NOSCO-STOM을 통한 프레젠테이션 레이어 프레임웍

Presentation Layer Framework using NOSCO-STOM

권 기 현*
KiHyeon Kwon

요 약

웹 애플리케이션을 개발하는데 있어서 가장 중요한 고려사항 중의 하나는 프레젠테이션과 비즈니스 로직을 효과적으로 분리하여 개발 생산성을 높이고 유지보수 비용을 낮추는데 있다. 기존의 애플리케이션 개발 기법으로 서블릿, JSP, ASP.NET 등의 스크립트 기반 기술에서부터 Struts, JSF(Java Server Faces), Spring MVC 같은 동적인 서버 페이지 개발 프레임워크가 있다. 이들 방법은 각기 다른 웹 티어(Web Tier) 처리 방법을 제공하나 프레젠테이션과 비즈니스 로직을 완벽하게 분리하지는 못하고 있다. 본 논문에서는 프레젠테이션과 비즈니스 로직을 완벽하게 분리하기위해 프레젠테이션과 비즈니스 로직을 분리 처리하는 커스텀 태그 처리 컴포넌트를 개발하고 새로운 동적 서버 페이지를 처리하기 위한 컨테이너(container)를 개발하였다. 그리고 DOM 트리를 개발한 컨테이너에 적용하여 프레젠테이션을 효율적으로 조작할 수 있도록 하였다. 끝으로, 개발 환경 구현을 통해 자동적으로 페이지 처리 컴포넌트 생성을 지원하는 시스템을 개발하였다.

Abstract

One of the most important factor while developing web application is to separate presentation and business logic lowering the maintenance cost. There are various web application development tools mainly categorized as script based such as Servlet, JSP, ASP.NET techniques and dynamic server page development frameworks such as Struts, JSF (Java Server Faces), Spring MVC etc. These tools provide web tier processing solution but not the complete separation of presentation and business logic. In this paper, we developed custom tag components that separate presentation and business logic, to process them we also developed container. In addition, DOM tree is applied to the developed container to manage the presentation effectively.

☞ Keyword : DOM, Dynamic Server Page, Presentation Framework

1. Introduction

Web has been playing an important role as an interface for all web application since the development of Internet, hence becoming an key factor for the web application's success both in terms of maintainability and popularity. Web application development is basically categorized as script based server page development techniques such as ASP, JSP, PHP etc. and web application development framework techniques like Struts [1], JSF (Jave Server Faces) [2], Spring MVC [3]. Apart from these, there are other many project under development or developed [4-6].

In script based server page development techniques, dynamic business logic code (ex. Java) and presentation logic (ex. HMTL) are complicatedly intermixed or contains the custom tags which are heterogeneous with design page code. These cause problem for debugging and maintainability. And the structural programming becomes difficult in script programming due to its sequential execution and output nature.

The framework based techniques like Struts, JSF, Spring MVC are based on MVC (Model

* 정회원 : 강원대학교 전자정보통신공학부 조교수
anyjava@empal.com
[2006/07/05 투고 - 2006/07/11 심사 - 2006/09/05 심사완료]

View Controller) Model 2. These techniques are developed to obtain the goal of separation of business and presentation logic, but lack the completeness.

ASP.NET's code behind method has opted new processing method, rather than following these kinds of old script based techniques. It uses DOM (Document Object Model) based code behind techniques improving server side development and maintainability with the separation of design and program code. But in case of Java, though many secure technologies like Velocity [8], Java Server Faces etc. are developed, they still lack the complete separation instincts since they are based on script based processing techniques like JSP.

In this paper, we have developed dynamic server page NOSCO-STOM based on DOM Model's code behind techniques. First, we developed custom tag components that process the separation of presentation and business logic. Second, container was developed for processing new dynamic server page. Third, DOM tree is applied to the developed container to manage the presentation effectively.

Section 2 outlines the related works and their pros and cons. Section 3 explains the proposed architecture and its design. We have presented implementation in section 4. Finally, we have compared our architecture with other frameworks in section 5 and summarized the conclusions in section 6.

2. Related Works and their problems

We discuss current available dynamic server page techniques and their methods of service, speciality and problems they are facing.

2.1. Problems of Script language

The script based server page techniques like Servlet, JSP, ASP, PHP etc. are presently available as web application development techniques. In these script based techniques, dynamic business logic code (ex. Java) and presentation logic (ex. HMTL) are complicatedly intermixed or contains the custom tags which are heterogeneous with design page code. For example, in case of JSP, dynamic business logic code (ex. Java) and presentation (ex. HTML) are complicatedly intermixed together. In this case, the page based coding and debugging becomes harder. And, the structural programming hardly becomes possible.

2.2. Problems of Web Application Framework

Struts[1] has made positioned itself as a web application development model based on HTML tag library. But, it takes more time while writing ActionForms and lacks the ability to test. Spring MVC [3] owns many similarity while handling view with JSP/JSTL, Tiles, Velocity, FreeMarker, Excel, XSL, PDF which is its weakness. WebWork [9] lacks validation of client side content processing though it has simple structure and extendibility. Tapestry [10] is simple in terms of structure due to HTML templates making ease to learn. Though JSF (Java Server Faces) [2,10], a J2EE [11] standard, provides easy development framework with detail navigation, but the connection is not that much easy due to excessive inclusion of JSP tag.

2.3 Design Strategies for Web Tier Service

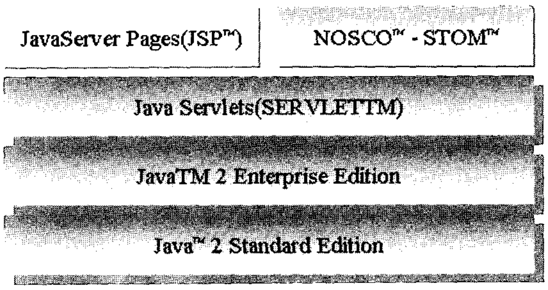
We adopted following strategies to solve the problems seen in sub section 2.3.

(1) Work on Servlet basis. NOSCO also gets inheritance from Servlet and works on Servlet basis like JSP.
(2) Remove the additional programming from design page code. It is necessary not to include server-side programming code, which is required for dynamic content creation, directly in design page code (Code Behind).
(3) Provide dynamic content creation code development framework. It provides API (STOM API) for creating dynamic content and accessing design page code.
(4) Parse the page content once at the time of first request and manage the part that requires dynamic processing as STOM tree object. The program code is executed in similar style as Servlet or Applet managing proper lifecycle (NOSCO bean). STOM tree object for dynamic processing is accessed as STOM API from developer implemented "NOSCO bean" and dynamic content is created. NOSCO-STOM provides the abstract classes for JSP, Servlet and Javabean making development and extension simple.

3. NOSCO-STOM Architecture

NOSCO (No JSP in COHALS) engine internally uses COHALS (Component Of Html And Logic Separation) components. COHALS component is used to support HTML Table tag in terms of JSTL (Java server pages Standard Tag Library). STOM (Simple Tag based document object Model) is the DOM tree based page processing technique(method) and is used in combination with NOSCO, hence becoming a new dynamic server page framework where NOSCO embeds STOM techniques.

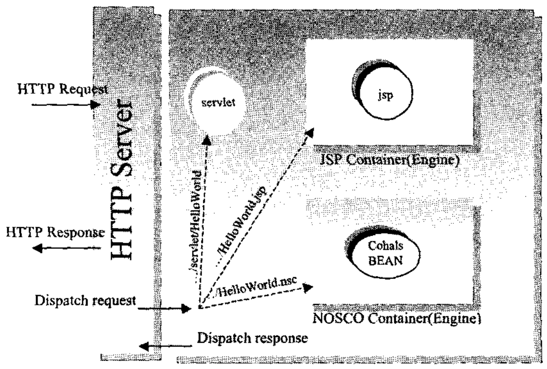
3.1. Technology Building Blocks



⟨Fig 1⟩ NOSCO-STOM Building Blocks

The building blocks for NOSCO-STOM is as in Fig. 1. Like JSP, it is also Servlet based technique and resides on the top together along with JSP, followed by Java Servlets. Then, there are J2EE (Java 2 Enterprise Edition) and J2SE (Java2 Standard Edition).

3.2. NOSCO Container



⟨Fig 2⟩ NOSCO Container (Engine)

NOSCO Container plays vital role in NOSCO-STOM architecture. It provides services being included in Servlet engine like JSP engine, though JSP comes along with Servlet as a single product, but NOSCO can come together or can be included used in all Servlet engines. As in Fig. 2, Servlet and JSP are called as /servlet/HelloWorld and /HelloWorld.jsp respectively where as NOSCO is as /HelloWorld.nsc. The following subsection 3.3 elaborates the life cycle of NOSCO and section 3.4 contains the processing procedure.

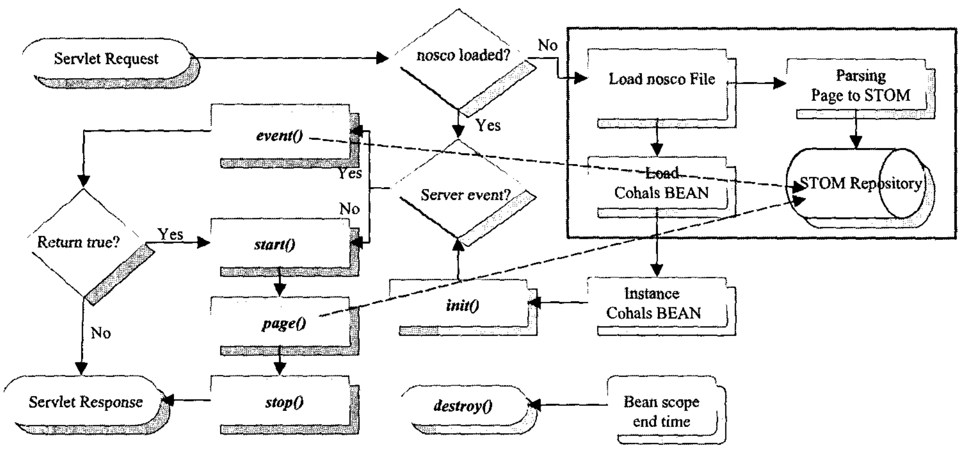
3.3. NOSCO Life Cycle

In NOSCO, Servlet and JSP are initialized for the initial request. after which can be used directly and the execution stops at reloading time. But JSP needs to be loaded, compiled and initialized for the initial request. Hence NOSCO life cycle provides service connecting NOSCO and STOM tightly, NOSCO file is uploaded according to the request and STOM is generated to provide the service. It serves instantly if NOSCO is already loaded, otherwise it needs to load NOSCO file, parse the web page and saves in STOM before serving. NOSCO bean is destroyed at the end of bean scope time.

3.4. NOSCO-STOM Execution Procedure

The execution sequence of NOSCO is shown in the Fig. 4.

(1) If the request URL pattern is *.nsc, then request comes through “NOSCO Servlet”.
(2) "NOSCO Servlet" analyzes request URL and makes NOSCO engine to execute the related URL.
(3) "NOSCO engine" gets the STOM tree of the page related to the request from STOM storage.
(4) If STOM tree couldn’t be obtained from "STOM storage", the request is sent to “page parser” to parse the requested page.
(4-1) "Page parse" looks for the requested page in local file system and parse it.



<Fig. 3> NOSCO Life Cycle

```
<chs:nsc xmlns:chs="http://www.cohals.co.kr/nosco/2006/XMLSchema">
   <chs:bean className="nsc.examples.NoscoMetaFileCHSB"
     scope="BEAN_SESSION" session="true">
        <chs:param name="foo2" value="bar2"/>
        <chs:param name="foo1" value="bar1"/>
   </chs:bean>
   <chs:page mimeType="text/html;charset=EUC-KR" chsIdName="chsid"
     pageCES="EUC-KR" path="NoscoMetaFile.html">
       <chs:trace name="header" value="REQUEST"/>
       <chs:trace name="parameter" value="NONE"/>
       <chs:trace name="stom_tree" value="BOTH"/>
    </chs:page>
</chs:nsc>
```

〈Listing 1〉 NOSCO Meta Information

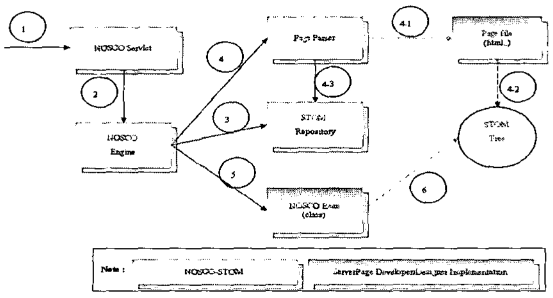
(4-2) "Page parse" creates the part which is going to be changed among parsed content as STOM tree object.

(4-3) "Page parser" saves the created STOM tree object in STOM storage.

(5) "NOSCO Engine" finds the NOSCO bean that processes the page dynamically and starts managing the life cycle of related bean. And, the related bean contains the "STOM tree" object that processes the bean.

(6) "NOSCO Bean" makes dynamic content using the allocated "STOM tree" object.

3.5. NOSCO Meta Information



〈Fig 4〉 Execution Procedure of NOSCO

If the page developed by web page designer is the tag based file that included HTML file, most of the job can be done. As in the Listing. 1, the meta information necessary to execute NOSCO is inserted in the file that has "nsc" extension. This work is handled by contents developer.

Listing 1. shows the meta information for executing NOSCO and has two sections : *<chs:bean/>* and *<chs:page/>*.

<chs:bean/> contains the property details of dynamic content processing Java class (NOSCO bean) like *className* property, *scope* property, *session* property. The scope property is similar to JSP's *<jsp:useBean/>* tag's scope property and the session resembles with session property of JSP page directive. The child tag *<chs:param/>* can be also used where the *name*, *value* properties get values anytime from the referenced NOSCO bean.

<chs:page/> contains the mimeType properties of the page. The page's file encoding is inserted in pageCES. If the meta information is in ".nsc" file, then the *path* property contains the path of that file. *<chs:trace/>* eases debug-

ging at the time of development.

Meta information defines the contents in any page that requires dynamic processing where as the dynamic content developer does using tags. The tag's property is set as *chsid*, an unique value in that page.

In case of Listing 2, the whole <BODY/> is defined as dynamic processing part and hence got *stomAssignment* as *chsid* value.

```
<HTML>
<BODY chsid="stomAssignment">
<center><BOOK LIST</center>
<table>
  <tr>
     <td>BOOK</td>
     <td>AUTHOR</td>
  </tr>
  <tr>
     <td>Hobbit</td>
     <td>Tolkien, John Ronald Reuel<.td>
  </tr>
</table>
</BODY>
</HTML>
```

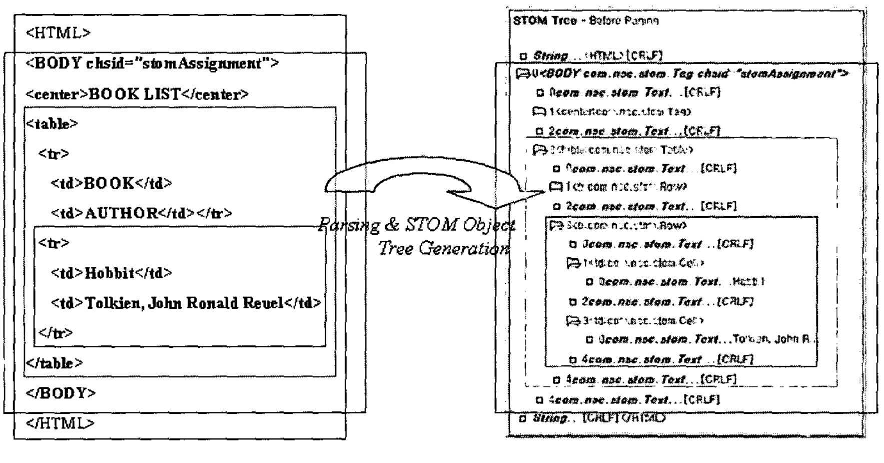
〈Listing 2〉 HMTL Assigning of Dynamic Processing Part

3.6. Transformation of nsc file into STOM Tree

The Fig. 5 shows how the HTML page defined parts are transformed into rectangular tree browser using NOSCO-STOM And, it also shows that HTML table are parsed and categorized into as tree structure. Page's Tag component is objectized as *com.nsc.stom.Tag* where as text elements are into *com.nsc.stom.Text.* The *<table/>*, *<tr/>* and *<td/>* components are objectized as *com.nsc.stom.Table*, *Row*, *Cell* class type making the tag's property easy to control.

3.7. Implementation of NOSCO bean that references STOM Tree

STOM tree is referenced by NOSCO bean. As in Listing 3, NOSCO bean is coded and scope of bean defined by scope property (request, session, application) of meta information. When the bean object is first called, *init()* is called where *start()* is called if that bean is called next time followed by *paging()* method.

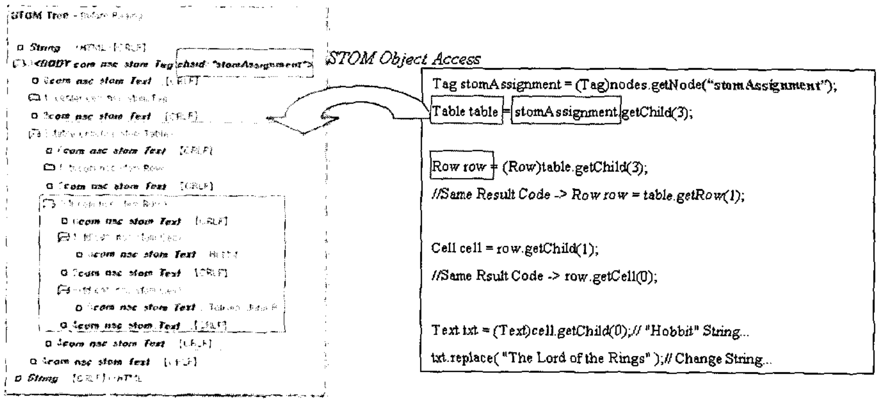


<HTML>
<BODY chsid="stomAssignment">
<center>BOOK LIST</center>
<table>
<tr>
<td>BOOK</td>
<td>AUTHOR</td></tr>
<tr>
<td>Hobbit</td>
<td>Tolkien, John Ronald Reuel</td>
</tr>
</table>
</BODY>
</HTML>
Parsing & STOM Object Tree Generation
STOM Tree - Before Paging
String ...<HTML>[CRLF]
0<BODY com.nsc.stom.Tag chsid="stomAssignment">
0com.nsc.stom.Text..[CRLF]
1<center com.nsc.stom.Tag>
2com.nsc.stom.Text...[CRLF]
3<table com.nsc.stom.Table>
0com.nsc.stom.Text...[CRLF]
1<tr com.nsc.stom.Row>
2com.nsc.stom.Text...[CRLF]
3<tr com.nsc.stom.Row>
0com.nsc.stom.Text...[CRLF]
1<td com.nsc.stom.Cell>
0com.nsc.stom.Text...Hobbit
2com.nsc.stom.Text...[CRLF]
3<td com.nsc.stom.Cell>
0com.nsc.stom.Text...Tolkien, John R.
4com.nsc.stom.Text...[CRLF]
4com.nsc.stom.Text...[CRLF]
4com.nsc.stom.Text...[CRLF]
String... [CRLF]</HTML>

〈Fig. 5〉 HTML and STOM Tree

```
//HttpCohalsBEAN ..
public class TestCHSB extends com.nsc.fx.HttpCohalsBEAN {
 //start()
  public void paging(com.nsc.fx.NodeList nodes) throws com.nsc.fx.CohalsException {
     com.nsc.stom.Tag stomAssignment=
                     (com.nsc.stom.Tag)nodes.getNode("stomAssignment");
     //..to do list.
  }
  public void init() throws com.nsc.fx.CohalsException{};
  public void start() throws com.nsc.fx.CohalsException{};
  public void stop() throws com.nsc.fx.CohalsException{};
  public void destroy() throws com.nsc.fx.CohalsException{};
```

〈Listing 3〉 NOSCO bean Code



```
Tag stomAssignment = (Tag)nodes.getNode("stomAssignment");
Table table = stomAssignment.getChild(3);

Row row = (Row)table.getChild(3);
//Same Result Code -> Row row = table.getRow(1);

Cell cell = row.getChild(1);
//Same Rsult Code -> row.getCell(0);

Text txt = (Text)cell.getChild(0);// "Hobbit" String...
txt.replace( "The Lord of the Rings" );// Change String...
```

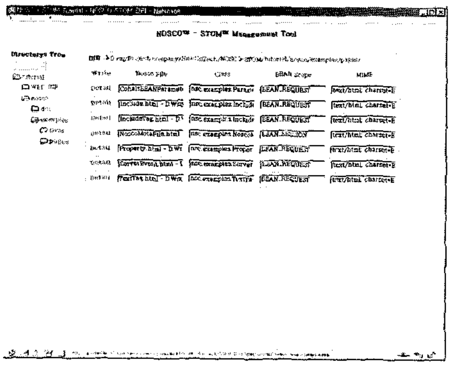
〈Fig 6〉 Dynamic Contents Processing using STOM tree

This method processes the dynamic contents. *stop()* called after completion of *paging()*. This is almost is similar to Servlet and Applet.

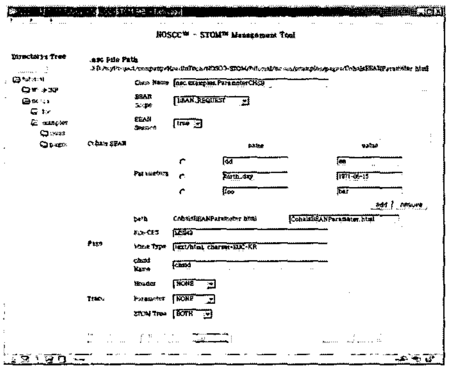
Fig. 6 shows how dynamic contents are processed in *paging()* using STOM tree. It shows how, the original page's "Hobbit" string is displaced by "The Lord of Rings".

4. Implementation of NOSCO-STOM Management Tool

NOSCO-STOM provides management tool for easing development of NOSCO. As in Fig. 7, the left side contains the directory tree. The directory gets link if it contains meta information file inside and after clicking on the directory we can get the list of those files as in Fig. 8.



〈Fig 7〉 NOSCO-STOM Management Tool


NOSCO™ - STOM™ Management Tool

Directory Tree

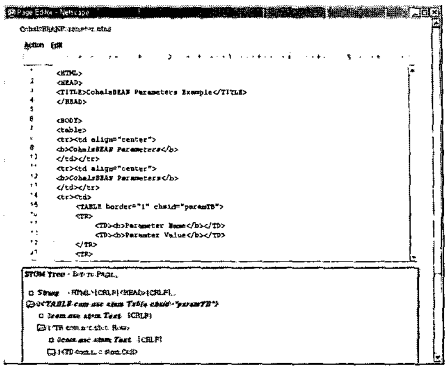
.asc File Path

Class Name	nsc.examples.ParameterCHS9
BEAN Scope	BEAN_REQUEST
BEAN Session	true

	name	value
Parameters	dd	ee
	birth_day	1971-06-15
	foo	bar

path	CobaltsBEANParameter.html
Mime Type	text/html; charset=EUC-KR
chsid Name	chsid
Header	NONE
Parameter	NONE
STOM Tree	BOTH

〈Fig 8〉 NOSCO Meta Information View



```
<HTML>
<HEAD>
<TITLE>CobaltsBEAN Parameters Example</TITLE>
</HEAD>

<BODY>
<table>
<tr><td align="center">
<b>CobaltsBEAN Parameters</b>
</td></tr>
<tr><td align="center">
<b>CobaltsBEAN Parameters</b>
</td></tr>
<tr><td>
    <TABLE border="1" chsid="paramTB">
    <TR>
        <TD><b>Parameter Name</b></TD>
        <TD><b>Parameter Value</b></TD>
    </TR>
    <TR>
```

STOM Tree - Before Page

```
String  <HTML>{CRLF}<HEAD>{CRLF}...
<TABLE com.asc.stom.Table chsid="paramTB">
  Item.asc.stom.Text {CRLF}
  <TR com.asc.stom.Row>
    Item.asc.stom.Text {CRLF}
    <TD com.asc.stom.Cell>
```

〈Fig 9〉 NOSCO Page Editor

It helps developer to modify meta information. Clicking on "Write" and designating the page in the server, NOSCO meta information can be input as in Fig. 7.

Clicking the "Run Editor" in figure 8. pops up simple editor page like in Fig. 9. The developer can editor the dynamic processing part from that editor page and can simultaneously view the STOM tree through Tree Browser. And, clicking on "Run NOSCO" in Fig. 8, the developer can execute the NOSCO file too where as "Generate Bean" makes the download of .java template that processes the page which is created by using "NOSCO meta information" and designated STOM tree.

5. Performance Evaluation of NOSCO

We have compared performance of NOSCO-STOM architecture with some of mostly used current tools. And the table 1. shows the results which clearly proved this techniques has upper hand in handling dynamic server pages.

In Table 1 we presented the key differences between our proposed architecture NOSCO-

〈Table 1〉 Property Comparison of Key Frameworks

Attribute	Tool				
	Struts	Spring MVC	WebWork	Tapestry	NOSCO-STOM
Role Separation	based on HTML tag library	based on tag library	tag library based, easy to customize	contrib:Table component	based on custom tag component, separates completely
Complexity & Maintainability	ActionForms are pain	Complex since lots XML, JSP code	Simple structure and ease to extend	document very conceptual	uses DOM and lessen the complexity and eases the maintainability
Testability	not sufficient, StrutTestCase	easy testing with Mocks	easy testing with mocks	difficult to test	similar like JSP, provides lots of examples
Validation	use common validator	use common validator	immature validation	robust validation	robust

STOM and other present day frameworks. It shows that NOSCO-STOM has certain benefits over the other frameworks in terms of role separation, complexity and maintainability, testability and validation of clients.

6. Conclusion

Many server side programs were developed to provide the server-side capability to web application. Although they played a major role in the explosion of Internet, their performance, scalability and usability issues make them less than the optimal solution. In this paper, we have presented a dynamic content generation code development framework based on page processing technique NOSCO bean and DOM tree model. This system also provided better and easier development of web application. NOSCO-STOM framework doesn't include directly server side program code (Java or custom tag) necessary for generating dynamic contents, rather it links the original design page and dynamic contents generation program code to execute readied dynamic contents generation program according to the page execution request.

Since the looks of first page doesn't change, it eases the maintainability, and the understanding between designer and programmer. In addition, programmer can focus only on software development and the development is possible normal Java environment. MVC model2 frameworks that uses JSP as page processing techniques normally owns complex processing method, but NOSCO-STOM provides simple, and easy development of web application. Hence, it completely separates the responsibility and speciality of designer and developer simplifying the development and lessening its development period.

References

[1] Apache Struts project official site. Available at http://struts.apache.org/index.html, 2006.

[2] H. Mahmoud, "Developing Web Applications with Java Server Faces", Available at http://java.sun.com/developer./technicalArticles/GUI/JavaServerFaces, 2004

[3] Javid Jamae, "Simplify Your Web App Development Using the Spring MVC Framework", Available at http://www.devx.com/Java/Article/22134/1954, October 11, 2004.

[4] A. Saimi, T. Syomura, H. Suganuma, I. Ishida, "Presentation Layer Framework of Web Application Systems with Server-Side Java Technology", Computer Software and Applications Conference, 473-478, October, 2000.

[5] M. Jacyntho, D. Schwabe, G. Rossi, "A Software Architecture for Structuring Complex Web Applications", Journal of Web Engineering, 37-60, 2002.

[6] N. Al-Darwish, "PageGen : An Effective Scheme for Dynamic Generation of Web Pages," Vol. 45, Issue 10, 15 July, 651~662, 2003.

[7] "ASP.NET Code Behind Model Overview," Available at http://support.microsoft.com /default.aspx?scid=kb;en-us;303247, Jan, 2004.

[8] "Velocity", Available at http://jakarta.apache.org/velocity/index.html, 2006.

[9] Neal Ford, "Art of Java Web development: WebWork", Available at http://www.javaworld.com/javaworld/jw-03-2004/jw-0329-webwork

_p.html, March 29, 2003.

[10] Phil Zoio, "JavaServer Faces vs Tapestry : A Head-to-Head Comparison", Available at http://www.theserverside.com/tt/articles/article.tss?l=JSFTapestry, August 2005.

[11] Bill Shannon, "Java 2 Platform, Enterprise Edition Specification Version: 1.2", Available at http://java.sun.com/javaee, Dec. 1999.

[12] S. H. Cheon, G. H. Kweon, H. J. Choi, "Developing an Automatic Component Creating System in Distributed Environment", Journal of the Korea Digital Content, Vol. 2, 2001.

[13] S. Chung, Y. S. Lee, "Modeling Web applications using Java and XML related technologies", Proceedings of the 36th Annual Hawaii International Conference on System Science, January, 2003.

[14] V. Apte, T. Hansen, P. Reeser, "Performance comparison of dynamic web platforms", Computer Communications, Vol. 26, Issue 8, 888~898, 20 May, 2003.

[15] R. Jan, C. Lin, M. Chern, "An optimization model for Web content adaptation", Computer Networks, Vol. 50, Issue 7, 953~965, 15 May 2006.

[16] W. Li, O. Po, W. Hsiung, K. S. Candan, D. Agrawal, "Freshness-driven adaptive caching for dynamic content Web sites", Data & Knowledge Engineering, Vol. 47, Issue 2, 269~296, November 2003.

[17] C. Leaguea, "MetaOCaml server pages: Web publishing as staged computation", Science of Computer Programming, Vol. 62, Issue 1, 66~84, September 2006.

◐ 저 자 소 개 ◑

권 기 현(KiHyeon Kwon)
1993년 강원대학교 전자계산학과 졸업(이학사)
1995년 강원대학교 대학원 컴퓨터과학과 졸업(이학석사)
2000년 강원대학교 대학원 컴퓨터과학과 졸업(이학박사)
1996~2002 동원대학 인터넷정보과 교수
2002~현재 강원대학교 전자정보통신공학부 교수
관심분야 : 미들웨어, 임베디드소프트웨어, 무선센서네트워크
E-mail : anyjava@empal.com