
select2GoLiveCommand.prototype.execute = function ()
{
      var nodeselection = this.graph.nodeselection;
        if (nodeselection.size() <= 0 || nodeselection.size() > 1) {
            throw new GeneralError('You must select ONE node');
        } else {

            select2GoLiveCommand.superclass.execute.call(this);

            var v = nodeselection.toArray()[0].gnode;
            var nodeName=v.attribute["name"];
            getURL('../wrMsg.jsp?nodeName='+nodeName, null);
      }
};
////////////////////////////////////
//      selectFGoLiveCommand
////////////////////////////////////
inherit(Command, selectFGoLiveCommand);
selectFGoLiveCommand.prototype.desc = "select corresponding nodes in GoLive";
function selectFGoLiveCommand (graph) {
  if (arguments.length > 0) {
      this.init(graph);
  }
}
selectFGoLiveCommand.prototype.init = function (graph)
{
  this.graph = graph;
};
selectFGoLiveCommand.prototype.execute = function (obj)
{
      getURL('../msg.txt', select);
};
```

Appendix D REGoLive Installation Guide

1. Download and install Adobe GoLive 6.0 SDK
2. Create a sub folder named "RE Tool" under the "Modules/Extend Scripts" folder within the GoLive application folder
3. Copy "Main.html" of REGoLive under "RE Tool" folder created in the previous step
4. Copy the "svgMsg" folder (containing SVG scripts) of ReGoLive to a Web server with JSP engine.
5. Download and install Adobe SVG viewer 3.0
6. Restart GoLive application and REGoLive will be loaded, you will be able to see the drop down menu "RE Tool" on the menu bar