iDocument Builder: An Environment for Building XML-Based Interactive Teaching Materials

Yasushi HAYASHI, Zhenjiang HU, and Masato TAKEICHI
Graduate School of Information Science and Technology
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo, 113-8656, Japan
{hayashi,hu,takeichi}@mist.i.u-tokyo.ac.jp

ABSTRACT

PSD (Programmable Structured Document) is a framework in which structured documents are edited efficiently and safely by evaluating embedded codes in themselves. Justsytem's xfy has a technology called Vocabulary Connection, which supports creating and editing source XML documents on the HTML view. Combining these two technologies allows us to produce an environment to support creating XML-based interactive teaching materials, which generate some outputs dynamically according to user's inputs. The way to produce the environment is explained and the usefulness of this tool is demonstrated by an example of an XML-based exam paper.

Keywords: XML, HTML, Authoring Tools, Teaching Materials, Document Processing

1. INTRODUCTION

Teaching materials on web pages are becoming increasingly popular as more and more students can access computers through the Internet in school or at home. It is widely recognized that web-based distributions of those materials have many advantages. Among them, materials that are interactively changeable by user's actions are considered to be very effective in promoting active learning. Those materials are usually written in HTML and some scripting language such as JavaScript since standard web browsers can handle them. In recent years, making an HTML from an XML by using a transformation language such as XSLT is becoming widely used. No matter which route is taken, it is difficult to construct such teaching documents since the author usually needs to learn HTML tags and Cascading Style Sheet (CSS) when writing an HTML directly, or XSLT when converting an XML to an HTML. Furthermore, to add an interactive nature to pages, the author needs to learn a scripting language such as JavaScript. The learning costs for these techniques are considerably high for the usual authors. Even for those who are familiar with these techniques, it is a time-consuming task to build such web-based interactive documents without some editing supports. We believe that the existence of a more useful environment to support building web-based interactive documents will encourage more teaching documents authors to try writing this kind of teaching materials. The following features are desirable for such an environment.

- The author can build a web-based document on the HTML view like when she writes a usual text with a usual editor without knowing the definition for elements.
- The author can easily add interactive actions by entering codes in the documents on the HTML view.

The aim of this work is to develop a system to construct such an environment by combining two recent XML related technologies, that is xfy [3] developed by Justsytem Corporation, our collaborator, and PSD (Programmable Structured Documents) [2] proposed by our PSD project [5] at the University of Tokyo. Using an environment constructed by our system, the author can easily make interactive documents (we call them *iDocuments*) that have the following features (that will be explained in detail in Section 6).

- Texts are dynamically changeable for user 's inputs.
- Textbooks or tutorials can be customized for a user.

- Programs in the textbook can be modified and executed on the document itself.

The structure of the rest of the paper is as follows. Section 2 explains how to predefine building blocks with which the author can build documents on an xfy window. Section 3 explains the predefined building blocks for iDocument. Section 4 describes our PSD evaluating system. Section 5 explains how to write an embedded code in Haskell. Section 6 describes the features of iDocuments and gives an example of iDocuments, that is exam paper (we call it *iExam*). Finally, Section 7 gives conclusions.

2. BUILDING BLOCKS ON XFY

Justsystem xfy [3] is an XML management architecture developed by Justsystem Corporation. It has an innovative technology called Vocabulary Connection (VC), which transforms a source XML document to a target HTML. The distinguished features of xfy that are useful for building interactive documents are the facility that the HTML view can take user's actions to edit the XML source document and the support to build documents from building blocks using the commands on the menu bar. Here, a building block can be regarded as an XML element that has some meaning as a document component. Vocabulary Connection Descriptor (VCD) is a scripting language to apply VC to XML documents. It can be seen as an extension of XSLT1.0. An important task for the construction of an environment for building a document on xfy is to predefine commands for inserting and deleating a building block in the VCD program, with which the author constructs an XML document. For example, a command for inserting a paragraph block after the element where cursor is located is defined by

```
<vcd:command name="add-paragraph">
<vcd:insert ref=
"vcd:caret-node()/
ancestor-or-self::psd:*[parent::psd:content][1]"
position="after">
<psd:paragraph />
</vcd:insert>
</vcd:command>
```

The command can be assigned to the main menu so that it is invoked by selecting it from the main menu on the xfy window. To achieve it, we include the following description in the VCD program.

```
<vcd:vocabulary match="idocuments"
label="idocuments"
call-template="root">
<ui:ui>
<ui:main-menu>
<ui:menu label="Block">
<ui:menu-item label="Paragraph"
command="add-paragraph" />
</ui:menu>
</ui:main-menu>
</ui:ui>
</vcd:vocabulary>
```

In addition to these definitions, we need to define how to display the paragraph element on the view in the same way we do it in XSLT. After describing these statements in the VCD program, the "Paragraph" command appears in the main menu, and selecting it automatically inserts an empty paragraph element in the source XML and displays a cursor with a prompt to enter a content of paragraph on the HTML view (Figure 1). Note that

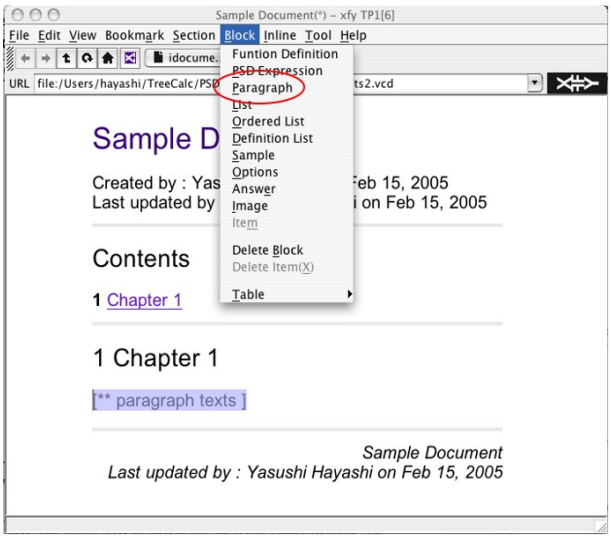

Figure 1: Inserting a new paragraph

VC allows the author to input the content directly on the HTML view.

In an environment with the necessary commands for editing building blocks predefined, the authors can create and edit the document by selecting a command from the main menu and inputting contents on the HTML view without knowing definitions for elements and attributes.

3. BUILDING BLOCKS FOR IDOCUMENTS

We have defined commands for editing building blocks for iDocuments. The building blocks we can

insert and delete by the commands under *Block* item of our main menu are classified into three kinds, that is basic blocks, PSD related blocks, and application specific blocks. The basic blocks are blocks we often use to write a general document. Those blocks include *section*, *paragraph*, *list*, *ordered list*, *sample code*, *image* and *table*. The PSD related blocks are related to the PSD manipulations. Those are *Function Definition* and *PSD Expression*. The Function Definition command generates an empty *psdfun* element whose contents will be definitions of user-defined functions given by the user. The view of a psdfun element shows its contents and a "Hide Code" button to hide the contents when needed. The PSD Expression command generates an empty *psdexpr* element whose contents will be expressions that can be evaluated by an evaluator. The view of a psdexpr element usually shows its contents and two buttons, that is an "Evaluate" button as a trigger of evaluation and a "HideCode" button to hide the embedded codes from the view. The application specific blocks are additional building blocks for documents for a specific application. For example, building blocks for iExam in Section 6 include *options*, and *answers*.

Another support for the document manipulation is *Inline* whose commands are used to give some effects to a part of sentence, for example, changing a font style, placing a link and breaking line.

4. PSD EVALUATING SYSTEM

In [2], we proposed the notion of Programmable Structured Documents. Simply speaking, PSD can be seen as an XML with embedded codes. It can be manipulated by evaluating an embedded code in itself. In that sense, the document can be seen as a program. In the current PSD systems, the content of a *psdexpr* element is regarded as an evaluatable expression that depends on other parts of the documents, and it is displayed with a trigger button for evaluation on the HTML view. When pressing the button on the view, the expression is evaluated by an evaluator and its result is displayed in the place the button was displayed.

We apply the paradigm of PSD to add interactive natures to teaching materials that are built on the xfy window. The main issue is how to realize this kind of evaluation process on xfy. The xfy's XML editor keeps data as DOM [6]. But we think that the language for embedded codes should not be restricted to languages like Java that can manipulate DOM directly. Especially, we hope that functional languages such as Haskell [4] can be used as well. Our design choice of the PSD system that works on xfy is the separation of the xfy editing system and an evaluator that evaluates expressions outside of xfy. In order to make this kind of evaluating system workable in practice, we should guarantee the existence of an efficient interface between the editing system and evaluator.

We developed "Pruned-tree DOM Interface" [1] as a plugin of xfy for that purpose. For efficient evaluation, it picks up only three kinds of necessary components from source XML documents and sends them to the external evaluator. Those components are the expression to be evaluated, the definitions of functions that are used in the expression, and the elements referred by the expression. Particularly, the referred elements are gathered in the form of a tree (called "Pruned DOM" tree) composed of a new "arg" node as its root and the subtrees whose top nodes are referred by the expression as direct children of the root. The external evaluator receives those components and generates a runnable code from them in the employed language, and then performs the evaluation. The editor reads the resulting output, parses it to generate a DOM fragment, and plugs it back into the source document. The process is illustrated in Figure 2.

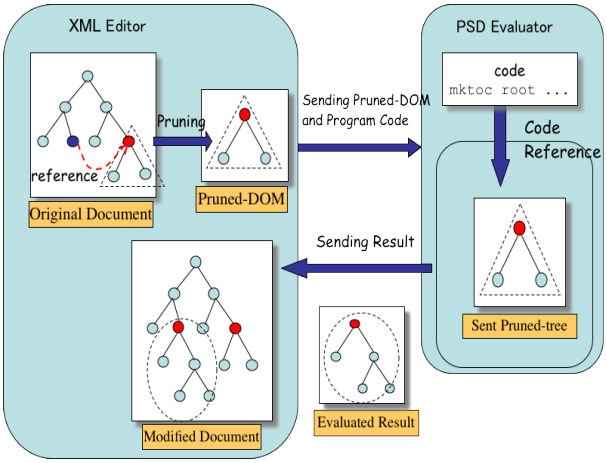

Figure 2: Pruned-tree DOM interface

5. EMBEDDED CODES IN HASKELL

The PSD evaluating system described above is, in principle, language independent. We have implemented it with Haskell as a language for embedded

codes. We use Haxml [7] to convert an XML to a Haskell' s tree data structure to handle it as a Haskell program. The underlying data modeling in Haskell is

```
data Element = Elem Name [Attribute] [Content]
data Content = CElem Element
             |CString String
```

An expression can have functions that are already predefined in a Haskell library or user-defined functions defined in the PSD. Those functions can take some Haskell expressions or other parts of the XML tree as arguments. For example, suppose the function *grade* takes three arguments, that is the correct answer, an answer given by the user, and a given score if the answer is correct, and then generates a score from the answer. The expression can be *grade* (2, ../answer,10), where "../asnwer" is the relative path of the element whose content is the answer given by the user, 2 is the correct answer, 10 is the score if the answer is correct. The definition of *grade* is given by the user as the content of a *psdfun* element in the document as follows.

```
  grade(x, (Elem "arg" a0 [CElem (Elem
  "answer" a1 [CString ans])]), y)
=(if x==ans
     then (Elem "psdresult" [] [CString y])
     else (Elem "psdresult" [] [CString  "0"]))
```

where, `(Elem "arg" a0 [CElem (Elem "answer" a1 [CString ans])])` is the pruned tree in Haskell expression generated from the reference `../answer`, and `ans` is the answer given by the user. The grade function compares `x` with `ans` and return a psdresult element whose content is `y` if `x` = `ans`, or return a psdresult element whose content is `"0"` if `x` $\neq$ `ans`.

6. IDOCUMENTS

Using iDocument builder, we have been developing several iDocument applications. Those include iTextbook (interactive textbook), iTutorial (interactive tutorial) and iExam (XML-based exam paper), and have the following features. First, texts are dynamically changeable for user 's inputs. This means the interactive nature of iDocuments. For example, in an iTextbook on algorithms, when an algorithm is described with sample inputs, intermediate outputs, and the final outputs, the description can be changed dynamically when the user changes some inputs because those intermediate outputs and the final outputs are recomputed by the evaluator. Second, textbooks or tutorials can be customized for a user. This means that the user can change and/or choose some contents of teaching materials. For example, she can hide some parts of materials she is already familiar with, or might be given options to choose if she need more detailed explanations. Third, programs in the textbook can be modified and executed on the document itself. This is particularly useful in a programming textbook. The user can change the example codes on the document and try to run it on the document itself.

An Example: iExam

As an example of the use of iDocument builder, we demonstrate how to build an iExam, which is an XML-based exam paper. When the author makes a new document, it is more efficient to build it from a predefined template than to build it from scratch. A template for iExam is predefined in the associated VCD program as follows.

```
<new:new-fragment name="empty-idocument">
  <psd:idocuments>
    <psd:create />
    <psd:last-update />
    <psd:section>
      <psd:title />
      <psd:content>
        <psd:paragraph />
      </psd:content>
    </psd:section>
  </psd:idocuments>
</new:new-fragment>
```

This definition enables the author to select "empty idocument" from the menu to generate the template on the new xfy window from which she adds contents of an exam (Figure 3).

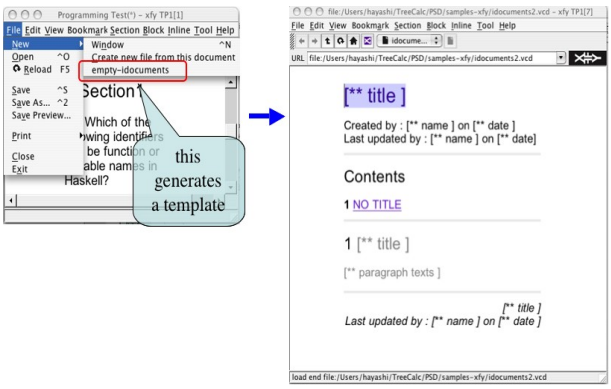

Figure 3: Creating a new exam

As explained in Section 3, we prepared building blocks for iExam. For example, *option* is a block used in the type of questions in which the user try to choose the correct answer from the given multiple options. The definition of the command to insert an option is:

```
<vcd:command name="insert-option">
<vcd:insert ref=
"vcd:caret-node()/
ancestor-or-self::psd:*[parent::psd:content][1]"
 position="after">
<psd:options>
<psd:item />
</psd:options>
</vcd:insert>
</vcd:command>
```

This command is assigned to the main menu by the description:

```
<vcd:vocabulary match="idocuments"
 label="idocuments"
 call-template="root">
  <ui:ui>
   <ui:main-menu>
    <ui:menu label="Block">
     <ui:menu-item label="Option"
     command="insert-option" />
    </ui:menu>
   </ui:main-menu>
  </ui:ui>
</vcd:vocabulary>
```

These descriptions enable the author to select "Options" from the main menu to add an empty option to which she enters a content (Figure 4).

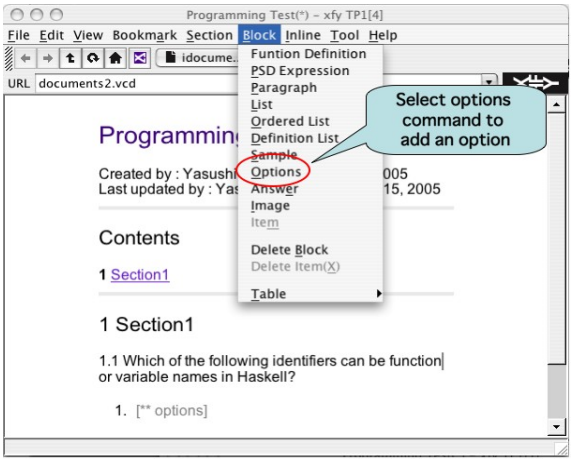

Figure 4: Adding contents

After making options and an answer box, suppose the author wants to make a button to grade the user's answer. She selects "PSD Expression" from the main menu to generate an empty *psd-expr* element that prompts the author to input psd codes and displays the two buttons "Evaluate" and "HideCode". The function *grade* takes three arguments, that is the correct answer, user's answer, the score given when the given answer is correct. For example, it can be `grade(2, ../answer,10)` (Figure 5).

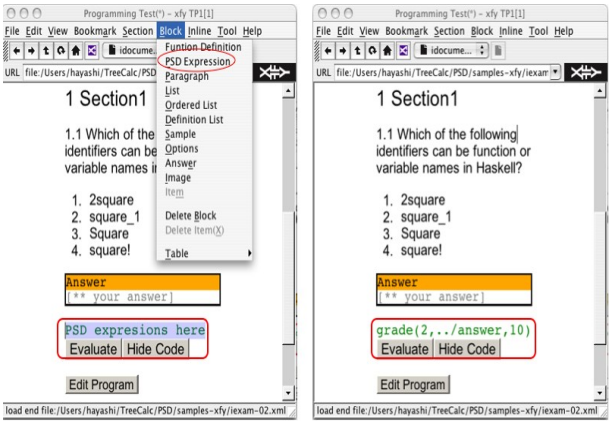

Figure 5: Inputing a program

Pressing "HideCode" button hides the code from the view. Pressing "Evaluate" button after the user inputs an answer evaluates the expression and displays the obtained score (Figure 6).

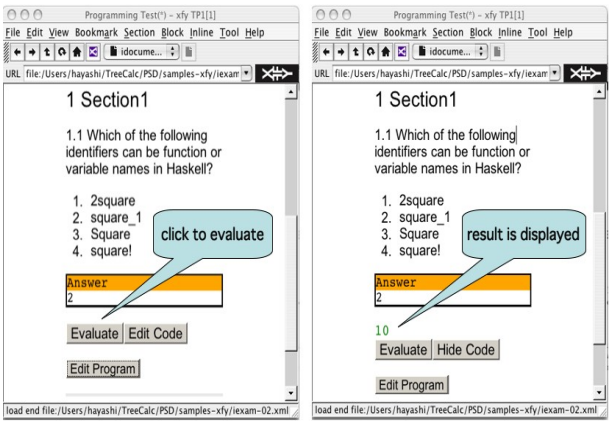

Figure 6: Evaluating expression

Note that the definition of *grade* function is given by the author selecting "Functions Definition" command and entering the defintion in Haskell.

Although this example is very simple for the explanation purpose, the same technique is applicable to a type of questions that involves more complex computations to evaluate user's answer. Also, by giving appropriate expressions, it is easy to calculate the total score automatically when the exam includes multiple questions, and to provide the user with instructive comments about each answer. This kind of questions can be easily inserted at any point of an iTextbook by using iDocument builder as exercises in order to test the user's understanding of the material.

7. CONCLUSIONS

We made use of Justsystem xfy technology as a basis of document processing environment. The facility of xfy to create and edit an XML document on the target HTML view enables the view to accept user's actions. We defined commands to insert and delete building blocks of documents in VCD so that the author can create and edit the document by selecting a command from the main menu and inputting contents without knowing definitions for elements and attributes. To add interactive natures to materials, we developed a PSD evaluating system with an interface called pruned-tree DOM interface. The author can enter some code on the HTML view that takes the user's input and generates some output dynamically. We demonstrated that how easily this interractive document can be built using the example of XML-based exam paper.

Acknowledgement

This work has been done under collaboration between University of Tokyo and Justsystem Corporation. The project is supported by the *Comprehensive Development of e-Society Foundation Software* of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

[1] Y. Hayashi, Z. Hu, M. Takeichi. N. Wake, M. Hara and N. Oshima. Pruning DOM Trees for Structured Document Processing. In *The 21st JSSST Annual Conference*, 2004.

[2] M. Takeichi, Z. Hu, K. Kakehi, Y. Hayashi, S.-C. Mu, and K. Nakano. TreeCalc: Towards Programmable Structured Documents. In *The 20th JSSST Annual Conference*, 2003.

[3] Jastsystem corporation. xfy technology. `http://www.xfytec.com`.

[4] S. P. Jones. Haskell 98 Language and Libraries. Cambridge University Press, 2003.

[5] PSD Group. Programmable Structured Documents(PSD). `http://www.psdlab.org/en/`.

[6] W3C. Document Object Model (DOM). `http://www.w3.org/DOM`.

[7] M. Wallace and C. Runciman. Haskell and XML: Generic Combinators or Type-based Translation?. In *Proceedings of the 1999 ACM SIGPLAN International Conference on Functional Programming*. ACM Press, 1999.