Transforming Legacy Web Applications to the MVC Architecture

Yu Ping

Dept. of Electrical &
Computer Engineering
University of Waterloo
Waterloo, ON. Canada
yping@swen.uwaterloo.ca

Kostas Kontogiannis

Dept. of Electronics &
Computer Engineering
Technical University of Crete
731 00 Chania, Greece
kkontog@softnet.tuc.gr

Terence C. Lau

IBM®
Centre for Advanced Studies
IBM Canada Laboratory
Toronto, ON. Canada
lautc@ca.ibm.com

Abstract

With the rapid changes that occur in the area of Web technologies, the porting and adaptation of existing Web applications into new platforms that take advantage of modern technologies has become an issue of increasing importance. This paper presents a reengineering framework whose target system is an architecture based on the Model-View-Controller (MVC) design pattern and enabled for the Java™ 2 Platform, Enterprise Edition (J2EE). The proposed framework is mainly concerned with the decomposition of a legacy Web application by identifying software components to be transformed into Java objects such as JavaBeans, JavaServer Pages (JSP), and Java Servlet.

1. Introduction

Just a few years ago, Web applications were mainly composed of a few simple and static HTML pages, which were used to share the information over the Internet. Today, however, many organizations manage their business activities by using Web-based systems, whether in business-to-business (B2B) or business-to-consumer (B2C) contexts. Traditionally, these so-called legacy Web systems were developed by utilizing obsolete architectures and technologies. In most cases, legacy Web applications cannot be simply discarded because they often encapsulate a great deal of business knowledge accumulated over the years, and constitute a significant investment for the organization that operates them. For this reason, the need to migrate existing legacy Web systems towards modern Web technologies has become an important objective.

Modern Web systems refer to the software systems that are designed by multi-tier architectures and supported by open standards and new technologies. Particularly, J2EE is now one of the most widely accepted and prevalent technologies for design, development, and deployment of Web-enabled applications in enterprise environments [6].

In this paper, we present a transformation framework that aims at adopting existing Web systems into enterprise Java environments based on the MVC design pattern. Specifically, the database access functionality is extracted from source programs, and then encapsulated in JavaBeans objects. Consequently, the JSP pages translated from the legacy presentation components can reference these newly created JavaBeans objects. Moreover, we develop a transformation process for refactoring a JSP-based Web site to a controller-centric architecture by extracting the linkage information from the JSP application analysis. As a result, the target migrant system would be more scalable, portable, and maintainable than the original one [1,16,28,29].

The remainder of this paper is organized as follows. Section 2 provides pointers to the related work. Section 3 outlines the migration framework, including migration methodology and architectures, while experimental results and case studies are discussed in Section 4 and Section 5. Finally, Section 6 offers conclusions and directions for the future research.

2. Related Work and Discussion

Several works on migration of legacy applications into new Web-based platforms have been proposed in the literature.

Aversano et. al. [18] present a tool to migrate COBOL systems into Web applications based on the methodologies described in MORPH project [19]. The tool decomposes the original system into its user interface and server (application and database). The user interface has been migrated into a Web browser using Active Server Pages® and VBScript®, while the server is wrapped

This work was funded by IBM Canada Ltd. through IBM Centre for Advanced Studies and by the Natural Sciences and Engineering Research Council of Canada.

by dynamic load libraries written in Microfocus Object COBOL.

Bodhuin et. al. [25] present a strategy to incrementally migrate the COBOL program decompose into new architecture based on Model-View-Controller (MVC) design pattern. The system decomposition process includes static analysis, restructuring, and slicing techniques. A toolkit [2] is developed to translate the view into JSP pages, and the model and controller are wrapped into the Web environments.

The main difference with our work is that the source program in [2,18,25] is not a Web-based system where the user interface has to be re-implemented from character-based screen to the HTML-oriented Web page. On the contrary, our source system is the existing Web application. Thus, the target system can reuse the same Web page design as the original one, as well as most of display elements from the source system, such as HTML and JavaScript without changing them. Another difference is that the database and business logic is kept untouched by applying wrapping technique in [2,18,25], whereas we encapsulate database access components (SQL statements) into JavaBeans objects by conforming to the data bean architecture.

Ricca et. al. [8] sketch several possible Web application transforms with the aim of improving their qualities. They have classified HTML transformations into six categories: syntactic clean up, page restructuring, style renovation and grouping, improving accessibility, update to new standards, and design restructuring. A case study has been provided based on transforming original navigation structure into HTML frames by utilizing the DMS® Software Reengineering Toolkit™, an integrated tools infrastructure for automating customized source program analysis and modification of large scale software systems [22].

In another paper [7], Ricca and Tonella propose a migration process aimed at restructuring static Web sites into dynamic ones using the software clustering technique, where a common template is extracted from the HTML pages in the same cluster, and the variable information is isolated from the template and then moved into a database. They introduce a re-engineering tool, so called *ReWeb*, which is able to perform source code analysis and graph representation on Web sites.

These works in [7,8] focus on the analysis and the representation of static Web sites in which the Web pages are not generated by the server-side scripting language. By contrast, our approach includes the analysis of dynamic and behavioral aspects of Web applications. We share with [7] the idea of using the clustering technique to restructure the Web sites in our proposed transformation framework. However, we aim to refactore the Web sites by adopting a controller-centric architecture instead of the frame-based one described in [7].

The work most related to ours is [12]. In this work, Cordy et. al. describe the Whole Website Understanding Project (WWSUP), a long-term project with design-level understanding of Web applications by analyzing their source codes. One subproject is to transform Perl to Java platforms, which consists of three major tasks: Perl to JSP/Java, Perl modules to EJB, and translation of any modules imported by the Perl modules. However, there are very few documentations or publications available to discuss this project in detail.

There have been also some considerable activities and prototype tools on the migration of IBM Net.Data [9,10] systems to enterprise Java platforms. Several tools have been developed by using Java technologies to support the migration works, especially concerning the SQL statement transformation [13,14,21,23,24,26].

Most of the existing IBM Net.Data migration tools focus on the database components transformation and ignore the presentation logic conversion. Moreover, they try to generate the JavaBeans objects based on analyzing or parsing the SQL statement itself. They only accept the static or prepared dynamic SQL statements, of which the column names are explicitly specified, but not the non-prepared dynamic SQL statements or any statements of which the column names are implicitly specified [28]. With this restriction, there is no way to extract the output parameters (column names) from such SQL statements, and thus the tools will fail to generate the JavaBeans objects due to insufficient information.

Finally, there has been extensive research work conducted on programming language translation, such as C to RPG [3] or PL/IX to C++ [17], Pascal to C, and Cobol to OO-Cobol to name a few. Compared with those works that translate monolithic systems, our source system being translated is much more complex. Web applications involve several intertwined languages dealing with different aspects of the system. This makes the parsing process difficult. In addition, our target system is represented in several languages stacked one on top of the other. It requires not only the literal translation of the underlying languages, but also the stratification of the functionalities represented in legacy Web applications.

3. Architectural Patterns for Java Enterprise Applications

In addition to the translation of the source code from one language to another, the software migration process should also allow for improving the quality characteristics of the migrant system. For this work we focus on design principles for the migrant system that relate to improvements on the portability and customizability of the new application as these are supported by the open Java standards and technologies that are introduced as part of the reengineering process. By applying these open architectures, the target migrant system would be more

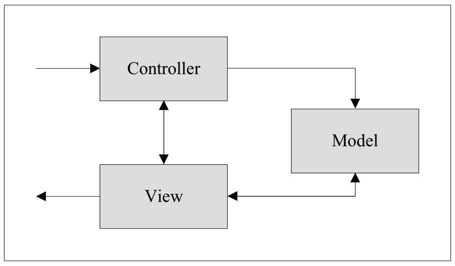

Figure 1: Model-View-Controller Architecture

scalable and maintainable than the original system. The following subsections present in detail the issues pertaining to the merits of such open architectures.

3.1 Model View Controller (MVC)

The MVC [11] design pattern is a widely used architectural pattern in J2EE applications. It separates the data persistence, user interface, and application control. To this end, an application is decoupled into three core components: the model, the view, and the controller (Figure 1).

The **model** contains the core functionality of application components, such as database access and transaction management. It encapsulates the state of the applications and conducts associated transformation on that state. Typically, the model has no specific knowledge of either the view or the controller.

The **view** provides the presentation of the state represented by the model. It manages the visual display of the applications. Particularly, there is no processing logic within the view; it is simply responsible for retrieving objects (in the model) that may have been previously created by the controller. The view should be notified when the state changes in the model. In addition, it has no knowledge about the controller.

The **controller** is in charge of user interaction with the model. It manages the request processing and the creation of any objects (in the model) used by the view. Moreover, it forwards the user request to the view depending on the user's actions.

In our proposed reengineering approach, the new system will adopt the MVC architecture in an enterprise Java environment. As a result, the source application is refactored into JavaBeans (Model), JavaServer Pages (View), and Java Servlet (Controller).

3.2 Data Bean Compliant Architecture

Data beans are JavaBeans objects that are mainly used to provide a logical collection of data in JSP pages. They are not a set of well-defined Java classes, but rather, an architectural construct.

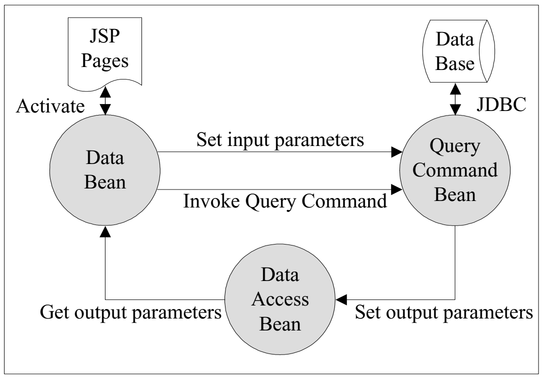

Figure 2: Data Beans Architecture

In the proposed migration framework, we define a data bean architecture that consists of three types of JavaBeans objects: query command bean, data access bean, and data bean (Figure 2). The transformation program generates these three Java objects for each SQL statement extracted from the source application.

The **query command bean** is the actual execution of the SQL statement extracted from the source program. It is an implementation of JDBC™. It also performs a certain set of actions, such as setting input parameters to the SQL statement, constructing a complete SQL statement, and mapping the result set data returned from the SQL statement to a data access bean (setting output parameters).

The **data access bean** is a serializable Java object, which mainly contains a set of getter and setter methods. It carries a set of data from a query command bean to a data bean.

The **data bean** is an object wrapper to encapsulate the invocation of a query command bean in a JSP page. It also performs a certain set of actions, such as setting the input parameters to the query command bean, and populating the data access bean (getting the output parameters) so that JSP pages can then use the data bean to display the data that it contains.

By utilizing the data bean architecture, the target JSP pages eventually remove the SQL statements from the display logic. Additionally, using data bean wrapper significantly reduces the Java codes included in the JSP pages, since the Java implementation of the data access is encapsulated inside the data beans.

3.3 Controller-centric Architecture

The controller-centric architecture, as illustrated in Figure 3, is an architectural approach based on the Model-View-Controller (MVC) design pattern. In this approach, a controller is built on the top of a Web system to perform the central management for the client request processing and Web page forwarding. More specifically, the front-

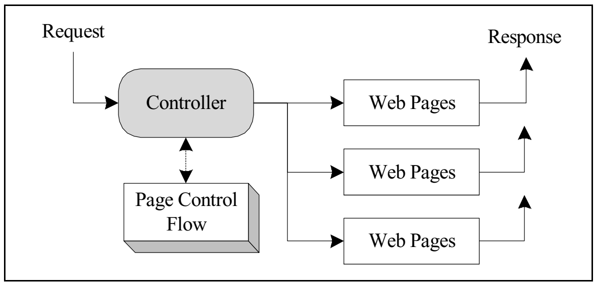

Figure 3: Controller-centric Architecture

end controller provides a single entry point for intercepting HTTP requests coming from end users. It takes control of the page flow that is originally managed by individual Web pages. In other words, each Web page is not directly linked to another page. Instead, it is connected to its associated controller, and the controller then forwards the request to the target page. The page control flow is often stored either in a flat file or a database, which can be accessed by the controller in order to select corresponding Web pages to forward the request to [29].

To this end, the page management becomes a simpler and easier task. For example, when Page A is changed to Page B, all the links to Page A are broken until these links are updated. On a traditional Web site, as illustrated in Part 1 of Figure 4, link references are maintained by individual Web pages, where we need to search all the pages (Page X to Z) that have link references to Page A and updating these links to Page B. However, on a controller-centric Web site, as shown in Part 2 of Figure 4, we do not need to find and modify all the pages (Page X to Z) that have link references to the old page (Page A). Instead, we only need to update the page control flow information maintained by the controller, and change the corresponding link reference to the new page (Page B).

Comparing to Part 1, Part 2 significantly simplifies the page modification process as well as eliminate the broken link or the missing link, practically in a large scale Web site. In addition to providing the central control for the page flow, adapting Web sites to the controller-centric architecture allows us to remove potential duplication codes from Web pages. This can be achieved by including common services, such as page access authentication, to the controller.

4. Transformation Framework

The proposed framework is twofold. First, it establishes a stage-wise methodology concerning the implementation of the migration works. Second, it defines the architecture of the transformation program according to the transformation methodology. In this section, we describe the framework in detail.

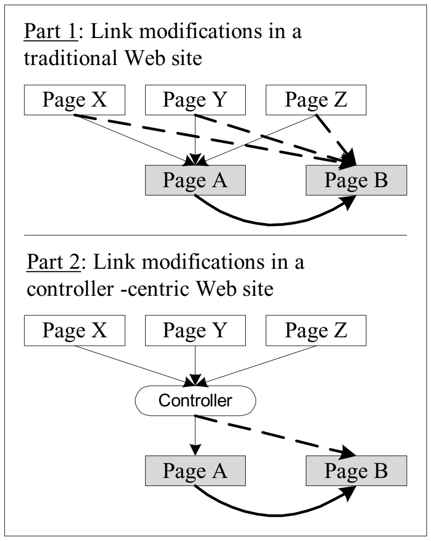
Part 1: Link modifications in a traditional Web site
Page X
Page Y
Page Z
Page A
Page B
Part 2: Link modifications in a controller -centric Web site
Page X
Page Y
Page Z
Controller
Page A
Page B

Figure 4: Page Management

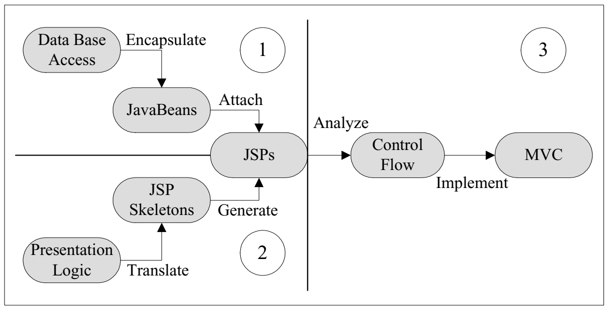

Figure 5: Transformation Methodology

4.1 Transformation Methodology

To ensure that the transformation process is organized in a systematic way, a methodology for migration of Web legacy applications to enterprise Java environments is formulated, as depicted in Figure 5. The activities described in the transformation methodology are performed in a sequential manner and the whole process is divided into three implementation phases.

Phase 1. Separating database access (model) from presentation logic (view):
The display components in a Web legacy application commonly include SQL statements to implement database access. Therefore, the first phase focuses on the extraction and analysis of SQL statements in the legacy program. In this respect SQL statement functionality is encapsulated

into the JavaBeans objects by conforming to the data-bean-compliant architecture, and thus to isolate the model from the view.

Phase 2. Making the transition to JSP pages:
A Web legacy application commonly uses proprietary constructs to customize display pages. JSP pages are used by a J2EE application to handle the presentation logic. Hence, in the second phase, the display elements of the legacy program need to be replaced by JSP pages with a combination of JavaBeans objects generated from Phase 1.

Phase 3. Adopting a controller-centric architecture:
The objective is for the migrant system to be adapted to modern and customizable architectural patterns such as the MVC. In this context, the third phase focuses on the analysis of application control flow, which is presented as a collection of HTML files, generated JSP pages, and JavaBeans objects. Control flow can be extracted from the migrant system in the first two phases, and eventually provide a roadmap for the generation of controller components.

4.2 Transformation Architecture

The transformation methodology outlined in the previous section consists of three subprocesses, database access encapsulation (Phase 1), presentation logic conversion (Phase 2), and JSP pages refactoring (Phase 3). Each subprocess is supported by a common transformation model that is composed of five different layers, which are based on reengineering activities. As Figure 6 shows, the proposed model includes the source system layer, the software analysis layer, the information extraction layer, the new code generation layer, and the target system layer, from the top to the lowest level, respectively.

The **source system** layer, from which each transformation process starts, represents the software applications that are going to be reengineered. The **software analysis** layer focuses on representing the source of the system being analyzed at an appropriate level of abstraction, such as in the form of an Abstract Syntax Tree (AST). The **information extraction** layer aims at the extraction of associated facts by a series of analysis steps applied to the source code representation generated at the **software analysis** layer. The **new code generation** layer supports the automatic generation of the new applications. The **target system** layer, where each transformation process ends, represents the newly created platforms based on the migration approach. This layered design approach has resulted in the overall transformation

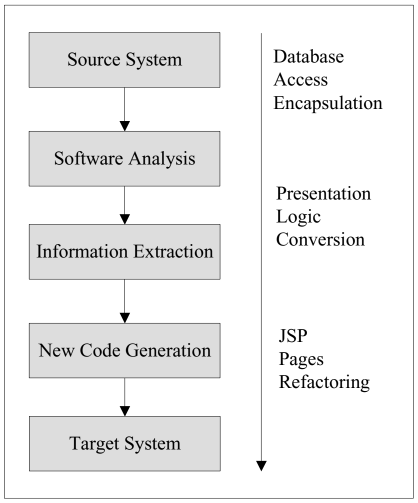

Figure 6: Transformation Layer

architectures shown in Figure 7 and discussed in the following sections.

4.2.1 Database Access Encapsulation and Presentation Logic Conversion Architectures

The database access encapsulation and the presentation logic conversion focus on the different aspects of the migration process. Both transformations, however, start the migration tasks by selecting the same legacy Web application at the source system layer, and use the same tool to analyze the source system at the software analysis layer. Then they can be performed independently after the generation of the source code representation, and independently of the choice of the target language. Therefore, we combine the database access encapsulation and presentation logic conversion together into a single architecture in order to simplify the actual implementation of the transformation works (shown in the left part of Figure 7).

In the software analysis layer, the **Language Analyzer** reads the source codes provided by the source system layer and recognizes its structure according to the source language grammar. It consists of two subsystems: a lexical analyzer that breaks the input string into tokens, and a language parser that discovers the hierarchical structure of the program.

The output of the analyzer is an Abstract Syntax Tree (AST), which stores the intermediate representation of the input program [27]. The **Language Analyzer** also provides an Application Program Interface (API) for the AST. The **SQL Information Extractor** and the **Display Information Extractor** use this API to access the information from the AST.

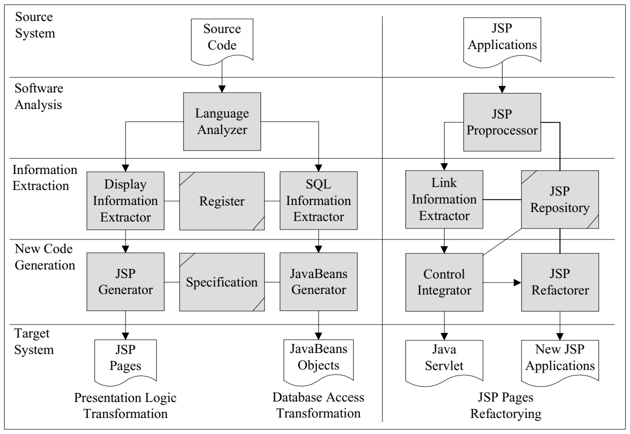

Figure 7: Transformation Architecture

There are two extraction components in the information extraction layer: the **SQL Information Extractor**, which is part of the database access encapsulation process, and the **Display Information Extractor**, which is part of the presentation logic conversion process. The **SQL Information Extractor** traverses the AST generated from the source program in the software analysis layer, and collects the essential information, such as SQL statements, input parameters, and output parameters. As a result, it produces an SQL properties file that stores the SQL statements, and a parameter container file that contains the input and output parameters, as well as other SQL information in an XML format. The **Display Information Extractor** extracts and analyzes the display elements represented in the AST of the source program, and sends the results to the **JSP Generator** for the conversion and construction of complete JSP pages. Both extractors are independent processes, but they can share the information by storing their intermediate results in the **Register**.

The new code generation layer contains two generators, the **JavaBeans Generator** included in the database access encapsulation process, and the **JSP Generator** included in the presentation logic conversion process. The **JavaBeans Generator** is a tool that can automatically generate the JavaBeans objects (run-time Java code) for the actual execution of the SQL statement. It takes the SQL properties file and the parameter container file provided by the **SQL Information Extractor** as input, and consequently creates three JavaBeans objects according to the data-bean-compliant architecture: data bean, data access bean and query command bean. The **JSP Generator** converts display elements provided by the **Display Information Extractor** to corresponding JSP elements. In addition, the **JSP Generator** composes the new JSP pages with other generated components, such as included JSP pages and JavaBeans objects. Both generators make the translation process according to the predefined conversion **Specification**.

4.2.2 JSP Pages Refactoring Architecture

The database access encapsulation and presentation logic conversion phases aim to separate the SQL statements from the display pages in the legacy Web application and consequently generate the model and view components. As the third phase of our proposed transformation methodology, the JSP pages refactoring phase focuses on the reconstruction of JSP pages in order to support the generation of controller components. As a result, the target system will eventually adopt the MVC architecture in an enterprise Java environment.

A Web site is a hyperlinked network environment, which consists of hundreds of interconnected pages, usually without a well engineered architecture [2,15]. Therefore, in addition to supporting the migration of legacy Web applications to new platforms, the JSP pages refactoring process can also serve as a generic approach to restructuring an existing JSP-based Web site to a controller-centric architecture.

The right part of Figure 7 depicts the JSP pages refactoring architecture by conforming to our proposed five-layer transformation model. The JSP applications contained in the source system layer represent a collection of HTML pages, JSP pages, and JavaBeans objects, which are either generated from the first two phases, the database access encapsulation and presentation logic conversion, or provided by an existing JSP-based Web site. At the software analysis layer, the **JSP Preprocessor** analyzes the source of the JSP applications and identifies all the information useful to the subsequent layers, such as control statements and hyperlinks. The **JSP Preprocessor** is designed as a lightweight JSP analyzer, instead of a heavyweight parser-based one. Next, the **Link Information Extractor** at the information extraction layer starts to collect link information included in the JSP and HTML pages by utilizing the pattern-matching technique. The results of the link extraction can be represented as an XML document. The **Control Integrator** in the new code generation layer then clusters the JSP and HTML pages according to the identified link dependencies between them, and consequently generates Java Servlet objects to implement the created page control flow. Finally, the **JSP Refactorer** in the same layer modifies the link information included in the current JSP and HTML pages by conforming to the page control flow, and thus the resulting JSP application adopts the controller-centric architecture. Additionally, each component stores its intermediate results and shares the information in the **JSP Repository** over the entire refactoring process.

5. Case Study

To demonstrate the effectiveness of the framework advocated in this paper, we have developed a set of reengineering tools written in Java programming language, which implement all of three major transformation processes presented in our proposed migration architecture. We also had them tested for migrating two Web systems, which are IBM WebSphere Commerce applications, and a JSP-based Web site. The detailed case studies and experimental results are presented in this section.

5.1 IBM Net.Data to JSP Transformation

IBM WebSphere Commerce (formerly IBM Net.Commerce®) is a platform for building e-commerce Web sites and applications. Since the release of IBM Net.Commerce Version 1.1 in 1996, there were several major revisions and IBM Net.Commerce was later renamed to IBM WebSphere Commerce Suite and IBM WebSphere Commerce. The earlier versions (IBM WebSphere Commerce V4 and IBM Net.Commerce) were based on a proprietary scripting language (Net.Data) and C++. Since IBM WebSphere Commerce V5, the product is built on the Java programming model, and supports open standards like EJB, JSP and XML technologies. Thus, the need for porting existing e-commerce applications for IBM customers to the newer versions of IBM WebSphere Commerce has become an important objective.

The application selected for examination is a “demo mall” for demonstration purpose that is packaged with the IBM WebSphere Commerce products. This demonstration system provides a showcase of an online shopping mall including user registration, address book, shopping cart management, and other associated shopping actions. It was originally written in Net.Data for the earlier versions of IBM WebSphere Commerce. The application is to be migrated for compliance with later versions of IBM WebSphere Commerce (IBM WebSphere Commerce V5).

Migrating the demonstration mall application involves two major tasks: Net.Data macro translation, and IBM WebSphere Commerce integration. The Net.Data macro translation is a fully automatic process by applying our transformation tool. The tool takes Net.Data source programs from the shopping mall application as input, and then generates corresponding JSP pages and JavaBeans objects. In order to test and display the transformation results, we need to integrate the new codes into the IBM WebSphere Commerce V5 platform. The following works were implemented to fulfill the IBM WebSphere Commerce integration:

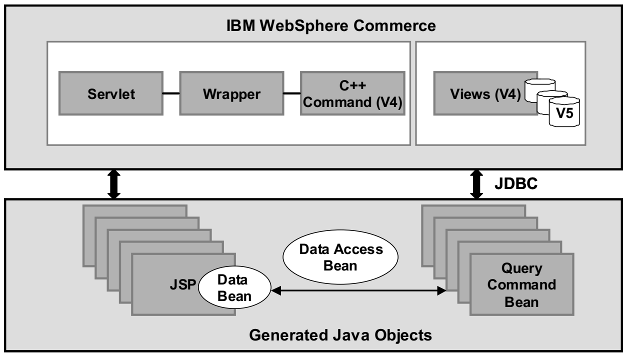

Figure 8: WebSphere Commerce **Integration**

- A new request servlet class was created in order to invoke the command objects that wrapped the C++ commands of IBM WebSphere Commerce V4.

- In IBM WebSphere Commerce V5, a new database schema has been introduced to support more complex business models. In order to use the SQL codes in existing Net.Data macros, we created views, which were based on the V4 schema, on top of V5 tables.

- The generated JavaBeans classes were compiled and packed into a Java Archive (JAR) file. The JAR file can be deployed to the server by placing it in the library folder.

- The new JSP pages must be registered in the view-related tables of an IBM WebSphere Commerce V5 database so that the request Servlet object can redirect the client request to them. In addition, the JSP pages were copied to the associated JSP folders in the server.

5.2 JSP-based Web Site Refactoring

The strategy and the architecture described in previous sections have also been used in an experiment to restructure a JSP-based Web site, which was originally developed to publish photography information to the Web. The Web site allows the user to browse and manage photographs by the categories or the location. It is implemented by using a free software, called PhotoDB, which is mainly composed of JSP pages and HTML pages [20].

We developed a reengineering tool used to support the refactoring process, which consists of four major implementation steps (Figure 9). The first step, the Page Preprocess, in our approach is to perform preliminary processing on a Web site in order to make Web pages ready to be analyzed in the second step. In particular, this step focuses on identifying the control block in Web pages. The control block can be recognized as either control statements such as IF-statement, WHILE-statement, and SWITCH-statement, or custom tags that

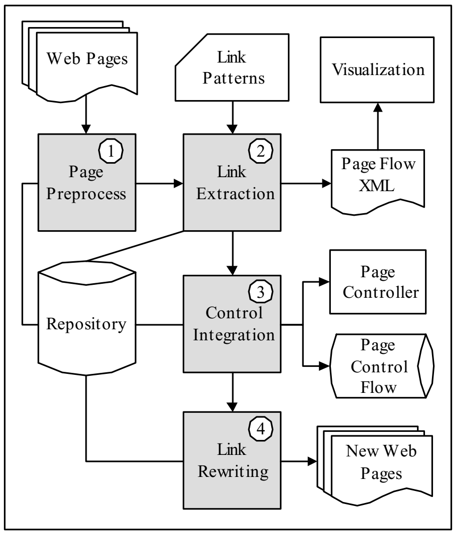

Figure 9: JSP Refactoring Implmentation

implement the control logic. Any links found in the control blocks are conditional links according to the categorization of the link type. Moreover, it discovers any implicit link information that cannot be directly handled by using the pattern matching technique in the following steps, specifically, the links inside Java codes, VBScript or JavaScript. Then it makes these implicit links to be explicitly specified by using a pre-defined method.

The Link Information Extraction step aims to analyze the preprocessed Web pages and to extract the embedded link information by using the pattern-matching technique. The lexical patterns, which are used to represent link relations, are stored in the Link Pattern component, and may be defined in different forms of expressions according to the actual implementation of Web programming language specifications. The Link Information Extractor, the implementation program of the Link Information Extraction step, will create an XML file to include the discovered link objects for each Web page. Then it will combine these newly generated XML file into a single XML file to represent the page flow for the whole Web site. As an extension of the Link Information Extraction step, the page flow XML file can be translated to an appropriate graphic language and thus the current Web site structure can be displayed by using the associated graph visualization tool, such as Dotty from ATT Laboratory [5]. During the Control Integration step, controller components of the new architecture is generated, as well as a corresponding page control flow is created according to the control structure. This can be accomplished by applying certain software clustering techniques based on the analysis of the page flow identified in the Link Information Extraction step. A server side program, such as Java Servlet, then implements the functionalities of the controller components. The newly created page control flow can be

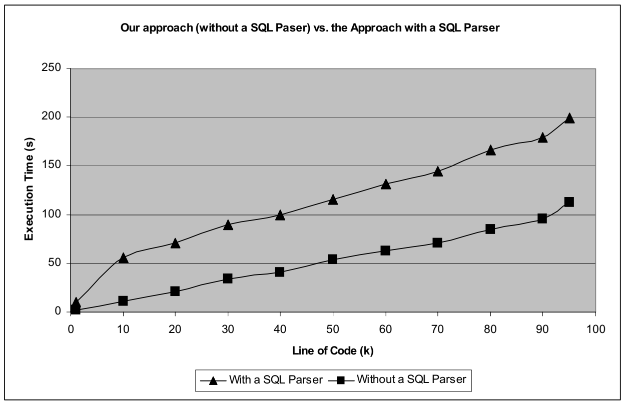

Figure 10. Time Complexity Comparison

stored in either a flat file in an XML format or a database system. The last step of the refactoring process is referred to the rewriting of the original Web pages by conforming to the generated control-centric architecture. This is performed by modifying the link information included in Web pages. As a result the destination of the link in a source page is replaced by an associate controller specified in the page control flow.

The above components store and share the results produced over the refactoring process in the Repository. In addition, the Repository contains the information related to the file directory, the Web site, the custom tag library, and other useful data, which are input from users or generated by other tools. The Repository can be either a database system or a set of collections of flat files.

5.3 Experimental Results

In this section we describe the result of our experiments on the migration of legacy Web applications to J2EE platforms. The tests were run on a PC with a 500Mhz processor and 256Mb of RAM memory. We tested the transformation application in the Eclipse V2.1 platform, which is an open extensible Integrated Development Environment (IDE). The Java JDK was version 1.4 and the operation system was Windows XP.

Compared with the existing transformation tools in [13,24], using the proposed tools gather the following benefits:

	Our Approach	SQL Parser Approach
Line of codes in the parser specification	944	1448
Size of the source language analyzer (Net.Data)	117KB	193KB

Table 1. Space Complexity Comparison

1. Reducing the time and space complexities of the transformation program:

Our transformation tool does not include a SQL parser to analyze the SQL statement extracted from the Net.Data source file. As a result, an Abstract Syntax Tree based on the structure of the SQL language is not generated during the transformation phase. Instead, a lightweight language source code representation is used. It makes the analysis process much faster than the existing transformation applications [13,24] that use a SQL parser when dealing with very complex SQL statements (Figure 10). Another advantage for not using a SQL parser is that it simplify the generation of the Net.Data Analyzer and significantly reduces the size of the Net.Data Parser Specification and Net.Data Analyzer, which do not need to implement the functionality to parse a SQL statement (Table 1).

2. Supporting presentation logic conversion and dynamic SQL statements transformation:

We extend and build upon existing Net.Data migration tools and techniques. As a result, the proposed tool includes a JSP translation program that can implement the transformation of Net.Data macros to JSP pages. It also supports generation of JavaBeans objects from either static SQL statements or dynamic SQL statements. In addition, The SQL Extractor provided in the proposed tool can reduce the complexity of the source code of JavaBeans objects by ignoring the column names (output parameters) specified in the SQL statement but not receiving values in the program. Moreover, the JavaBeans Generator provided in the proposed supports the automatic generation of data bean objects.

3. Supporting JSP pages refactoring:

Most Web site reengineering strategies leave the Web site structure untouched. The JSP Refactorer in the proposed tool, can automatically or semi-automatically implement the JSP-based Web site refactoring towards a well-engineered architecture, the controller-centric architecture. In this context, the tool extracts dependencies between JSP pages, replaces the original link information with symbolic link information, then provides controllers that map the symbolic link information to the actual link.

In conclusion, the prototype tool can successfully transform the legacy Web applications to a Java compliant one. Specifically, from the end user's point of view, the resulting Web pages created by the new Java platforms are exactly the same as the original ones in the legacy environment. In addition, this case study illustrates that the proposed framework can adapt to real life Web sites. The whole migration process is algorithmic and automated, except the JSP pages preprocessing and the generation of JSP pages clusters, where the human assistance is required to identify dynamic links and possible JSP page groups. However, the proposed tool can be further extended to support dynamic link analysis and page clustering in the refactoring process. Moreover, the evaluation result shows that the proposed tool has better performance and capability than the existing transformation tools. Finally, the structure of new Web system has been simplified from the original one. The structure comparison between the generated Web system and the old one is not presented here because of the space limitation.

6. Conclusion

The accelerated development of Web applications and the fast growth of associated Web technologies have resulted in a variety of maintenance concerns. One of the major maintenance problems is the porting and adaptation of existing Web applications into modern Web-based technologies. In this paper, we have addressed this problem by applying reengineering techniques, including the source code analysis, the software component extraction, and the Web application refactoring. We proposed a framework for incrementally migrating legacy Web systems to new platforms based on J2EE technologies. Using this framework, a source application was refactoried into JavaBeans format (Model), JavaServer Pages format (View), and Java Servlet format (Controller).

Future extensions of the work presented in this paper may focus on the following directions: Firstly, we will investigate the use of the wrapping technology and the connector architecture in order to integrate any other language environments imported by the existing legacy Web system. Secondly, the usage of a software clustering technique will be examined with the purpose of identifying cohesive groups of Web pages. A particular clustering algorithm needs to be developed in order to provide the support for the control integration of the migrant applications. Thirdly, we will investigate the technique for automatic implementation of analyzing dynamic Web components that can only be formed at run time, such as HTML form filling, and database querying. Finally, we will work on the generation and use of the Enterprise JavaBeans (EJB) technology, especially entity beans, for the back-end execution of SQL statements.

Acknowledgments

We would like to thank Ying Zou of Queen's University, Tack Tong and Bo Yi of IBM Canada Laboratory, Jianguo Lu of the University of Windsor, Joe Wigglesworth of IBM CAS, John Mylopoulos of the University of Toronto, as well as all the anonymous reviewers for their valuable suggestions, comments, and insights.This paper intends to represent the views of the authors rather than IBM.

Trademarks

IBM, Net.Data, Net.Commerce and WebSphere are trademarks of International Business Machines Corporation. Java, all Java-based marks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. Active Server Pages, and VBScript are trademarks or registered trademarks of Microsoft Corporation. DMS Software Reengineering Toolkit is a registered trademark of Semantic Designs, Inc. Other company, product, and service names may be trademarks or service marks of others.

References

[1] Amanda W. Wu, Haibo Wang, and Dawn Wilkins, "Performance Comparison of Alternative Solutions for Web-To-Database Applications", in *Proceedings the Southern Conference on Computing*, the University of Southern Mississippi, October 2000.

[2] Carmine Albanese, Thierry Bodhuin, Enrico Guardabascio and Maria Tortorella, "A Toolkit for Applying a Migration Strategy: a Case Study", in *Proceedings of the 6th European Conference on Software Maintenance and Reengineering* , 2002.

[3] Christy Lu, "A C to RPG Program Transformation Tool", M.Sc Project, University of Waterloo, Department of Electrical & Computer Engineering, 1998.

[4] Cornelia Boldyreff and Richard Kewish, "Reverse Engineering to Achieve Maintainable WWW sites", IEEE 2001.

[5] Dotty, AT&T, http://www.research.att.com/sw/tools /graphviz/

[6] Eric Armstrong, Stephanie Bodoff, Debbie Carson, Ian Evans, Maydene Fisher, Dale Green, Kim Haase, Eric Jendrock, Monica Pawlan, and Beth Stearns, "The J2EE 1.4 Tutorial", Sun Microsystems Inc., May 2003.

[7] Filippo Ricca and Paolo Tonella, "Using Clustering to Support the Migration from Static to Dynamic Web Pages", in *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, page 207-216, May 2003.

[8] Filippo Ricca, Paolo Tonella, Ira D. Baxter, "Web Application Transformations Based on Rewrite Rules", *Information and Software Technology*, 44(13):811-825, 2002.

[9] IBM, "IBM Net.Data Administration and Programming Guide", Version 7, June 2001.

[10] IBM, "IBM Net.Data Reference", Version 7, October 2001.

[11] Inderjeet Singh, Beth Stearns, Mark Johnson, and the Enterprise Team, "Designing Enterprise Applications with the J2EE Platform", Second Edition, Sun Microsystems Inc., 2002.

[12] James R. Cordy, Thomas R. Dean, Xinping Guo, Mykyta Synytskyy, Scott Grant, "The Whole Website Understanding Project", http://www.cs.queensu.ca/~stl /stg/.

[13] Jianguo Lu, "NetData4J: A NetData Parser Written in Java", October 2000. http://www.cs.Toronto.edu/~jglu /netData4j/

[14] Jianguo Lu, John Mylopoulos, "Automated EJB Client Code Generation Using Database Query Rewriting", in *Proceedings of the 7th International Database Engineering and Application Symposium*, Hong Kong, China, July 2003.

[15] Jon Kleinberg and Steve Lawrence, "The Structure of the Web", *Science*, Vol 294, pp. 1849-1850, November 2001.

[16] Karl Avedal, Danny Ayers, Timothy Briggs, Carl Burnham, Ari Halberstadt, Ray Haynes, Peter Henderson, Mac Holden, Sing Li, Dan Malks, Tom Myers, Alexander Nakhimovsky, Stephane Osmont, Grant Palmer, John Timney, Sameer Tyagi, Geert Van Damme, Mark Wilcox, Steve Wilkinson, Stefan Zeiger, and John Zukowski, "Professional JSP", Wrox Press Ltd., 2000.

[17] Kostas Kontogiannis, John Mylopoulos, Richard Gregori, Greg Mori, "Tools for the transformation of PL/IX based systems to C++ based systems", Technical Report, IBM Centre for Advanced Studies, August 1997.

[18] Leerina Aversano, Gerardo Canfora, Aniello Cimitile, and Andrea De Lucia, "Migrating Legacy Systems to the Web: an Experience Report", IEEE 2001.

[19] Meldoy Moore, Lilia Moshkina, "Migrating Legacy User Interfaces to the Internet: Shifting Dialogue Initiative", in *Proceedings of the 7th Working Conference on Reverse Engineering*, Brisbane, Australia, December 2000.

[20] PhotoDB, http://www.magiccookie.com.

[21] Ramzan Khuwaja, Corrado Balducci, Vesselin Ivanov, Manivannan Kubendranathan, Lisa Tong, Heidi Yan, Adrian Warman, "Best Practices and Tools for Creating IBM WebSphere Commerce Sites", IBM Redpapers, February 2003.

[22] Semantic Designs, Inc. DMS Software Reengineering Toolkit. http://www.semdesigns.com/Products/DMS/ DMSToolk it.html

[23] Terence C. Lau, Jianguo Lu, Erik Hedges, Emily Xing, "Migrating E-commerce Database Application to an Enterprise Java Environment", in *Proceedings of CASCON 2001*, page 68-78, Toronto, Canada, November 2001.

[24] Terence C. Lau, Jianguo Lu, John Mylopoulos, Kostas Kontogiannis, "The Migration of Multi-tier E-commerce Applications to an Enterprise Java Environment", *Information System Frontiers*, 5:2, pp. 149-160, 2003.

[25] Thierry Bodhuin, Enrico Guardabascio and Maria Tortorella, "Migrating COBOL Systems to the WEB by using the MVC design pattern", in *Proceedings of the 9th Working Conference on Reverse Engineering*, 2002

[26] Vesselin Ivanov, "Moving to a Java object environment: Best practices of WebSphere Commerce migration and LOQS", December 2002. http://cas.ibm.com/toronto /publications/TR-74.188 /27/ivanov.pdf

[27] Ying Zou, Kostas Kontogiannis, "A Framework for Migrating Procedural Code to Object-Oriented Platforms", in *Proceedings of the 8th IEEE Asia-Pacific Software Engineering Conference*, pp. 408-418, Macau, China, December 2001.

[28] Yu Ping, Jianguo Lu, Terence C. Lau, K. Kontogiannis, Tack Tong, and Bo Yi, "Migration of Legacy Web Applications to Enterprise Java Environments – Net.Data to JSP Transformation", in *Proceedings of CASCON 2003*, Toronto, Canada, October 2003.

[29] Yu Ping and K. Kontogiannis, "Refactoring Web Sites to the Controller-centric Architecture", in *Proceedings of 8th European Conference on Software Maintenance and Reengineering*, Tampere, Finland, March. 2004.