Migration of Legacy Web Applications to Enterprise Java™ Environments – Net.Data® to JSP™ Transformation*

Yu Ping¹ Jianguo Lu² Terence C. Lau³ Kostas Kontogiannis¹ Tack Tong³ Bo Yi³
University of Waterloo¹ University of Windsor² IBM® Canada Laboratory³
{yping, kostas}@swen.uwaterloo.ca jlu@cs.uwindsor.ca {lautc, tacktong, boyi}@ca.ibm.com

Abstract

As Web technologies advance, the porting and adaptation of existing Web applications to take advantage of the advancement has become an issue of increasing importance. Examples of such technology advancement include extensible architectural designs, more efficient caching protocols, and provision for customizable dynamic content delivery. This paper presents an experience report on the migration of legacy IBM® Net.Data® based applications to new enterprise Java™ environments. In this respect, a Net.Data application is refactored into JavaBeans™ (Model), JavaServer Pages™ (View), and Java Servlet™ (Controller). To evaluate the effectiveness of the migration methodology, a tool has been developed to support the automatic translation of Net.Data to JavaServer Pages. Using such a tool, a case study is presented to deal with IBM WebSphere® Commerce applications.

Keywords: Migration, Transformation, Model-View-Controller (MVC), Java 2 Enterprise Edition (J2EE™), JavaServer Pages, JavaBeans, Net.Data, and SQL.

1 Introduction

With the widespread use of Java technologies, many new large distributed applications are being developed using the Java 2 Enterprise Edition (J2EE) platform [9]. This has triggered a plethora of research with the main objective to leverage existing Web applications, which were designed by less flexible architectures or implemented by older Web technologies into J2EE platforms. J2EE technology introduces a component-based architecture that provides an open standard for the design, development, and deployment of multi-tiered enterprise applications. In this respect, to facilitate the reengineering of legacy Web systems in enterprise Java environments, our approach proposes a framework to support not only the language translation but also the architecture evolution. In particular, the proposed framework is based on the migration of IBM Net.Data [7,8] to JavaServer Pages (JSP™) [21] that aims at the separation of the database access functionality from the presentation logic. As a result, the SQL statements are extracted from the source programs, and then encapsulated in JavaBeans objects [20]. Consequently, the JSP pages translated from the legacy presentation components can reference these newly created JavaBeans objects. Thus, the target migrant system would be more maintainable than the original system because it adopts the Model-View-Controller (MVC) architecture.

The remainder of the paper is organized as follows. Section 2 provides background information concerning the source and target
systems for our migration task. Section 3 discusses the design consideration for the target system, while Section 4 outlines the transformation framework, including migration methodology and architecture. Section 5 and Section 6 describe in detail the process for SQL information extraction and JSP translation. Section 7 discusses a case study based on IBM WebSphere Commerce applications. Section 8 provides the comparison to the related work and Section 9 offers conclusions and directions for future research.

2 Background

2.1 Net.Data

Net.Data [7,8] is a server-side scripting language developed by IBM, which operates as middleware infrastructure on a Web server. Net.Data application programs, called macros, enable the dynamic generation of Web pages from a variety of data sources as well as flat files. Net.Data macros consist of two parts: the declaration part that defines variables and functions used in macros, and the presentation part that specifies the layout of Web pages and acts as entry points of the Net.Data macros. Figure 1 illustrates a general structure of a Net.Data macro with a simple example (sample.d2w).

One of the main possible limitations of Net.Data macros is that they allow the mingling of the presentation and business logic, by having SQL statements in the FUNCTION blocks to access the database directly. Such structure violates the clear separation of database access components from the presentation logic. Therefore, Web page customization may become complicated because it requires significant interaction between the application developers and graphic designers during the development process. Another possible limitation using Net.Data is low availability of resources, tools, and programmers to support the design and development of new Net.Data applications. These reasons trigger the transition of Web application development from using Net.Data to the more flexible technology of JavaServer Pages.

2.2 JavaServer Pages and JavaBeans

JavaServer Pages (JSP) [21] is one of the core server-side technologies in the enterprise Java environment. JSP is a web presentation layer built on top of the Java Servlet framework. A Java Servlet is a Java program that runs on a Web server. It acts as a middle layer between requests coming in from clients and back-end applications. Contents in a JSP page can be broken into two categories: JSP elements, including directives, standard actions and scripting elements, and JSP template data, such as HTML and JavaScript. JSP elements are processed on the server, whereas JSP template data are ignored by the server and sent directly to the client.

JavaBeans [20] support reusable components to be written in the Java language, which encapsulate the logic behind the Web applications. With the combination of JavaBeans, JSP technology allows the separation of the database access functionality from the display logic. Additionally, there are plenty of JSP application development resources and tools available. Thus, JSP technology enables the design, implementation, and deployment of very scalable and portable dynamic Web sites.

2.3 SQL Statement

Net.Data uses SQL statements for creating, updating, and querying back-end database systems. In this paper, we classify SQL
statements included in Net.Data macros into two types: static SQL statements and dynamic SQL statements (Figure 2).

| Static SQL Statement: | SELECT firstname FROM customer WHERE lastname = 'Smith' |
| Prepared Dynamic SQL Statement: | SELECT firstname FROM customer WHERE lastname = '$(lname)' |
| Non-prepared Dynamic SQL Statement: | SELECT $(fname) FROM customer WHERE lastname = '$(lname)' |

Figure 2: SQL Statements

A static SQL statement is a complete SQL statement. That is, the whole statement can be determined at source code level. Such SQL statements do not change each time when the program is executed. Hence, the static SQL statement must not contain any variables.

A dynamic SQL statement is defined as an incomplete SQL statement, some or all of which must be supplied at run time. Such SQL statement contains one or more program variables, and values of the variables are to be obtained when the program is executed. A dynamic SQL statement is a prepared dynamic SQL statement, if this SQL statement can be precompiled and stored in a Java PreparedStatement object. All of the variable references in the statement are replaced by question marks (?) as parameter markers, which act as placeholders for input data provided at run time. Otherwise, it is a non-prepared dynamic SQL statement.

2.4 Abstract Syntax Tree

At the lowest level of abstraction, Abstract Syntax Trees (ASTs) [27] have been successfully used by the data flow analysis community in order to analyze and transform source code entities. These trees contain information about the source program in the form of nodes and edges. Such tree-like structures represent the source program in a top-down manner. For example, Net.Data macros are represented at the top level as HTML statements, function statements, and define statements, while at lower levels as function calls, variable reference and SQL statements, to name a few. The internal nodes of the AST represent the non-terminal phrases such as statements and functions, and the leaf nodes represent the terminal symbols, such as strings and key words. An edge denotes tree attributes, which are represented as mappings between AST nodes. AST nodes correspond to programming language constructs such as IF-Statements, FOR-Statements, and WHILE-Statements.

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 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 [9] 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 3).

![Model-View-Controller Architecture](image)

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.

![Data Beans Architecture](image)

**Figure 4: 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 4). 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.

### 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 the following, we describe the framework in detail.

#### 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.

![Transformation Methodology Diagram]

**Figure 5: Transformation Methodology**

**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 predefined “data bean architecture”, and thus to isolate the model from the view.

**Phase 2. Making Transition to JSP:** A Web legacy application commonly uses proprietary constructs to customize display pages. JSP is 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 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 previously presented. In this context, the third phase focuses on the analysis of application control flow that 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.

In this paper, we focus mostly on reengineering techniques pertaining to Phase 1 and Phase 2, that is the separation of database access from presentation logic, and the deployment of the new components to JSP compliant architectures.

### 4.2 Transformation Architecture

The migration methodology outlined in the previous section is supported by a transformation architecture that has two subprocesses, database access transformation (Phase 1) and presentation logic transformation (Phase 2). Database access and presentation logic transformations can be performed independently after the generation of the source code representation, and independently of the choice of the target language.

Furthermore, modularity is another major design consideration in our transformation architecture. As a result, the proposed reengineering process contains five layers, which represent source system, software analysis, information extraction, new code generation and target system, from the top to the lowest level. These design approaches have resulted in the overall transformation architecture shown in Figure 6.

The software analysis layer focuses on representing the source of the system being analyzed at an appropriate level of abstraction, for instance, in the form of an annotated Abstract Syntax Tree (AST). In this layer, the **Language Analyzer** reads the source code 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, which stores the intermediate representation of the input program. The Language Analyzer also provides an Application Program Interface for the AST. The API is used by the SQL Information Extractor and Display Information Extractor to access the information from the AST.

The information extraction layer aims at the extraction of associated facts by a series of iterative analysis steps applied at the AST level. There are two extraction components in this layer, the **SQL Information Extractor** that is a part of the database access transformation process, and the **Display Information Extractor** that is a part
of the presentation logic transformation 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 a SQL properties file that stores the SQL statements, and a parameter container file that contains the input and output parameters, and other SQL information. 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.

We provide further discussion about the SQL Information Extractor in Section 5 and presentation logic transformation in Section 6, respectively. All of these discussions are based on the migration of Net.Data macros to JSP pages.

5 SQL Information Extraction

As one of the core components of the database access transformation process, the SQL Information Extractor focuses on the extraction and analysis of SQL statements in the legacy Web program. In order to generate JavaBeans objects, the SQL Information Extractor must obtain the SQL statements, input parameters that provide input data at run time, and output parameters that are the result sets after execution of the SQL statements, from the source program. Consequently, it generates a SQL properties file and a parameter container file that are used as the inputs for the JavaBeans Generator. The major components in the SQL Information Extractor are illustrated in Figure 7:

![Figure 7: SQL Information Extractor](image-url)
5.1 Function Analyzer

The Function Analyzer extracts function information, such as function names, SQL statements and variable names, from SQL FUNCTION blocks in the source program. A SQL FUNCTION block defined in Net.Data is used for executing a SQL statement. The function body contains a SQL statement, a number of REPORT and ROW blocks that are for formatting output from a SQL statement, and a MESSAGE block which specifies messages to display and actions to take based on the return code from a function. The Function Analyzer has two major subsystems: the SQL Analyzer and the Row Analyzer.

The SQL Analyzer collects input parameter candidates in the SQL Statement, which is extracted from the FUNCTION block, and sends the candidates, one by one, to the Input Parameter Identifier for validation. All of the variables, which are referenced in the SQL statement, are candidates to be considered as input parameters. Each dynamic SQL statement contains at least one input parameter candidate. For example, the SQL statement,

```
SELECT $(store_manager) FROM $(store_name) WHERE store_id = 1
```

has two input parameter candidates, namely “store_manager” and “store_name”, respectively. A static SQL statement, however, cannot contain program variables; therefore, there are no input parameter candidates in a static SQL statement. The SQL Analyzer does not check whether a variable has been referenced more than once in the SQL statement, so an input parameter candidate may be a duplicate one. It also updates the output list according to the return value from the Output Parameter Identifier.

5.2 Parameter Identifier

The Parameter Identifier is designed to make the selection of the valid input and output parameters from the given candidates.

The purpose of the Input Parameter Identifier is to identify duplicate parameters from the input parameter candidates. When it receives a candidate from the SQL Analyzer, the Input Parameter Identifier checks the target input list. If the candidate already exists in the list, the Input Parameter Identifier deduces that the input candidate is a duplicate one. Otherwise, the Input Parameter Identifier sends back the candidate as a valid input parameter to the SQL Analyzer.

The purpose of the Output Parameter Identifier is to identify an output parameter candidate if it matches with predefined patterns of output parameters. The pattern is in the form of “V_name” (“name” is a column name), or “Vn” (“n” is a column number). The Output Parameter Identifier also checks the duplication of the parameters. As a result, it returns a valid output parameter to the ROW Analyzer.

5.3 Parameter Container

The Parameter Container uses XML to model the infrastructure of the generation of JavaBeans objects. It contains the input list and the output list for each SQL statement extracted from the source program. It also describes the file directory, parameter properties and name conversions used by the transformation components. Figure 8 is the XML file (loqs.xml) created from the Net.Data sample codes (sample.d2w).

The input list contains names and types of input parameters required for a SQL statement. Each SQL statement can only have one input list. The input list will be empty when the SQL statement is static. It is defined between the <query> and </query> tags in the XML file (loqs.xml).
The **output list** contains names and types of output parameters for a SQL statement. Each SQL statement can only have one output list and the output list must contain at least one parameter. If there is no output parameter found for the SQL statement, the SQL Information Extractor will add an output parameter, namely “V1”, into the output list. It is defined between the `<ResultSet>` and `</ResultSet>` tags in the XML file (loqs.xml).

### 5.4 SQL Properties File

```xml
<?xml version="1.0" ?>
<LOQSGen outputDirectory="c:\NetDatatoJSP\Target\JavaBean">
  <QueryCommand class="sample_bean.query" superclass="AbstractQueryCommand">
    <query name="sample_bean.query">
      <iParam name="fname" type="String" />
      <Param name="name" type="String" />
    </query>
  </QueryCommand>
</LOQSGen>
```

Figure 8: loqs.xml

The SQL properties file is a flat file that records all the SQL statements extracted from the Net.Data source file. Each SQL statement in the SQL properties file has a unique key, which consists of the file name and the function name. For example, if an SQL statement,

```
SELECT * FROM customer WHERE store_id = 1
```

is included in the FUNCTION block (called “query”) in the file (called “shopper.d2w”), the record in the SQL properties file will be:

```
shopper_bean.query = SELECT * FROM customer WHERE store_id = 1
```

### 6 Presentation Logic Transformation

Net.Data applications generate Web pages by using Net.Data macros, which consist of Net.Data elements, HTML tags, and JavaScript. Contrary to Net.Data systems, migrant applications use JSP pages, with a combination of Java technology, to implement the Web interface and display logic. Therefore, a translation program that transforms the representation of presentation components from Net.Data to JSP is required. This section discusses how a JSP translation program converts Net.Data macros to JSP technology, and structures JSP pages with other generated components, such as variable definitions, built-in functions, and JavaBeans objects.

#### 6.1 Conversion Specification

To build a presentation transformation system, a logical architecture with four components has been identified. These components, their major functionalities and interdependencies are represented in Figure 9, and explained in the following subsections.

Figure 9: Presentation Component Transformation

The JSP translation program converts Net.Data macros to JSP pages by conforming to a given specification, which defines a set of transformation routines. This specification, called a Conversion Specification, provides a description of a set of predefined mapping rules from Net.Data macros to JSP pages, which includes Naming Conversion, Variable Conversion, Construct Conversion and Built-in Function Conversion.
6.1.1 Naming Conversion

The Naming Conversion presents a set of guidelines to standardize the name creation for JSP pages and JavaBeans objects. The new names should be unique in the file system, and consistent with the original names of Net.Data macros, and thus allow a greater opportunity for fuller object names. Table 1 shows the file names of JSP pages generated from Net.Data DEFINE block, FUNCTION block, and HTML block. Table 2 shows the class names of JavaBeans objects generated from a SQL statement in the Net.Data FUNCTION block.

<table>
<thead>
<tr>
<th>JSP Page Generated from</th>
<th>File Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFINE block</td>
<td>fileName_define.jsp</td>
</tr>
<tr>
<td>FUNCTION block</td>
<td>fileName_functionName.jsp</td>
</tr>
<tr>
<td>HTML block</td>
<td>filename_htmlName.jsp</td>
</tr>
</tbody>
</table>

Table 1: File Name Conversion

<table>
<thead>
<tr>
<th>JavaBeans Objects</th>
<th>Class Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Access Bean</td>
<td>FuncNameDAO.java</td>
</tr>
<tr>
<td>Data Bean</td>
<td>FuncNameDataBean.java</td>
</tr>
<tr>
<td>Query Command Bean</td>
<td>FuncName.java</td>
</tr>
</tbody>
</table>

Table 2: Class Name Conversion

6.1.2 Variable Conversion

<table>
<thead>
<tr>
<th>Net.Data Variable</th>
<th>JSP Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Declaration (String only)</td>
<td>JSP Variable Declaration</td>
</tr>
<tr>
<td>Variable Reference</td>
<td>1. Java Variable</td>
</tr>
<tr>
<td></td>
<td>2. JSP Expression</td>
</tr>
<tr>
<td>Table Processing</td>
<td>Getter method in data bean</td>
</tr>
<tr>
<td>V_name</td>
<td>DataBeanArray[i].getName()</td>
</tr>
<tr>
<td>Vn</td>
<td>DataBeanArray[i].getVn()</td>
</tr>
</tbody>
</table>

Table 3: Variable Conversion

Net.Data provides two types of variables: user-defined variables that are defined by application developers, and Net.Data variables that are defined by Net.Data language for file manipulation, table processing, and report formatting. Table 3 shows some examples of conversion rules from Net.Data variables to JSP elements.

6.1.3 Construct Conversion

A Net.Data program consists of a series of language constructs to define variables and functions, and perform special tasks such as specifying the layout of Web pages. To migrate Net.Data applications to JSP systems, the transformation program needs to convert Net.Data constructs to an appropriate JSP component with the same functionality. The Construct Conversion specification provides a set of mapping disciplines between Net.Data constructs and their corresponding JSP elements in order to support the migration. Table 4 provides the conversion guidelines for common Net.Data constructs.

<table>
<thead>
<tr>
<th>Net.Data Construct</th>
<th>JSP Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMMENT</td>
<td>JSP hidden comment</td>
</tr>
<tr>
<td>DEFINE</td>
<td>Declaration part, or JSP scriptlet</td>
</tr>
<tr>
<td>FUNCTION</td>
<td>JSP page</td>
</tr>
<tr>
<td>Function call</td>
<td>JSP include directive</td>
</tr>
<tr>
<td>HTML</td>
<td>Main JSP page</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>JSP include directive, or <a href="">jsp:include</a> element</td>
</tr>
<tr>
<td>IF</td>
<td>Java IF-statement, or JSP IF-tag</td>
</tr>
<tr>
<td>WHILE</td>
<td>Java WHILE-Statement, or JSP WHILE-tag</td>
</tr>
<tr>
<td>REPORT</td>
<td>A part of the JSP page generated from Net.Data FUNCTION block</td>
</tr>
<tr>
<td>ROW</td>
<td>Java FOR-loop statement</td>
</tr>
<tr>
<td>MESSAGE</td>
<td>Java IF-statement, or Java TRY-CATCH statement</td>
</tr>
</tbody>
</table>

Table 4: Construct Conversion

6.1.4 Built-in Function Conversion

Net.Data provides a wide variety of built-in functions to simplify the Web application development, such as math functions, string manipulation functions, and file interface functions. In the target system, Net.Data built-in functions are rewritten by Java methods with similar functionalities in the Java library. One or more Java classes, which are packaged into a Java Archive (JAR™) file, include those generated Java methods. The generated JSP page can import this JAR file with other Java packages by specifying them in the JSP page directive.
6.2 Entity Register

An Entity Register is a place where the translation programs store their intermediate results over the transformation process. It serves as a history record retrieved by the transformation program in order to generate corresponding JSP elements.

JavaBeans Register: A JavaBeans Register records the information of the JavaBeans objects generated from SQL statements, such as class names of JavaBeans objects, instance names of JavaBeans objects used in JSP pages, and names of Java packages that contain JavaBeans objects. The JSP transformation program uses this information to compose <jsp:usebean> elements and data bean invocations in the target JSP pages.

Variable Register: A Variable Register keeps a single copy of each Net.Data user defined variable identified by the transformation program.

Function Register: A Function Register saves the names of user defined functions, the names of the associated JSP pages, and information of input and output parameters. The transformation program uses this information when converts Net.Data function calls to JSP include statements.

6.3 Display Information Extractor

Net.Data macros consist of two components: a declaration part that mainly includes DEFINE blocks and FUNCTION blocks, and a presentation part that can only contains HTML blocks. Hence, extracting and analyzing DEFINE, FUNCTION, and HTML blocks is a major task of a JSP transformation program. In this respect, a Display Information Extractor acts as a transformation engine to perform such a task.

The Display Information Extractor first retrieves the AST generated by the Net.Data Language Analyzer from the Net.Data source program to gather the information about DEFINE blocks, FUNCTION blocks, and HTML blocks. And then, these three major blocks in combination with the elements they include, such as IF block, REPORT block, and MESSAGE block, are analyzed by the Display Information Extractor. The Display Information Extractor also initializes and sets up the Entity Register during the analyzing phase. Finally, the Display Information Extractor sends the results to the JSP Generator for the construction of complete JSP pages. It ignores the SQL statements that are processed by the SQL Information Extractor, and copies the static HTML and JavaScript to the resulting pages.

6.4 JSP Generator

A JSP Generator is responsible for constructing the final JSP pages: main JSP pages, define JSP pages, and function JSP pages, which are converted from HTML blocks, variable definition statements (DEFINE blocks and any other implicit declarations), and FUNCTION blocks, respectively. The major task executed in the JSP Generator is to convert the Net.Data macros to corresponding JSP elements by conforming to a given specification, and consequently finalizing the JSP pages by adding header and footer elements, such as a JSP page directive, and JavaBeans instantiation and invocation. Figure 10 represents the new JSP pages converted from Net.Data example codes (sample.d2w).
7 A Case Study

To demonstrate the effectiveness of the framework advocated in this paper, we have developed a prototype tool written in Java programming language, which implements all of five major components presented in our proposed transformation architecture. We also had it tested for migrating IBM WebSphere Commerce applications.

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++ [1]. 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 (Figure 11). This demonstration system provides a showcase of an online shopping mall including user registration, address book, shopping cart management, and other associated shopping actions [16]. 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:

- 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.

In conclusion, the prototype tool can successfully transform the demonstration mall application 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 Net.Data environment.
8 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. [14] present a tool to migrate COBOL systems into Web applications based on the methodologies described in MORPH project [15]. 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 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 [2,14,25] 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 [2,14,25], whereas we encapsulate database access components (SQL statements) into JavaBeans objects by conforming to the predefined “data bean architecture”.

Ricca et. al. [6] 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 [18]. In another paper [5], 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 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 [5] the idea of using the clustering technique to restructure the Web sites in Phase 3 of our proposed transformation framework. However, we aim to refactor the Web sites by adopting a controller-centric architecture instead of the frame-based one described in [5].

The work most related to ours is [10]. 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 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.

Tree at the end of the parsing, instead of executing the application. It also includes a SQL language parser to analyze SQL statements extracted from Net.Data macros.

The Net.Data-to-JSP Migration Helper Tool [22, 24] was first released in June 2001 on the IBM alphaWorks Web site [23], and later incorporated in the IBM Transition Tool Suite (TTS) [17]. TTS is an integrated set of helper tools that support the transition of existing WebSphere Commerce sites built on the IBM WebSphere Commerce V4 framework to V5. The Net.Data-to-JSP tool is not a language translation tool that automatically converts Net.Data to JSP Pages. Rather, it is an assistant tool to provide recommendations on the migration of SQL statements included in Net.Data programs to Enterprise JavaBeans (EJB), which can then be further invoked by JSP Pages.

Lightweight Object Query System (LOQS) [26] is another helper tool in TTS that is designed to support transformation of SQL statements to Java objects. LOQS consists of a run-time engine and a Java code generation tool. The LOQS runtime engine is dedicated to the executing of read-only database queries in IBM WebSphere Commerce V5 environment. The generation tool takes a SQL statement and associated parameters as input, and stores the input information into a XML file. As a result, the tool generates the required Java codes that can be deployed and executed with the LOQS run-time engine.

Lu et. al. [12] describe the automated generation of EJB client code in JSP pages that is equivalent to an SQL query. This paper focuses on SQL query translation and is independent on the presentation formalisms. Hence it does not cover the Net.Data translation itself.

Most of the existing 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 such as:

```
SELECT $(column_list) FROM $(customer)
```

or any statements of which the column names are implicitly specified such as:

```
SELECT * FROM shopper
```

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.

In this paper, we extend and build upon existing Net.Data migration tools and techniques. Within this respect, we present a prototype transformation tool, which deals with not only static and dynamic SQL statements but also presentation components. Using this tool we can gather the following benefits:

**Supporting presentation logic conversion:** The tool includes a JSP translation program that can implement the transformation of Net.Data macros to JSP pages.

**Performance of the SQL statement analysis:** The 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 other transformation applications that use an SQL parser when dealing with very complex SQL statements. Another advantage for not using an SQL parser is that it significantly reduces the program size of the Net.Data Analyzer, which does not need to implement the functionality to parse an SQL statement.

**Handling of dynamic SQL statements:** The tool supports generation of JavaBeans objects from either static SQL statements or dynamic SQL statements.

**Reducing the complexity of JavaBeans classes:** The SQL Extractor provided in the tool can reduce the complexity of the source codes of JavaBeans objects by ignoring the column names (output parameters) specified in the SQL statement but not receiving values in the program.
Creating Data Bean: The JavaBeans Generator provided in the tool supports the automatic generation of data bean objects.

Finally, there has been extensive research work conducted on programming language translation, such as C to RPG [3] or PL/I/X to C++ [13], 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. Net.Data involves 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 Net.Data.

9 Conclusion

In this paper, we presented a reengineering approach to migrate Web systems written in Net.Data macros into enterprise Java platforms. We also described a detailed process whereby the database access functionality was isolated from the displayed pages. In this context, SQL source statements can be encapsulated in JavaBeans objects. Consequently presentation components need to access the encapsulated services from within J2EE enabled platforms. This can be achieved by utilizing JSP technology. Moreover, the migration framework discussed in this paper has been evaluated with a transformation tool that was implemented and tested on IBM WebSphere Commerce applications. In addition to IBM WebSphere Commerce, our proposed framework can be applied to other Web system migrations.

Future extensions on the work presented in this paper may focus on two directions. The first direction is on investigating the use of the Java Native Interface™ (JNI™) or Java connector technologies in order to integrate the language environments included in Net.Data macros. The second direction is on developing a technique of discovering application control flows in order to provide the roadmap for the control integration of the migrant applications.

We will also work on the generation and use of JavaBeans, especially entity beans. In the tools reported in this paper, each legacy SQL statement is wrapped as a session bean. However, the power and motivation of EJB lies in entity beans. We will investigate the automated generation of entity beans from a legacy application, based on the existing database schema and SQL statements.

Acknowledgments

We would like to thank Ying Zou of the University of Waterloo, Vesselin Ivanov of IBM Canada Laboratory, Norbert Runge of IBM Santa Teresa Laboratory, 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.

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

This paper intends to represent the views of the authors rather than IBM.

About the Authors

Yu Ping is a visiting research student at the IBM Centre for Advanced Studies. He is currently working towards a Master of Applied Science degree in Electrical and Computer Engineering at the University of Waterloo under the supervision of Professor Kostas Kontogiannis.

Dr. Kostas Kontogiannis obtained his Ph.D from McGill University, School of Computer Science. He is an Associate Professor at the Department of Electrical and Computer Engineering, University of Waterloo where he is leading the Software Re-engineering group. Kostas' research focuses on the design and implementation of tools and techniques that allow for large legacy software systems to be re-engineered and integrated in distributed and Network-Centric environments. He is the recipient of one Canada Foundation for Innovation Award and of three IBM University Partnership Program Awards.

Dr. Terence C. Lau is a senior research staff member at the Centre for Advanced Studies, IBM Canada. Previously, he was a senior system architect in various IBM product groups in e-commerce, imaging, and data communication, and a faculty member of the University of Hong Kong. Dr. Lau received a
B.Sc. from University of Hong Kong and a Ph.D. in computer science from University of Waterloo.

Dr. Jianguo Lu obtained his Ph.D. and Master’s Degrees from Nanjing University in Computer Science. Presently, he is an Associate Professor of Computer Science at the University of Windsor and a visiting scientist at the IBM Toronto Lab. Before joining the University of Windsor, he worked as research associate for the University of Toronto and Carnegie Mellon University, and as Associate Professor for Fudan University. His research areas are software engineering, artificial intelligence, and database systems.

Tack Tong is an architect in Electronic Commerce Development at IBM Toronto Laboratory. He received his B.Sc. from the University of Hong Kong.

Dr. Bo Yi obtained his PhD. from Nanjing University in Computer Science. He joined IBM in 1999 as an IT Specialist. He has worked on Browser-based Application Toolkit (BAT), Reporting Framework, Net.Data support in WebSphere Commerce and Data Migration Framework for WebSphere. Presently he works in WebSphere support team.

References


