Migrating Web Frameworks Using Water Transformations

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

ABSTRACT
We propose an approach (based on *Water Transformations*) to migrate web applications between various web development frameworks. This migration process preserves the structure of the code and the location of comments to ease future manual maintenance of the migrated code. Developers can move their applications to the framework that meets their current needs instead of being locked into their initial development framework. We give an example of using our approach to migrate a web application written using Active Server Pages (ASP) framework to Netscape Server Pages (NSP) framework.

1 INTRODUCTION

A number of frameworks for web application development have been introduced to solve various problems associated with the development of web applications. Java Server Pages (JSP), Netscape Server Pages (NSP), Allaire Cold Fusion (CF) and Active Server Pages (ASP) are some of the most common frameworks in web application development. Each framework provides support for essential functions to expedite the development of web applications. For example, the NSP framework uses the JavaScript language along with a set of built-in libraries that are accessible through JavaScript. Similarly, ASP framework uses the VBScript language and a set of supporting libraries.

Current web application development emphasizes implementation productivity with little concern for maintenance and evolution of the applications, because of the fast pace of the industry and its immaturity [8]. Some frameworks provide a rich set of libraries to speed up the initial product release, but they usually suffer from many performance bottlenecks, which hamper the application's evolution. Furthermore, applications need to be migrated to new frameworks when their current framework is no longer supported or lacks new technology features.

Unfortunately, the choice of a framework is done early in the life of the project and too often with little thought about the future impact of such choice. Once a web application is built on top of a framework, migrating it to a new framework to meet new demands is a challenging task, especially under tight schedules.

In this paper, we propose an approach for migrating from one web development framework to another, in particular we show an example of migrating a web application from the ASP to the NSP framework. To facilitate future maintenance of the migrated code, our approach preserves the structure of the code and the location of comments in the migrated code.

The approach is based on *Water Transformations*, an extension of *Island Grammars* [13], which have been previously used for reverse engineering, software understanding, and impact analysis of traditional [15, 16] and web based legacy software systems [10]. The approach is largely automated. When human intervention is needed, the tool skips the code segments and marks them. Later, the developers must review and manually migrate the marked code segments.

Through our approach, developers can move their applications to the framework that meets their current demands instead of being locked in their initial development framework.

Organization of Paper

The rest of this paper is organized as follows. Section 2 describes web application frameworks. Section 3 presents our migration process, introduces the concept of *Water Transformations* and explains how we use such transformations to ease the migration of multi-lingual software systems. Section 4 gives a detailed example of the migration using the formal approach presented in the previous section. Section 5 contrasts our work to other research projects which address similar problems. Section 6 draws conclusions from our work and proposes future directions.

2 WEB APPLICATION FRAMEWORKS

Each web application framework provides support for the essential functions for the development of a web application, notably:

- programming language support
- a set of built-in objects (an object model).

A Web application is composed of active pages and static pages. Static pages are written in pure HTML. Active pages are like static pages but they contain *active* control code written in languages such as VBScript, Java, and JavaScript. HTML tags are used to layout the data on the screen, and control code uses the functionalities offered by the framework to customize the web page. Figure 1 shows an example of an active page written in VBScript.

```
1: <HTML><TITLE>Main Page</TITLE>
2: <% If Session("LoggedIn") Then 'User already logged in %>
3: Welcome
4: <% Response.Write(Session("username")) %>
5: to your account, to display balance
        <A HREF="balance.asp">click here</A>
6: <% Else 'User needs to login %>
7: Please login first, <A HREF="login.asp">click here</A>
8: <% End If %>
9: </HTML>
```

Figure 1: Example ASP File

When a client requests a static page, the server returns the page without modification. When a client requests an active page, the server preprocesses it. The control code is executed and the result of the execution is merged with the static content and returned to the requesting browser. The example shown in Figure 1 when viewed in a browser will appear as a page with title `Main Page`. The page will either welcome the logged-in user or ask the user to log into the system.

WEB FRAMEWORKS PROGRAMMING LANGUAGES

Various programming languages are used in the development of active pages in a web applications. VBScript, Java, ColdFusion, and JavaScript are the most commonly used ones.

VBScript is the main programming language for the Active Server Page (ASP) framework. VBScript is a subset of the Visual Basic language developed by Microsoft Corporation. It is a scripting language that is interpreted and runs primarily on Microsoft Windows. It is a weakly typed language with only one data type (`Variant`). It supports procedural and object-oriented programming paradigms, but it is not a true object-oriented language. Instead, it is an object-based language that permits developers to specify objects (in terms of methods and properties) but does not support other object-oriented features such as inheritance. The VBScript language is not as well documented as the JavaScript Language. No extensive reference manual exists that describes the language in detail. To develop our migration tool, we examined many systems and code samples written in VBScript to infer the language's structure and syntax.

JavaScript, also called ECMAScript [7], is the main programming language used in the Netscape Server Pages (NSP) framework. It is an object-based scripting language; it is not a subset of any other language. It shares many similarities with object-oriented languages such as the Java and C++ languages. The language is developed under the control of the ECMA [6] standardization organization. JavaScript is the name of Netscape Corp.'s implementation of the ECMAScript language whereas JScript is Microsoft Corp.'s name. Both implementations are supersets of the ECMAScript language; they provide extensions specific to the implementing company. The extensions add many built-in objects and provide mechanisms for the language to interact more easily with other components in the NSP framework. The ECMAScript language can be interpreted or compiled. It has many built-in `data types` such as `String`, `Number`, `Boolean`, `Object`, `Array`, `Null`, and `Undefined`. New types cannot be defined. The language is loosely typed; the data type of variables do not need to be defined ahead of time and conversion between the different data types is done automatically without the need of a cast operator. The data type of a variable is based on the value of the variable at run-time.

ColdFusion (CF) is another web development framework, developed by Macromedia Corp [1]. Whereas JSP and ASP are based on active code embedded in HTML, CF uses a specialized markup language called Cold Fusion Markup Language (CFML) which is embedded in HTML as well. CFML has tags which represent the control flow of a traditional programming language such as `<CFIF>` or `<CFELSE>`. Figure 2 shows a program written in the CFML markup language.

```
1: <HTML>
2: <CFOUTPUT>Hello World</CFOUTPUT>
3: </HTML>
```

Figure 2: "Hello World" Active Page Written in Cold Fusion

Finally, Java Server Pages (JSP) framework permits the embedding of active code written in the Java programming language, a strongly typed object-oriented language which shares many similarities with the Modula-3 and C++ languages.

WEB FRAMEWORKS COMMON WEB OBJECT MODEL

Each language is supported by a set of built-in objects which are provided by each web application development framework. These objects abstract the commonly needed functionalities in the development of a web application, such as access to the client's request or the maintenance of the client's state across multiple HTTP requests. As we studied various web applications development frameworks we were able to define a set of common objects across all frameworks:

Request

The Request object represents a server side model of a web browser request. It is commonly used to retrieve information entered by a user in a form or to retrieve cookies stored on the client side.

Response

The Response object represents the information sent from the web server to the web browser. It is used to write output back to the browser.

Session

The Session object represents a particular user session. A user session starts when the user's browser requests the first

page from a web site. It ends after a preset period of time since the last page request by the user. The session can also be terminated on demand, for example if the users log out from the web application before shutting down their browsers. This object stores the session variables that retain their values as the user browses from one page to the next in the web application.

Application
The Application object represents a global space for the whole web application. A web application starts when the first page in the application is requested by a browser. It ends when the application server is terminated. Application variables are shared between all the active pages. An Application object may store a variable to count how many users have accessed the application since it started. Also it can be used to pool database connections for reuse by the active pages.

Server
The Server object represents the internal state of the application server. It stores common configuration properties that are shared among the applications running on the application server. For example, the email of the server administrator may be stored in the Server object.

Error
Each application page has an Error object that represents all the errors that occurred during the interpretation of the application page by the server.

Table 1 maps the various objects in our common object model and the built-in objects in the JSP, NSP, ASP, and ColdFusion frameworks. Using this mapping we are able to easily migrate across the frameworks' built-in objects, as we explain later.

3 THE MIGRATION PROCESS

This section presents our process for migrating web applications to a new framework. As described in the previous sections, web applications contain two types of pages: active pages, and static pages. Static pages do not contain any control code; thus they are framework independent and aren't processed during our migration process. Instead we focus on migrating active pages, which contain control code that is framework or language dependent.

It is common for the migrated code to undergo repeated change: many new features and enhancements are added to it. Thus, an automated migration process which produces unreadable code with comments stripped out of the original application isn't an acceptable solution as the migrated application needs to be manually maintained and evolved under the new framework.

Furthermore, the user interface of the application, which is defined using the HTML code intermixed with the active code, should not be altered as the application is migrated to a new framework. The users of the application should not notice user interface changes. Consequently, we defined two main objectives for our migration process:

1. In the migrated application, the HTML code should remain in the same locations relative to the control code in the original application. The user interface should not be affected by the use of a different programming language or a development framework.
2. In the migrated application, code comments should remain in the same locations relative to the original source code so developers can still easily maintain the migrated application manually.

Island Grammars
The intermixing of control code, comments inside the control code, and HTML in an active page tremendously increases the complexity of parsing such pages. In our previous work [10], we concentrated on visualizing web applications. We dealt with the existence of multiple sections written in different programming languages inside of a single source file by choosing to extract only the entities which we are interested in visualizing. By considering each processed file as an ocean of tokens, a set of extractors were developed using grammars for each island of interest (control code islands, HTML islands, etc.). Each extractor processes the source file and locates the subsections (islands) of interest in the file. Once these are located, the appropriate parse is performed to extract the information. For example if we define the HTML code as being our interested island, then all the control code tokens become water that is ignored by our HTML parser. Looking at Figure 1, that would mean lines 1, 3, 5, 7, and 9 are islands and the other lines are water.

As the size of an island grammar is much smaller than a full language grammar then the parsing done using Island grammars is much faster. As well, Island grammars are more robust to changes in the language or to simple syntax errors as they only specify small sections of the language in detail and not the whole language. Furthermore, a full knowledge of the parse tree is not required.

Definition: An **Island Grammar** is a grammar which contains two types of productions: a) productions which describe constructs of interest (*Islands*), and b) liberal productions which catch the remaining uninteresting sections (*Water*).

Formally, for a context free language (CFL) L_G, a context free grammar (CFG) G, such that $L(G) = L_G$ is defined as $G = (V, \Sigma, S, P)$, where

1. V and Σ are finite sets with $V \cap \Sigma = \emptyset$: V is the set of variables or nonterminal symbols, and Σ is the alphabet of terminal symbols (terminals)
2. $S \in V$ is the $start$ symbol; and
3. P is a finite set of $productions$ or grammar rules of the form $A \rightarrow \alpha$, where $A \in V$ and $\alpha \in (V \cup \Sigma)^*$.

We define an *Island Grammar* $G_I = (V_I, \Sigma_I, S_I, P_I)$ as a set of constructs of interest $I \subseteq \Sigma^*$ such that $\forall\ i \in I$, S

Common Objects	ASP Objects	NSP Objects	JSP Objects	CF Objects
Request	Request	Request	javax.servlet.ServletRequest	<CFHttpParam>
Response	Response	_[1]	javax.servlet.ServletResponse javax.servlet.jsp.JspWriter	<CFOutput>
Session	Session	Client	javax.servlet.http.HttpSession	<CFApplication> <CFCookie>
Application	Application	Project	javax.servlet.ServletContext	<CFApplication>
Server	Server	Server	javax.servlet.ServletContext javax.servlet.ServletConfig	<CFRegistry>
Error	ASPError	Ssjs_onError	java.lang.Throwable	<CFError>

1. NSP does not encapsulate Response related methods in their own object; instead they reside in the global namespace.

Table 1: ASP, NSP, JSP and CF Object Models Mapped to the Common Web Object Model

$\Rightarrow^* s_1 i s_2$, where s_1 and $s_2 \in \Sigma$ and $s_1 i s_2 \in L_G$. This definition ensures that the island grammar is more liberal than the original grammar $i.e.$ $L(G) \subseteq L(G_I)$. Therefore G_I can correctly parse text that is not valid under G. As the input of the migration process is valid under G we do not have to be concerned about this fact.

Water Transformations

Whereas *Island Grammars* are used to help in lightweight understanding of large code bases, they are not capable of performing powerful language transformations/migrations. As the generated parse tree is restricted to the island (areas of interests), the transformations can only be done on simple/local areas inside in the islands of interests [15]. In this paper, we tackle the problem of migrating web applications. To perform the migration we simply need to migrate the control code written in languages such as VBScript, or JavaScript; while keeping the rest of the source file intact and the locations of HTML, code comments fixed relative to the control code. To accomplish this task, we identified three different alternatives:

1. Remove all the HTML and control code comments from the active page, perform the language/framework migrations, then re-inserts the HTML and code comments. In [3] the authors propose this approach to handle comments during the migration of a legacy COBOL system. Their approach removes the comment from the legacy code, performs the migration, then re-insert comments using a modified diff algorithm which compares the pre-migration code and the post-migration code; and attempts to re-insert the comments in the appropriate locations. We chose not to use this approach because the extensive intermixing between HTML code, control code, and comments in a single active page file would increase the complexity of the proposed diff algorithm.
2. An alternative approach is to extend the grammar of the control code language to support intermixing HTML code, comments, and control code. For example to migrate from JavaScript to VBScript, we would extend the JavaScript and VBScript language grammars to support the embedding of HTML code and comments in the generated parse tree. Furthermore, we would need to have rules to specify how these new comment and HTML tokens are processed during the language transformation. Certainly, the complexity of extending each programming language grammar to support such intermixing is too high since it would require a good understanding of the grammars of each language, and good experience in crafting such complex grammars. Furthermore this would increase the complexity of the grammars which would slow down the migration process. Therefore we decided not to adopt this approach.
3. Our last alternative and the one we adopt is *Water Transformations*. The approach is a more light weight technique to migrate multi-lingual source code bases. The complexity of developing the transformation technique is much simpler than the other two described approaches. By unifying the various languages into one language, the approach reduces the complexity of the migration process.

We now explain *Water Transformations*. Given a file that has a large intermixing of different languages (comments, HTML, and control code), we define an *Island Language*. The island language is the language we are interested in migrating. For our case that would be the control code lan-

guages, such as JavaScript, or VBScript. Following the same analogy as *Island Grammars*, the rest of the tokens are considered as *Water*. We now define *Water Transformations*, which are code transformations that convert water tokens to special islands. These islands follow the syntax of the island language (in our case the control code language). Once we apply these *Water Transformations* on the input file, the result is a valid file in the language of the island with no Water (uninteresting) tokens.

We then perform the migration using the grammar of the island language. The migration can be performed by any technique described in the extensive literature of language migration.

After the migration is performed, the inverse of the water transformations is applied on the output to revert the special islands back to water. The final output is a migrated file with the location of the HTML code and source comments left constant relative to the location of the newly migrated control code.

Definition: We define a **Water Transformation** T_W, as the transformation that maps each set of consecutive tokens $(s_i)^*$ in the input file to A_j ($i.e.$ $(s_i)^* : \xrightarrow{T_W} A_j$), where $\forall i$, $(s_i)^* \not\subseteq L(G_I)$; $\forall j, A_j \subseteq L(G_I)$; and $j \in \{1, ..., n\}$, where n is the number of independent consecutive streams of tokens $(s_i)^*$ in the input file.

In simpler terms, each continuous stream of tokens that are not valid in the island language, $(s_i)^* \not\subseteq L(G_I)$ are mapped to A_j which is valid in the *island language*. For example, during the migration of an active page in ASP to NSP we can transform each stream of continuous HTML code into a call to a dummy function called `HTMLCALL()`. Whereas the HTML code is not valid in the VBScript language, `HTMLCALL()` is valid in the VBScript langauge.

Using *Water Transformation*, the migration of a software system can be written formally as:

$$T_W^{-1}(T_M(T_W(Input_File))),$$

where T_M is the migration transformation, which converts from one programming language to the other. T_W^{-1}, the *Inverse Water Transformation* is defined as $A_j : \xrightarrow{T_W^{-1}} T_M^{-1}(A_j) = (s_i)^*$. This formula can be extended to as many embedded languages as needed. A more general formula is:

$$T_{W_x}^{-1}(....T_{W_1}^{-1}(T_M(T_{W_1}(....(T_{W_x}((Input_File))))))).$$

In the following section we present a case study of the migration process and a more detailed example rather than the abstract examples presented in this section. We use the term *Preprocessing* to indicate the application of the *Water Transformations* and the term *Postprocessing* to indicate the application of the *Inverse Water Transformations*.

4 CASE STUDY: AN EXAMPLE OF THE MIGRATION PROCESS

Figure 3 shows an overview of the migration process. In this section, we show the migration of a simple application page to clarify the details presented in the previous section. More complex pages have been migrated but we use this simple example page to showcase the various stages in the migration process. The migration process is divided into four main stages:

1. The *Preprocessing* stage removes the HTML code and comments from the active page. These are set aside to be later inserted into the migrated file in the (*Postprocessing* stage). This stage corresponds to the `rmHTML` and `rmComment` boxes in Figure 3.
2. The *Language transformation* stage translates the active code from the initial programming language to the programming language used by the target web application framework. The translation is carried out by a program written in TXL [18].
3. The *Object model mapping* stage transforms access to objects in the original application framework to the corresponding objects in the target framework. For example, we would map *Response.Write* to *Write* to migrate from the ASP to the NSP frameworks.
4. The *Postprocessing* stage reinserts the HTML code and comments that were removed in the *Preprocessing* stage. This corresponds to the `addHTML` and `addComment` boxes in Figure 3.

We now discuss these stages in more detail.

Preprocessing Stage

During the *Preprocessing* stage the active page is processed by two `Perl` scripts:

1. The first `Perl` script (`rmHTML`) processes the ASP file and replaces contiguous sections of HTML code with a call to dummy function named `HTMLCALL()` with a numbered parameter to permit the reversal of this step after the migration. The removed HTML code is stored in another file in an XML format. This XML file and the numbering is used later to reconstruct the full NSP file. Figure 4 shows the result of this action when processing the active page shown in Figure 1.

```
1: HTMLCALL(1);
2: If Session("LoggedIn") Then 'User already logged in
3: HTMLCALL(2);
4: Response.Write(Session("username"))
5: HTMLCALL(3);
6: Else 'User needs to login
7: HTMLCALL(4);
8: End If
9: HTMLCALL(5);
```

Figure 4: Example ASP File After `rmHTML`

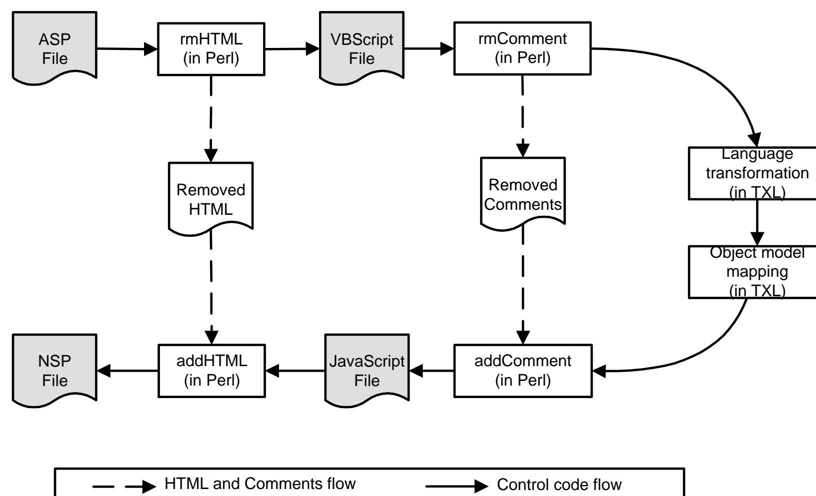

Figure 3: Migration Process

2. The second Perl script (rmComment) processes the output of the first script; it removes each line of comments in the VBScript code and replaces it with a call to dummy function named COMMENTCALL() with a numbered parameter. Furthermore, it performs some massaging of the source code to reduce the complexity of the grammar used in the following step to parse the VBScript file. For example, it adds a semicolon at the end of each line of VBScript and breaks all statements that are separated with a ":" into separate lines[1]. Figure 4, shows the results after running the rmComment script.

```
1: HTMLCALL(1);
2: If Session("LoggedIn") Then; COMMENTCALL(1);
3: HTMLCALL(2);
4: Response.Write(Session("username"));
5: HTMLCALL(3);
6: Else; COMMENTCALL(2);
7: HTMLCALL(4);
8: End If ;
9: HTMLCALL(5);
```

Figure 5: ASP File After rmComment

At the end of the *Preprocessing* stage, the output file contains a valid VBScript program with no comments and with each line terminated by a semi-colon as shown in Figure 5.

As our migration process is performed on each file separately and code changes are only reflected locally in the processed file/code section, our migration process is simple and scalable. The location of code segments is not susceptible to large changes during migration, thus the location of HTML and comments remains constant relative to the active migrated code.

Language Transformation and Object Model Mapping Stages
Once the *Preprocessing* stage is completed, the output is processed by two programs written in the TXL [18, 3] programming language. The TXL programming language is a hybrid functional/rule-based language with unification, implied iteration and deep pattern matching. It is useful for structural analysis and transformation of formal notations such as programming languages, specification languages, structured documents. It was used extensively during the Y2K analysis of COBOL/PL1 programs to repair date problems [5, 2]. The TXL language provides high level constructs to:

1. Build parse trees of an input program, based on BNF specification.
2. Transform the parse trees using rules which are specified using the tree structure and textual patterns.
3. Re-emit the transformed parse trees.

```
1: HTMLCALL(1);
2: If (Client.LoggedIn) { COMMENTCALL(1);
3: HTMLCALL(2);
4: Write(Client.username);
5: HTMLCALL(3);
6: } else { COMMENTCALL(2);
7: HTMLCALL(4);
8: }
9: HTMLCALL(5);
```

Figure 6: ASP File After TXL Processing

Figure 6 shows the results of the two TXL programs:

[1] VBscript permits multiple statements on the same line if they are separated by ":"

1. The *language transformation* program translates one language to another, for example from VBScript code into JavaScript code. Due to the variation in the type systems of the web framework programming languages, special processing may be needed. For example, VBScript is a weakly typed language with only one data type, while JavaScript is a weakly typed language which has multiple data types. In the general case, our migration process would need to determine the data types of the converted variables. Luckily, the JavaScript interpreter can determine the type of a variable at run-time based on its content. Thus, we do not need to perform type inference analysis to determine the types of variables. Instead the determination of the data types is left to the interpreter at run-time. Alternatively, to migrate an application which uses the ASP framework to one that uses the JSP framework, we need to determine the type of each variable as JSP uses a strongly typed language (Java). A type inference analysis is required. From our experience in studying web applications [9, 10, 11], we noticed that a large percentage of variables used in active pages are of type `String`. During the migration to a strictly typed language, the migration process may simply assume that all variables are of type `String` as it is the most used variable type in web applications. The developer will need to examine the result of the migration to JSP and correct the output.
2. The *object model mapping* program transforms object references to built-in objects provided by the ASP framework to references to the corresponding built-in objects provided by the NSP framework (refer Section 2). For example, during migration of an ASP file to a JSP file, each reference to the `Request` object is replaced with a reference to `javax.servlet.ServletRequest` object. This `TXL` program specifies more accurately the mapping between the various frameworks - for example, it details the mapping at the level of method names and parameter order and type rather than just the mapping between the objects as shown in Table 1. In the cases where no appropriate mapping exists, a special token `<UNKNOWN>` is inserted and manual intervention is needed to correct the output; or the user may extend the object model mapping and define new mappings for their own objects. Our approach does not migrate user's objects, instead these objects can be migrated using various other techniques or can be wrapped and accessed remotely by the web application if possible.

Post-Processing Stage

This is the last step of the migration. Two `Perl` scripts process the output of the previous stage:

1. The `addComment` script reinserts source code comments that were removed by `rmComment` script in the pre-processing stage. It scans the input file and replaces each call to the place-holder function `COMMENTCALL()` with the corresponding comment line stored in the "*Removed Comment*" XML file.
2. The `addHTML` script reinserts the HTML code that was removed by `rmHTML` script in the pre-processing stage. Calls to the place-holder function `HTMLCALL()` are replaced with the corresponding HTML code.

Figure 7 shows the results of this last stage of the migration from ASP to NSP frameworks. The active page shown in Figure 7 for the NSP framework is equivalent to the active page shown in Figure 1 for the ASP framework.

```
1: <HTML> <TITLE>Main Page</TITLE>
2: <SERVER>If (Client.LoggedIn) { // User already logged in</SERVER>
3: Welcome
4: <SERVER> Write(Client.username) ; </SERVER>
5: to your account, to display balance
        <A HREF="balance.asp">click here</A>
6:  <SERVER> } else { 'User needs to login  </SERVER>
7: Please login first, <A HREF="login.asp">click here</A>
8: <SERVER> }  </SERVER>
9: </HTML>
```

Figure 7: Final NSP Active Page

5 RELATED WORK

A few methodologies/techniques have been proposed to migrate Enterprise Java Beans (EJB) application across servers developed by separate companies [12, 17]. The approaches are manual and are concerned with the variations between the EJB implementation of each application server. Our approach can be used to automate these proposed migration methodologies/techniques.

Tools have been proposed to automate or semi-automate the migration [4, 14]. These tools mostly focus on the migration of database access code in a web application. Our approach is not capable of processing complex variations in the database access code between the source and target framework. Instead we are only capable of migrating database access code that is done using built-in framework objects. Thus these approaches complement our work, in particular for applications that use complex database access techniques such as IBM's Net.Commerce.

6 OBSERVATIONS AND CONCLUSION

Web applications are tomorrow's legacy software. With short release cycles and a highly competitive market, the need for tools to assist developers in maintaining them is clear and pressing.

In this paper, we extend our previous work [9, 10, 11] on the visualization of web applications to perform migrations of web application to new frameworks. Researchers have recognized the need to adapt traditional software engineering principles to assist in the development of web applications. Instead of proposing a new paradigm for migrating

web applications, we aim to transform web applications to traditional single language software systems. Once transformed, we make use of the large literature of program migration and machinery which has been developed over the years. In our case, we used the `TXL` programming language which has been extensively used in the migration of large legacy traditional non-web based software systems.

We formalized our process through the introduction of the concept of *Water Transformations*. These transformations enable a light weight and formal process to convert a web application to a traditional single language system. *Water Transformations* and our migration process can be used more generally to migrate programming languages that are embedded inside other programming languages (for example to migrate embedded SQL to newer version of SQL) by using the ideas of preprocessing of the input file and the placement of place-holders to be later replaced in the *Postprocessing* stage after the transformations are done.

Furthermore, we present an approach to migrate between various object oriented frameworks. The approach uses traditional programming language transformation techniques to migrate from VBScript to JavaScript, and Object Model mapping based on a developed reference Object Model for web application frameworks (refer to Table 1). The approach automates error prone and low level migration details. Developers can concentrate on more interesting and complex problems in the migration process. Finally, the migration process ensures that the generated code is maintainable by preserving the structure of the migrated code and the relative position of comments in the generated code.

The presented approach has been used successfully to build a prototype tool to migrate the Hopper News application to NSP. The Hopper News application is a sample web application provided by Microsoft to showcase the ASP framework. It is described in detail in [10]. Other web application frameworks are being considered for migration to verify the generality of our approach. As each file is migrated separately without depending on the migration of other files in the application, the approach is quite scalable. The processing time and complexity of the migration process are limited by the time needed to migrate the most complex active page in the application.

Using this approach, developers are no longer locked into their initial development framework. Instead they can migrate their web applications to the most appropriate framework that suits their current requirements and future demands.

REFERENCES

[1] Macromedia, 2002. Available online at: `http://www.macromedia.com/software/coldfusion/`.

[2] J. Cordy, T. D. andA.J. Malton, and K. Schneider. Software Engineering by Source Transformation - Experience with TXL. In *IEEE 1st International Workshop on Source Code Analysis and Manipulation*, pages 168–178, Florence, Italy, Nov. 2001.

[3] J. Cordy, T. Dean, A. Malton, and K. Schneider. Software Engineering by Source Transformation. *Special Issue on Source Code Analysis and Manipulation, Journal of Information and Software Technology*, Feb. 2002.

[4] O. Corp. *In2j : Automated tool for migrating Oracle PL/SQL into Java.* 2001. Available online at: `http://www.in2j.com`.

[5] T. Dean, J. Cordy, K. Schneider, and A. Malton. Experience Using Design Recovery Techniques to Transform Legacy Systems. In *IEEE International Conference on Software Maintenance*, pages 622–631, Florence, Italy, Nov. 2001.

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

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

[8] P. M. G. Mecca, P. Atzeni, and V. Crescenzi. The Araneus Guide to Web-Site Development - Araneus Project Working Report. AWR-1-99, University of Roma Tre, Mar. 1999. Available online at: `http://www.dia.uniroma3.it/Araneus/publications/AWR-1-99.ps`.

[9] A. E. Hassan. Architecture Recovery of Web Applications. Master's thesis, University of Waterloo, 2001. Available online at: `http://plg.uwaterloo.ca/~aeehassa/home/pubs/msthesis.pdf`.

[10] A. E. Hassan and R. C. Holt. Architecture Recovery of Web Applications. In *IEEE 24th International Conference on Software Engineering*, Orlando, Florida, USA, May 2002.

[11] A. E. Hassan and R. C. Holt. A Visual Architectural Approach to Maintaining Web Applications. *Annals of Software Engineering- Special Volume on Software Visualization*, 16, 2003.

[12] S. iPlanet. *Migration Guide, iPlanet Application Server*. 2000. Available online at: `http://docs.sun.com/source/816-5774-10/mpreface.htm`.

[13] Island Grammars, 2001. Available online at: `http://losser.st-lab.cs.uu.nl/~visser/cgi-bin/twiki/view/Transform/IslandGrammars`.

[14] T. C. Lau, J. Lu, E. Hedges, and E. X. Xing. Migrating e-commerce database applications to an enterprise java environment. In *IBM Centre for Advanced Studies Conference (CASCON)*, Toronto, Canada, 2001. Available online at: `http://www.cas.ibm.com/archives/2001/papers/cascon01/htm/english/abs/lau.htm`.

[15] L. Moonen. Generating robust parsers using island grammars. In *Working Conference on Reverse Engineering*, Stuttgart, Germany, 2001.

[16] L. Moonen. Lightweight impact analysis using island grammars. In *10th International Workshop on Program Comprehension*, Paris, France, 2002.

[17] T. Research. *Moving from IBM WebSphere 3 to BEA WebLogic Server 5.1, White Paper*. Available online at: `http://www.bea.com/products/weblogic/server/Migration_WP.pdf`.

[18] The TXL Transformation System, 2001. Available online at: `http://www.txl.ca/`.