문서중심 및 웹기반 노심설계 자동화 시스템 개발

박 용 수* · 김 종 경**

Development of a Document-Oriented and Web-Based Nuclear Design Automation System

Yong Soo Park* · Jong Kyung Kim**

Abstract

The nuclear design analysis requires time-consuming and erroneous model-input preparation, code run, output analysis and quality assurance process. To reduce human effort and improve design quality and productivity, Innovative Design Processor (IDP) is being developed. Two basic principles of IDP are the document-oriented design and the web-based design. The document-oriented design is that, if the designer writes a design document called active document and feeds it to a special program, the final document with complete analysis, table and plots is made automatically. The active documents can be written with Microsoft Word or created automatically on the web, which is another framework of IDP. Using the proper mix-up of server side and client side programming under the LAMP (Linux/Apache/MySQL/PHP) environment, the design process on the web is modeled as a design wizard style so that even a novice designer makes the design document easily. This automation using the IDP is now being implemented for all the reload design of Korea Standard Nuclear Power Plant (KSNP) type PWRs. The introduction of this process will allow large reduction in all reload design efforts of KSNP and provide a platform for design and R&D tasks of KNFC.

Keywords : Nuclear Design Automation, Document-oriented Design, Active Document, Web, IDP

논문접수일 : 2004년 3월 9일 논문게재확정일 : 2004년 9월 12일

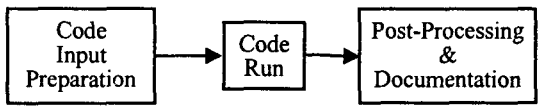
* 한전원자력연료㈜ 노심설계처 선임연구원
** 한양대학교 원자력공학과 교수

1. Introduction

The nuclear design is one of the several key design aspects that ensure nuclear fuel design limits will not be exceeded during normal operation of nuclear power plant. It requires the complex three-dimensional core nodal models and several computer codes. Much time and effort is spent on preparing accurate model inputs, running codes and processing the output. Many trials, therefore, have been made to reduce human effort and improve design quality and productivity. The conventional method is to make automation programs written in high level languages under X window system. C++ with Motif toolkit is popular for this purpose.

No matter what languages were used, the main idea is always *code-oriented*. The code-oriented design is that the designer prepares computer code input and generates the pre-fixed document using an automation program (See <Figure 1>). This approach seems to be suitable to normal reload design, in which little changes in design procedures are made. But it is also true that this approach deprives designer of the right to keep up with the design change. If any design changes happen, the automation program should be modified by programmer, not designer. Moreover, since the computing environment is so rapidly changing that software made today may be outdated easily, the cross-platform or platform-independent software is highly recommended. Therefore, a new paradigm to meet this claim is needed, too.

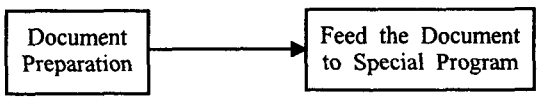
Korea Nuclear Fuel Company (KNFC) is developing an innovative design automation system called Innovative Design Processor (IDP). IDP is a system that enables designer to do all the nuclear design works with convenience. Two basic principles of IDP are the document-oriented design and the web-based design.



<Figure 1> Code-Oriented Design

2. Document-Oriented Design

The document-oriented or document-centered design is that the designer focuses on preparing design document, rather than computer code input. In other words, if the designer prepares a design document and feeds it to a special program, he/she can get the final document with complete analysis, tables and plots automatically (See <Figure 2>). This greatly relieves designer from the burden of documentation after computation. Because the designer can edit the document, he/she can take care of any design change without programmer's support. The document is not usual formatted text and called *active document* [Robert Spinrad, 1988].



<Figure 2> Document-Oriented Design

2.1 Active Document

The evolution of Information Technology has changed the way we think about documents.

The active document is one of them. Each time the document is accessed by a user or a program, it can be seen a different way and it can present a different content depending on the computing environment. In fact, an active document can be defined as a document that transforms itself and/or its computing environment according to the state of that environment and to the document editing operations. Applications that are embodied as active documents may be called *active document applications*. Such applications may be easier to build than their traditional counterparts because they can take advantage of the capabilities of a document editor [Douglas B. Terry and Donald G. Baker, 1990].

As a document editor, Interleaf [Paul M. English and Raman Tenneti, 1994] is an excellent platform for the design, implementation, and delivery of active documents. The main structure of Interleaf is called a *component*, which contains content as well as formatting information. For example, in the Interleaf representation of this paper, the current paragraph is a *para* component. Because Interleaf allows documents to be active, it is possible to build complete applications using *nothing but documents*.

Since the mid-1990s Westinghouse Electric Company (WEC) has developed the Physics Assessment Checklist (PAC) [J. A. Brown, et al, 1997] methodology, which correlates a small set of simple screening parameters to the more extensive reactor physics parameter set used in the plant safety analyses. In order to do PAC assessment easily, WEC developed an automation tool called Active Procedure Toolkit (APtk) [M. P. Rubin and S. G. Wagner, 1999] and Interleaf's active document, which has such design actions defined by *active components* as code input preparation, code run and output summary with table and figure. The active document is processed by Interleaf parser in APtk until the final document is made. Although APtk with Interleaf is very effective design automation tool, it has serious drawbacks as follows :

- Active document is Interleaf's unique ASCII format which can be edited by Interleaf only.
- Interleaf is old-fashioned and not suitable to modern computing environment.
- Interleaf is available no more in software market.
- APtk is written in UNIX C-shell and gawk scripts, which are hard to write a robust parser to parse active document in other popular formats.

Taking advantage of APtk, KNFC has developed a new active document processing system. The system eliminates APtk's drawbacks by adopting active document written in more popular markup language and creating a robust parser written in the more powerful language.

2.2 HTML

Hyper Text Markup Language (HTML) [Chuck Musciano and Bill Kennedy, 2002] has been the

```
<html>
<head>
  <title>IDP Test</title>
  <meta name = "keywords" content = "IDP, final parameter" />
  <meta name = "description" content = "IDP - Final Parameter Cal." />
  <link rel = "File - List" href = "./IDPtest.files/filelist. xml" />
  <link rel = "stylesheet" type = "text/css" href = "http://koyu/css/IDP.css" />
</head>
<body>
  <pre class = "APformula - hide">
  docnumber = "foo"
  num = 5
  </pre>
  <p class = "MsoNormal">Appendix to #docnumber# </p>
  <br class = "PageBreak" clear = "all" />
  <pre class = "APif"> num > 3 </pre>
    <pre class = "APfor"> INDEX=1 ; INDEX<= num ; INDEX++</pre>
      <pre class = "APsubmit-hide"> gnuplot < plot.inp </pre>
      <pre class = "APimage">plot.eps 1 2.3 4 4.2 A4 </pre>
      <br class = "PageBreak" clear = "all" />
    <pre class = "APendfor">End of APfor loop </pre>
 <pre class = "APendif">Bottom of APif Block</pre>
</body>
</html>
```

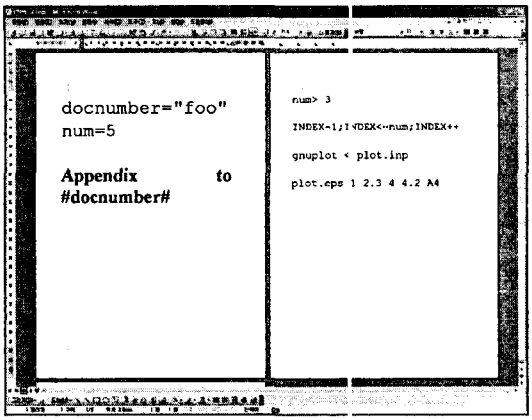
<Figure 3> Active Document in HTML Format

central markup language since the advent of web. Although HTML is not so suitable to document publishing, it can represent a document well if it is written with Cascading Style Sheets (CSS) [Eric Meyer, 2000]. With CSS, the document contents can be separated from style, and active component can be represented by the concept of *class*. Therefore, the active document can be written in HTML and CSS. <Figure 3> is a sample active document. A simple parser for an HTML document with the consistent structure can be made relatively easily. However, parsing non-structured HTML document (See <Figure 4>) requires a robust parser like event-driven or tree-based parser. A robust parser is written in Perl language to parse any HTML document. The parser is based on the Perl's HTML::Parser library and created by defining some callback functions suitable to the library.

```
<html><head><title>IDP Test</title><meta name
= "keywords"" content = "IDP, final parameter" />
<meta name = "description" content = "IDP - Final
Parameter Cal." /><link rel = "File-List" href = "./
IDPtest.files/filelist.xml" /><link rel = "stylesheet"
type = "text/css" href = "http://koyu/css/IDP.css" />
</head>
<body><pre class = "APformula-hide">docnumber
= "foo"
num = 5</pre><p class = "MsoNormal">Appendix
to #docnumber#</p>
<br class = "PageBreak" clear = "all" /><pre class =
"APif"> num > 3 </pre><pre class = "APfor">
INDEX = 1 ; INDEX<= num ; INDEX++</pre>
<pre class = "APsubmit - hide">gnuplot < plot.inp
</pre>
<pre class = "APimage">plot.eps 1 2.3 4 4.2 A4
</pre><br class = "PageBreak" clear = "all" /><pre
class = "APendfor">End of APfor loop</pre><pre
class = "APendif">Bottom of APif Block</pre>
</body></html>Bottom of APif Block</pre>
```

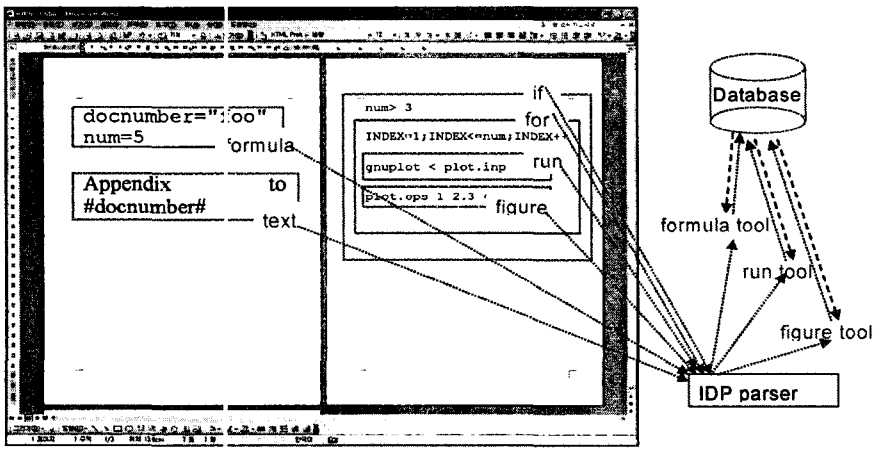
<Figure 4> Non-structured HTML Document

The active document can be created by simple notepad or HTML editor. MS Word can be used to create an active document, although it is not so good HTML editor. With MS Word, user can make an active document freely under WYSIWYG (What You See Is What You Get) environment and insert an active component by MS Word's *style* function. This helps *round-trip* the document for editing purpose. For example, if active document is created in MS Word and saved as HTML, the document can be re-opened in MS Word without any loss in style and contents. <Figure 5> is the active document opened by MS Word.

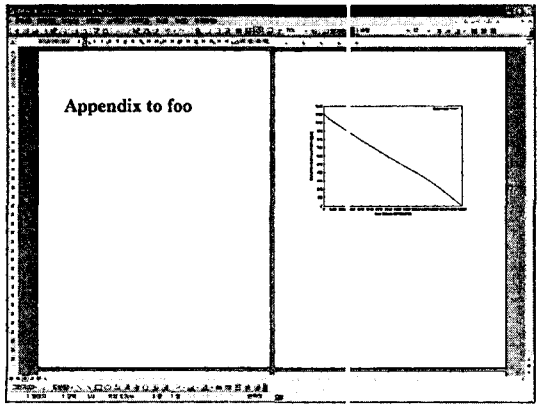


〈Figure 5〉 HTML Document Opened by MS Word

The created HTML document is processed by the HTML parser in IDP. In As shown in <Figure 6>, variable assignments are made by *APformula-hide* component, variable enclosed by # characters is replaced by its value, variable *num* is checked by *APif* component, a loop is constructed by *APfor* component, and a plotting program is executed until the loop is finished. The IDP handler controls whole process and all the variables with the associative array format are stored in the simple text-based database. <Figure 7> is the resulting active document opened by MS Word.



〈Figure 6〉 HTML Document Processing Architecture



〈Figure 7〉 The Final HTML Document

2.3 XML

As it was mentioned in the previous section, HTML is not the best solution to document publishing because HTML is intended mainly for display instead of print. That's why Extensible Markup Language (XML) [W3C, 2000] was created. XML is a simple, very flexible text format which is originally designed to meet the challenges of large-scale electronic publishing. XML is also playing an increasingly important

role in the exchange of a wide variety of data on the web and elsewhere.

XML is chosen as the main markup language of active document in IDP. However, considering the current transitional situation that HTML is more popular than XML, active document in HTML format should be maintained, too. Keeping active components in HTML format alive, all active components are redefined as XML tags. One of the advantages of XML is that user can create as many tags as he/she wants (See <Table 1>). With XML tags the active document looks more logical, leaner and legible because XML tags are self-explanatory. <Figure 8> lists the XML active document that is equivalent to HTML one in <Figure 3>. Although the Extensible Stylesheet Language (XSL) [Doug Tidwell, 2001] is usually used in XML document, CSS can be used instead. Note that XML tags and ordinary HTML tags are mixed in the active document. It shows that active document can be migrated to pure XML anytime without tears.

In order to parse an XML document, Perl's XML::Parser or XML::LibXML library can be used. However, HTML::Parser library can also be used if XML mode of the library is turned on as shown in <Figure 9>. Although XML mode of the HTML::Parser library was introduced to parse an XHTML document, it can also works well with generally well-formed XML document.

After an XML active document is created with simple notepad or any XML editor, the robust parser parses the document and translates it into equivalent HTML one, and the same procedure as HTML case is processed. Therefore, the resulting document becomes the same HTML document illustrated in <Figure 7>.

<Table 1> XML Active Components in IDP

Name	Functions
cdrom	Give a list of cdrom files from code run
edit	Edit variable values in tabular or map form
extract	Extract data from code outputs and store as variables
figure	Incorporate image (Postscript or PNG)
for	Means for looping using C-like syntax
formula	Set or calculate variables using formulas
if	Means for conditional branching
paste	Incorporate simple text files into the output document
read	Read data from files, or text databases
replace	Do variable replacement on external ASCII files
run	Execute shell scripts, including computer code jobs
?	Execute PHP commands

```
<?xml version = "1.0"?>
<doc>
 <head>
    <title>IDP Test</title>
    <keywords>IDP, final parameter</keywords>
    <descrip>IDP - Final Parameter Cal.</descrip>
    <link rel = "File - List" href = "./IDPtest.files/
    filelist.xml" />
    <?xml-stylesheet type = "text/css" href = "http:
    //koyu/css/IDP.css"?>
 </head>
 <body>
  <formula type = "hidden">
  docnumber = "foo"
  num = 5
  </formula>
  <p class = "MsoNormal">Appendix to #docnumber#
  </p>
  <pagebreak />
  <if condition = "num > 3">
    <for condition = "INDEX = 1 ; INDEX<= num ;
    INDEX++">
       <run type = "hidden"> gnuplot < plot.inp
       </run>
       <figure src = "plot.eps" margin-left = "1"
       margin-right = "2.3" margin-top = "4"
       margin-bottom = "4.2" paper = "A4" />
       <pagebreak />
    </for>
  </if>
 </body>
</doc>
```

<Figure 8> Active Document in XML Format

```
#!/usr/bin/perl -w
use strict ;
use HTML::Parser ;

my $p = HTML::Parser->new( api_version => 3,
          xml_mode     => 1,
          declaration_h => [\&declaration, "text"],
          start_h      => [\&start, "tagname, attr,
                           attrseq, text"],
          text_h       => [\&text, "text"],
          comment_h    => ["comment, "text"],
          end_h        => [\&end, "tagname, text"],
          process_h    => [\&process, "tagname"],
          default_h    => [sub { print shift }, "text"],
                         ) ;
```

〈Figure 9〉 Example of the Use of HTML::Parser Library

2.4 PHP and More

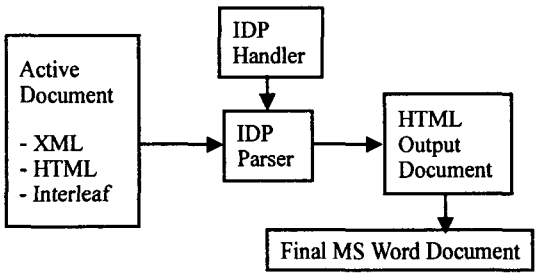
PHP : Hypertext Preprocessor (PHP) [Rasmus Lerdorf and Kevin Tatroe, 2002] is a widely-used general-purpose scripting language that is especially suited for web development and can be embedded into HTML. It can do not only server-side scripting, but also command-line scripting, client-side GUI application writing and even scientific applications [Volker Goebbels, 2003].

One of the strongest and most significant features in PHP is its support for a wide range of databases. Therefore, PHP tag is introduced to IDP as a new active component and IDP's parser is modified to handle PHP. With PHP handling, it is possible that data in database or any document fragment in any network can be inserted into active document easily, making active document more *active*. <Figure 10> is a sample active document that uses some PHP commands to add an HTML table from MySQL [George Reese, 2003] database.

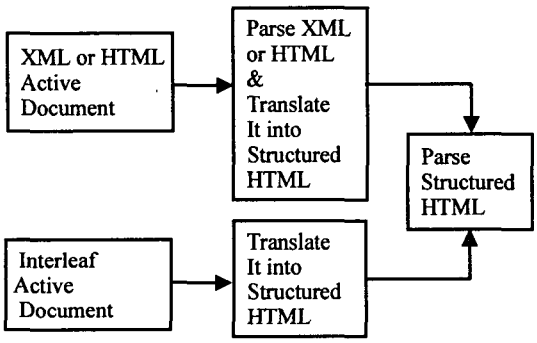
```
<?xml version = "1.0"?>
<doc>
  <head>
  <link rel = "File-List" href = "./IDPtest.files/filelist.
  xml" />
    <?xml-stylesheet type = "text/css" href = "http:
    //koyu/css/IDP.css"?>
 </head>
 <body>
   <formula>PHPtableRequired = 1</formula>
   <if condition = "PHPtableRequired">
    <? // insert a table to show the status of ongoing
    projects
        include "/users/www/htdocs/budget/inc/
        connect.inc" ;
        if(!$year) $year = date("Y") ;
        if(!$month) $month = date("m") ;
        $depts = array ("KHNP", "MOST", "MOCIE",
        "KNFC") ;
        echo "<table>\n<tr>\n" ;
        foreach ($depts as $dp) {
                $i = 0 ;
                $sql = "SELECT * FROM $atype_
                table WHERE acode = '$project_
                code' AND tyear = '$year' ORDER
                BY no" ;
                $result = mysql_query($sql) ; //
                number of total projects
                while($data = mysql_fetch_array
                ($result)) {
                  if ($data['dept'] == $dp) {
                    $tcode[$dp][++$i] = $data
                    ['tcode'] ;
                    $tname[$dp][$i] = $data
                    ['tname'] ;
                    $class[$dp][$i] =
                    ($data['class']) ? ($data
                    ['class']) : "  ; " ;
                    $period[$dp][$i] = $data
                    ['period'] ;
                    $pm[$dp][$i] = $data['pm'] ;
                  }
                }
                $maxSize = $i + 2 ;
                echo "<td class = 'table80' colspan
                = '$maxSize'>$dp</td>\n" ;
        }
        echo "</tr>\n</table>\n" ;
    ?>
   </if>
</body>
</doc>
```

〈Figure 10〉 Active Document with PHP Commands

To maintain backward compatibility with Interleaf and achieve a seamless transition from Interleaf to HTML, a translator program written in Perl is added to IDP. It translates Interleaf ASCII into comparable HTML. With this embedded translator, user can create active document with Interleaf software, but gets the final document written in HTML, which can be opened and edited by MS Word. <Figure 11> and <Figure 12> illustrates the whole flowchart of the document-processing part of IDP and the details of IDP parser, respectively.



〈Figure 11〉 Flowchart of Document-Processor in IDP



〈Figure 12〉 Details of IDP Parser

3. Web-Based Design

It is usual that the active documents are created under Microsoft Windows and transferred to UNIX workstation, where document-processing is made. Two operating platform cross-over is inevitable. If a robust system exists which creates, launches and manages all active documents, the designer can do all design works with convenience. That is why web-based design system was born.

3.1 Web as a Design Tool

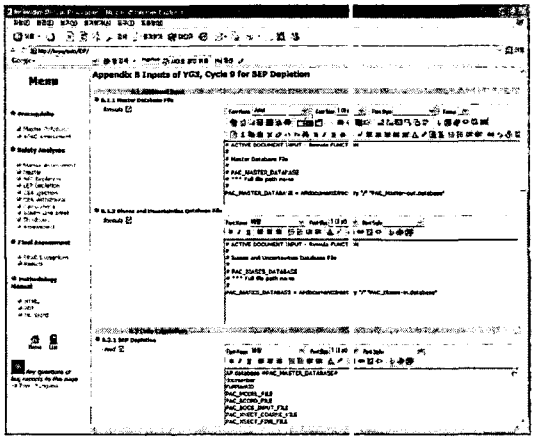
Web is used for design automation tool due to its merits such as openness, usability, scalability, flexibility, etc. Its Hypertext Transfer Protocol (HTTP) is suitable in multi-platform computing environment. In the late 1990s, KNFC has developed a web-based design automation system [Y. S. Park, et al., 2000]. Merging document-oriented design system with the web-based design system created the enterprising architecture of IDP.

Using the proper mix-up of server side and client side programming under the LAMP (Linux /Apache/MySQL/PHP) environment (See <Table 2>), the design process on the web is modeled as a design wizard style so that even a novice designer makes the design product easily. Therefore, when a designer interactively inputs basic data required in a reload design analysis via a web browser, error-free active document preparation, code run, output processing and documentation are performed automatically.

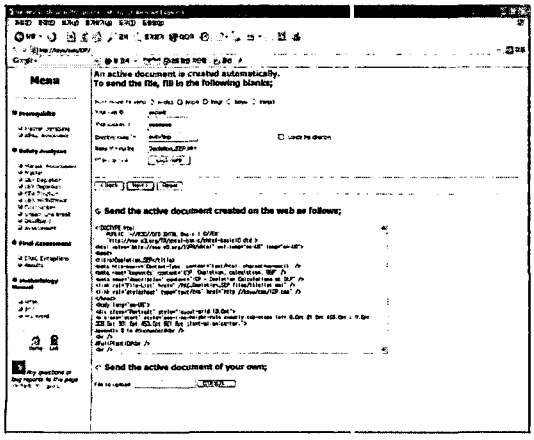
〈Table 2〉 Web Development Environment

	Server-Side Programming	Client-Side Programming
OS	Linux/HP-UX	MS Windows
Engine Program	Web Server (Apache)	Web Browser
Languages	PHP, Perl, MySQL	HTML, JavaScript

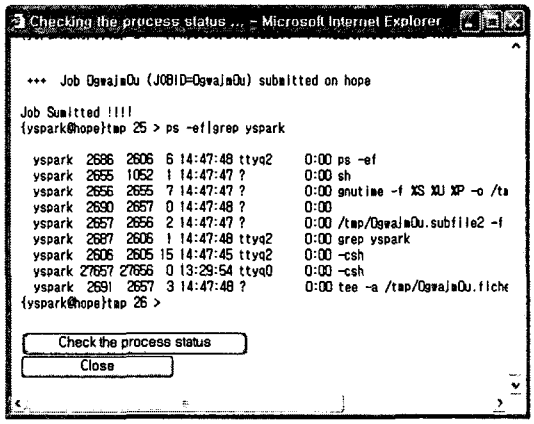
Figures 13 through 16 show how an active document can be created, launched and retrieved on the web. In order to give designer more freedom of editing active document template on the web, a simple WYSIWYG HTML editor is attached, as shown in <Figure 13>. <Figure 14> shows that designer can send the active document created on the web or his/her own one created by MS Word and etc. As shown in <Figure 15> the active document processing can be launched and the resultant document can be downloaded from the web (<Figure 16>).



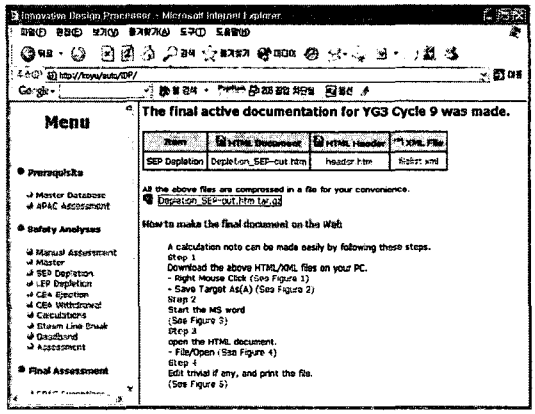
<Figure 13> Creating an Active Document on the Web



<Figure 14> Active Document Created and Ready to Transfer

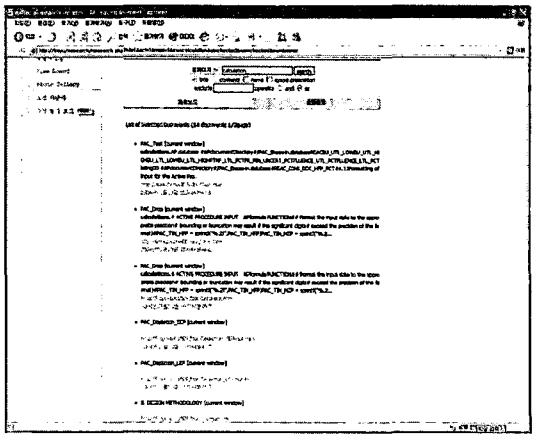


<Figure 15> Job Launched on the Web

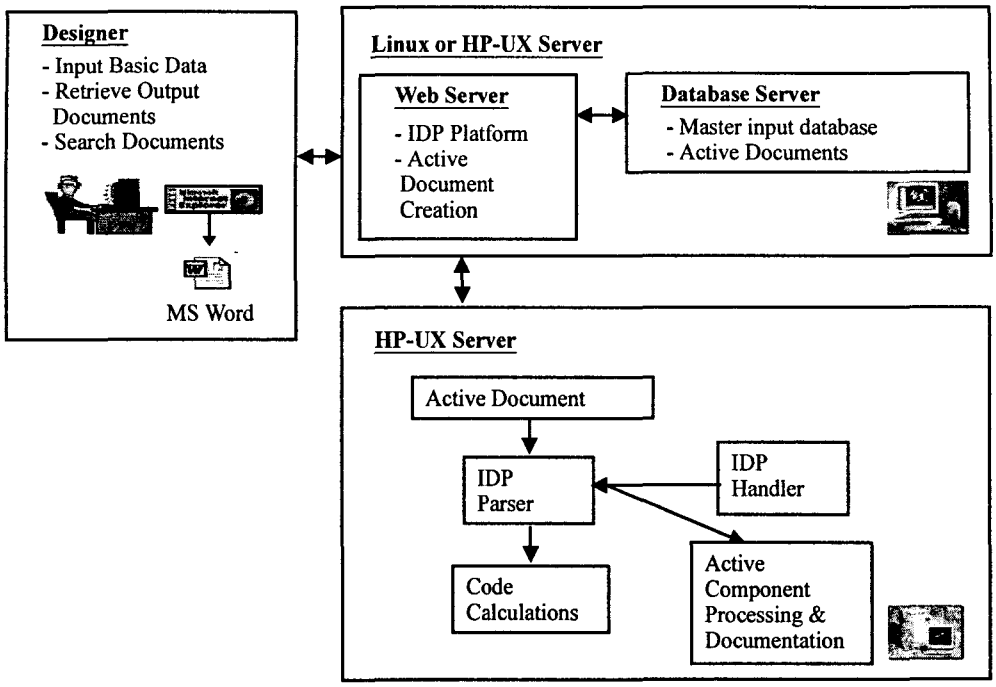


<Figure 16> Final Document Downloadable from the Web

3.2 Active Document Management on the Web



<Figure 17> Document Search Engine on the Web



<Figure 18> Overall Architecture of IDP

To manage active documents, a web-database system was built with PHP and MySQL. A number of active document templates for various tasks and reference documents including manuals are stored in the web-database system. All the active documents and reference documents can be searched easily by the search engine on the web (See <Figure 17>). It means that user can get all required information at user's fingertips. <Figure 18> illustrates the overall architecture of IDP.

4. Applications

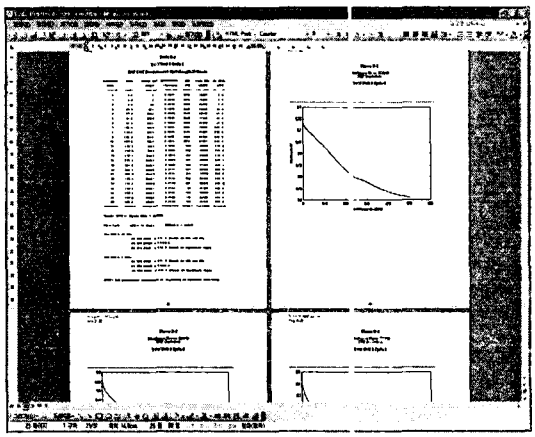
4.1 Physics Design Assessment

Physics design assessment consists of ensuring that the physics characteristics of the reload core design are supported by existing safety analyses. This is done by IDP for the following design items :

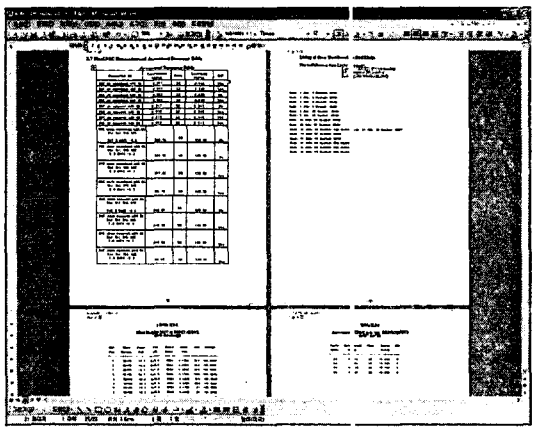
- Depletion calculations at short end point (SEP) burnup and long end point (LEP) burnup
- Peaking factors and axial shapes
- Symmetric rodded calculations
- Fine-mesh data
- Reactivity coefficients
- Kinetics parameters
- Scram worth and stuck rod worth
- Boron dilution analysis
- Steamline break analysis
- CEA drop analysis
- CEA ejection analysis
- CEA withdrawal analysis
- Single CEA withdrawal analysis

<Figure 19> shows the depletion calculation result obtained from the procedures shown in

Figures 13 through 16. The single CEA withdrawal analysis result is shown in <Figure 20>. Because the physics design assessment is highly automated with IDP, it takes about one week of computer time for whole calculations. Without IDP it would take several months to make whole calculation including documentation.



<Figure 19> Result of Depletion Calculation at SEP Burnup

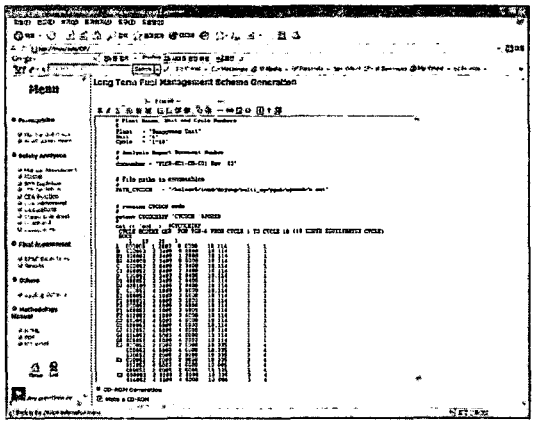


<Figure 20> Result of Single CEA Withdrawal Analysis

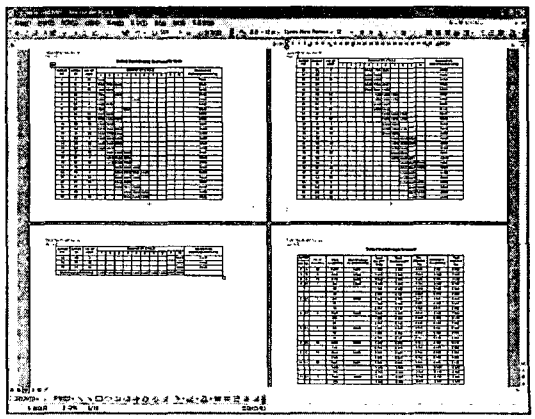
4.2 Long-Term Fuel Management Scheme

One of the reload design tasks involves the scoping studies over a cycle planning horizon. The scoping study provides the basic data in order for the utility to establish the long-term fuel management scheme in safe and economic manner. Included in the basic data are the required uranium masses in the forthcoming cycles, cycle lengths and burnups [Y. S. Park, et al., 1994].

If an active document for a long-term fuel management scheme is written and processed by IDP, the fuel management scheme can be made easily. <Figure 21> shows the active document preparation on the web and <Figure 22> illustrates the created fuel management scheme.



<Figure 21> Active Document for Fuel Management Scheme



<Figure 22> Result of Fuel Management Scheme Calculation

5. Conclusions

The automation using the IDP is now being implemented for all the reload design of Korea Standard Nuclear Power Plant (KSNP) type PWRs. The introduction of this process will allow large reduction in the nuclear design efforts. Great time saving was confirmed by showing that it can finish several-month jobs in a few days. Since the technology is also applicable to the non-nuclear design area like thermal hydraulic, fuel performance analysis and so forth, IDP can be used in all design tasks of PWRs.

In addition to the design tasks, the IDP can be used in all common tasks because the final result is usually a document with text, tables and figures, regardless of code calculation. Criticality calculations with MCNP code and monthly report generation can be such examples. Therefore, IDP will provide a platform for design and R&D tasks of KNFC under the already familiar web-work style environment.

References

[1] Chuck Musciano and Bill Kennedy, *HTML & XHTML : The Definitive Guide, 5th Edition*, O'Reilly & Associates, Inc., USA, 2002.

[2] Doug Tidwell, *XSLT*, O'Reilly & Associates, Inc., USA, 2001.

[3] Douglas B. Terry and Donald G. Baker, "Active Tioga documents : an exploration of two paradigms", *Electronic Publishing-Origination, Dissemination and Design*, Vol. 3, No. 2, 1990, pp. 105-122.

[4] Eric Meyer, *Cascading Style Sheets : The Definitive Guide*, O'Reilly & Associates, Inc., USA, 2000.

[5] George Reese, *MySQL Pocket Reference*, O'Reilly & Associates, Inc., USA, 2003.

[6] J.A. Brown, R.P. Harris, W.M. McDonald, R.E. Henderson, "The PAC Methodology for Reload Design Assessment", *Proc. of ANS Topical Meeting - Advances in Nuclear Fuel Management II*, ANS, USA, 1997.

[7] M.P. Rubin and S.G. Wagner, "Users Manual for the Active Procedure Toolkit (APtk)", *CE-CES-167-P Rev. 10*, ABB CE Nuclear Operations, 1999.

[8] Paul M. English and Raman Tenneti, "Interleaf active documents", *Electronic Publishing-Origination, Dissemination and Design*, Vol. 7, No. 2, 1994, pp. 75-87.

[9] Rasmus Lerdorf and Kevin Tatroe, *Programming PHP*, O'Reilly & Associates, Inc., USA, 2002.

[10] Robert Spinrad, "Dynamic Documents", *Harvard University Information Technology Quarterly*, Vol. VII, No. 1, 1988, pp. 15-18.

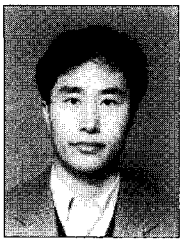
[11] Volker Goebbels, "Scientific Applications with PHP", *Proc. of International PHP 2003 Conference ; http://www.phpconference.de/2003/slides/business_track/goebbels_scientific.pdf*, Frankfurt, Germany, 2003.

[12] W3C, *Extensible Markup Language (XML) 1.0, W3C Recommendation, 2nd edition*, 2000.

[13] Y.S. Park, et al., "Establishing the Long-Term Fuel Management Scheme Using Point Reactivity Model", *Journal of Nuclear Science and Technology*, Vol. 31, No. 10, 1994, pp. 1001-1010.

[14] Y.S. Park, et al., "Interactive Nuclear Design Analysis Process Automation on World Wide Web", *Proc. of PHYSOR2000*, ANS, Pittsburgh, USA, 2000.

저자소개



박 용 수

Yong Soo Park is currently a senior researcher at Korea Nuclear Fuel Co. Ltd. He received his M.S. and B.S. degrees from Hanyang University in Seoul, Korea. H s research interests include design automation, neural networks and XML.

김 종 경

Jong Kyung Kim is currently a professor at the department of nuclear engineering, Hanyang University. He received his M.S. and Ph. D degrees from University of Michigan. His research interests include real-time system, computational mathematics and so forth.

◈ 이 논문은 2004년 3월 9일 접수하여 1차 수정을 거쳐 2004년 9월 12일 게재확정되었습니다.