Source Code Publishing on World Wide Web

Hayashi, Tsunetoshi
College of Life Sciences and College of Image Arts Sciences
Ritsumeikan University
Kyoto and Kusatsu, Japan
Email: tkmktoshi@gmail.com

Abstract—The design principle of publishing program source code document on-line is presented. In this design, Knuth’s literate programming is employed as the foundation of the publishing method. The concepts similar to world wide web are adapted to our publishing method utilizing the extended markup language design.

Concrete syntax of the markup language and some discussions on the document delivery of the methodology are also presented. Finally, notes on the implementation are given, as closing remarks.

Keywords—literate programming; source code publishing; world wide web page; xml;

I. INTRODUCTION

While we regard computer programming in general is an intelligent academic discipline comparable to mathematics, or physics, there may be some people who do not think so. This is because mathematics is usually engaged in rather higher level of abstraction, however, computer programming requires a lot of tedious lower level engagement as well as higher abstraction layer considerations. This duality prevents computer programming from presenting formal verification process like mathematical publications. This process is much more required as the importance of software reliability is increased. If computer programming would like to have the similar status as mathematical disciplines, it should make formal verification through peer review process.

For peer review process, the publishing method of computer programs is very important. Computer program publishing involves at least source codes as well as the explanation document about the codes. As a matter of fact, the practice of computer program publishing is in very poor status, except for some rare cases [3]. Many open source software projects are publishing source codes as their projects’ results, usually in raw text format, without any formal documentation. Comment lines in the source codes are insufficient for above mentioned peer review.

We are currently designing an on-line publishing mechanism using the literate programming and principle of the world wide web. The literate programming gives the formal document and source code at the same time, as well as excellent programming methodology. The world wide web principle, through use of extended markup language, gives system and language independent description of programming documents. And it will give any program a higher level abstraction description.

We will give the outline of our source code publishing design in the rest of this paper.

II. LITERATE PROGRAMMING

Literate programming [3] is a programming technology developed by Donald E Knuth, and extensively used in his $\TeX$ programming [6]. Literate programming is either a means to publish source codes and their documentation in a consolidated and comprehensive document, and, at the same time, is a means to force programmers to employ the programming methodology similar to the stepwise refinement.

A. Basic principles

In literate programming, the source code is divided into many fragments, called sections, in relation with their roles.

- A section may contain a range of highly abstract function definition down to concrete, a few lines of source code, accompanied with the document text explaining how the section is implementing its role.
- A section is referred to by its name, that is an adequate text concisely explaining its role.
- The programming language in which the source code is written is enhanced to permit section names may be present in any part of source code, as reference to separately defined sections. In a sense a section name is viewed as a virtual hyperlink to that section.
- Sections are organized by stepwise refinement programming principle, and not by code order defined by a programming language.

Literate programming can be, if executed well, applied to toy problems to large scale programming tasks such as $\TeX$ document formatting program, however, it may be best exploited in off-line, printed publishing.

B. Knuth’s WEB language

Knuth developed WEB language to write his software master pieces: $\TeX$ and METAFONT. Do not confuse this language name WEB with the world wide web.

The WEB language is designed as follows.
Pascal is the programming language of choice for source code.

Of course, \TeX is employed for preparing documentation of the source code.

Letter “@” identifies commands organizing the program document, such as sections, hyperlinks, and simple text macros, as seen in Fig. 1.

For writing a WEB program document, we have to juggle with three separate, mutually independent languages, as well as algorithm development and document preparation.

A WEB program is typically written as a large text file, its document and source code have to be extracted from the text. For this purpose, two programs Tangle and Weave are developed.

Tangle extracts Pascal code, replaces hyperlink by its body, processes macro definition. The resulting output is to be compiled by the Pascal language processor.

Weave pretty-prints Pascal source code, and output them with its explanation as a \TeX document source, which then will be processed by \TeX document formatter, as shown by Fig. 2.

This process is adequate for off-line working style, however, in case of developing software on-line, source code and document should better be obtained also in on-line. Then our motivation is that, combined with world wide web document publication capability, the literate programming concept will be far more usable in the modern program development environment.

For the purpose of fast development and deployment of any software, on-line publication of its program document and source code as well as the running application product is prerequisite.

However, manipulating three mutually different languages for document preparation and source coding is rather tedious and inadequate for publishing on-line.

III. LITERATE PROGRAMMING ON WORLD WIDE WEB

Therefore we propose an on-line source code publishing system, based on literate programming using the world wide web principle.

The publication method should employ the same underlying language for documentation, source code, and organization definition. This publication system may be also usable as the front end to any version control program, consolidating both functions of source code publication and development.

As a matter of fact, there is just one on-line publication means widely used nowadays for any document, that is, the world wide web on the Internet. Then we must exploit the basic principles underlying the world wide web, as the foundation of source code publication method. The world wide web is based on two concepts, the Hyper Text Markup Language (HTML) [5] and hyper text transmission protocol (http).

With this premise, the design of our source code publication should based on HTML, or its foundation eXtensible Markup Language (XML) [2]. The program documentation should be written in HTML, or XHTML; the source code and organization definition should be written in the XML format. The syntax for the new source code and organization XML data types have to be newly defined. And, they will be given in the following.

Note that XML tag definition in the following is only for presentation of the concepts, and for explanation purpose. They do not strictly conform to the W3C XML specifications [2].

A. Documentation in HTML

As mentioned earlier, Knuth’s literate programming employs \TeX document formatting program as the engine for program document portion. Since we like to construct our on-line source code publication system around world wide web on the Internet, our program document will be based on HTML, and nothing else.

The document may include mathematical formula described in some extensions to HTML such as MathML.

Usually a HTML document is enclosed by \texttt{<html>} and \texttt{</html>} tag, formatted as single page. However, the literate program document must have many sections, it may consist of many document elements. Therefore some considerations must be required, which will be discussed later part of this section.

B. Source code in XML

Any programming language can be expressed in an XML format defined as follows. We named this format codeML.

The internal structure of a program will be expressed as the program tree, or abstract program [7] rather than
the parse tree. In an existing compiler, the internal tree of
a source program is implemented as the binary tree with
various operators.

If we could represent this binary tree literally as XML
document, a program may be described in XML. The
following tags in Fig. 3 are only needed.

An expression equivalent to $1 + 2 * 3$ can be given as the
following XML fragment (Fig. 4). This description is very
simple but redundant and not easy to comprehend. Rather
we should give sufficiently rich repertoire of tags to reflect
the structure of a program. The tags should be prepared
according to the hierarchy of the program structure. That is:

- Module, file
- Class, function
- Control structure
- Executable statement
- Expression
- Variable

And we will discuss concrete design of the tags in the
following section.

According to the XML specification, a tag may contain
any number of attributes in its definition, which may reduce
the number of different tags. We didn’t utilize tag attributes
in this version of codeML definition.

C. Organization defined by XML

The XML tag definition of document element is rather
simple, it only requires a few number of tags. We tentatively
named this XML definition as DocML. The tags define
major and minor section, start of code fragments, and name
of code fragment.

The tags are shown in Fig. 5.

D. Embedding elements of source code document

In Knuth’s WEB language, the whole program document is
divided into multiple program fragments through organization
commands. A fragment is constructed by TeX document
source and Pascal source code. The organization of its
component elements is straightly hierarchical and simple.

Whereas, our proposed publishing method, the sectioning
and organization of a document is not so. Since HTML
usually specifies just one page document, our program
document is consisted of many fragments. The CodeML
specifies a source code element, and DocML defines the
organization of these component elements. They cannot be
embedded in a simple hierarchy.

There may be two solutions to organize these markup
language hierarchy.

1) Consolidated format. Single HTML document embeds
DocML, HTML and CodeML elements. The program
document organization is defined by DocML elements.

2) Distributed format. The program document consists
of multiple HTML documents. An HTML document
embed DocML and CodeML documents, representing
a single code fragment.

The outline of these formats are also shown in the appendix.

The consolidate format solution is suitable for the archiv-
ing of program documents, where understanding what the
program is doing is important. This format is suitable for
reading the whole program document.

The distributed format is suitable for the program develop-
ment and management. A program may be constructed and
manipulated by each fragment, and viewed by the hyper text
method as each fragment is treated as single page.

Anyway, documents in these two formats can be mutually
converted easily by simple program. A http server may
do this conversion if necessary. Even client side script
may be carry out this conversion. The conversion from the
distributed format to the consolidated format seems easier.

IV. DELIVERY OF CONTENTS

To publish a document means that its content must be de-
libered to viewing clients through some accessible measure.

We employ the world wide web principle as our means of
contents delivery. The clients may read program document
using their world wide web browser or some dedicated
viewing applications depending on the his intension on the
document. A client may want to understand the working of a program, or to obtain its source code for compiling and making the application.

The delivery method may vary depending on the format of the program document.

A. Program documentation

The delivery of program documents is not much different from that of the world wide web method. If client wanted to view the document at as a whole, or to keep the document as a printed volume, the document should be better delivered as one single page. The consolidate format document is suitable for this kind of purpose.

- If client wanted to read the document by the fragment with hyperlinked access, the distributed format document is adequate. The delivery method is also the same as that of the world wide web.
- If client want to participate program development activity, the browser must provide editing capability. The client side scripting or dedicated application might be necessary.

The CodeML element must be converted to pretty-printed conventional programming source code text by the server side application before delivery, or the client side scripting after delivery and before displaying on screen.

This CodeML to text conversion is carried out by top-down, depth-first tree walk on the DOM tree of CodeML element, emitting source text in the target language while scanning.

B. Source code

Delivery of source code text requires some processing on the server side as TANGling the program document is required. This process produces single CodeML document containing the full code of the program.

The generated CodeML document may be delivered in this format, or delivered as the source code text of any of conventional programming language, depending on the request of the client. The conversion is done in the same way as the above mentioned program document case.

For source code delivery, the distributed format document seems to work better.

C. Organizing contents

The program documents are stored and accesses on the server of on-line delivery system. The consolidated format document handling is not much different from that of usual world wide web server.

Whereas, the distributed format document requires some indexing treatment for retrieving related documents. Because, the TANGling of source code carries out the tracing of hyperlinks of all of relevant fragments.

V. CODEML SPECIFICATION

We need to establish the specification of CodeML to define the tags required for CodeML. for the time being, we employed the following principle. Our intention is that CodeML should be language independent, universal, and represent the intrinsic computing structure. Therefore tags do not specify reserved words and key words as well user defined identifiers, instead these words are given as text data placed between tags. Tags specify essential semantics of program source code.

They must be taken into considerations when we develop the interpreter of the language.

The next discussion concentrates on the specification of XML tags exploited in CodeML.

Variables Variables are statically typed. They can hold objects, numbers, strings. Array type is predefined class. However, tags define static variable declarations and their scope, and initial values.

- <vars> tag delimits the variable declaration.
- <name> tag gives a name of a variable.
- <type> tag gives the type of a variable.
- Optional <expression> tag gives an initial value.
- <name> with optional <expression> declares single variable.
- Single variable declaration may be repeated as required.

Expressions Operators may include arithmetic, comparison, logical, assignment, member selection, listing, and function call operators. The tags just define the expression tree structure, however.

- <node> tag gives a tree node. A tree node have three sub elements:
  - <op> tag is for its operation code.
  - <left> tag gives its left branch.
  - <right> gives its right branch.
- <left> and <right> may contain <node>, <name>, <value>, or empty.

Executable statements Executable statements are expression, return, break, and interrupt handler.

- <block> tag groups several executable statements together into a single executable statement.
- Each control structure is also executable statement.
- <expression> is executable statement. This turns out to be either assignment or function call.
- <return>, <break> tags give return and break statement.
- <return> may have optional <expression>.

Control structures Control structure consists of what control, that is, conditional or repetition, conditional expression and executable blocks. CodeML control structure tags reflect this consideration.

- <control> tag with embedded <name>, optional <default>, and repeated pairs of <cond> <block>.
• <name> with text like if, do, and case specifies the kind of the control, that is, selection, repetition and so on.

Function and class definition  Function definition must give the name of the defined function, parameters, local variables and executable statements. Class definitions must give the name of the class, local variables, and functions as their methods. These are easily described by adequate tags.
• <function> tag has <name>, <params>, <vars> and <block> tags embedded in this order.
• <class> tag contains <name>, <vars> and <function> tags.

Modules and files  Modules and files give the unit of programs. Tags define a program module and files define the unit of program request and loading.
• A CodeML module is delimited by <codeml> tag.
The outline of CodeML DTD is presented also in the Appendix.

VI. CONCLUSION

Here we will give some discussions on the client side applications and web server program design with respect to the server document format as concluding remarks.

A. Source code translation

Since the source code in a program document is in CodeML, it must be translated to source code text of adequate programming language. There can be two entities to work this translation: 1) the world wide web server program, and 2) the client browser program.

For the server program, this application supporting function is not unusual, any server applications, except for the most simplistic one, require such built-in functions, like as the facility to access contents database.

If this translation is left to the client browser, some heavy scripting will be necessary, however it can be achieved by any powerful modern browser. Or else, a browser dedicated to program document viewing had better be newly developed using an existing HTML document rendering engine.

B. Program document viewing

A program document may be viewed as 1) a single page, or 2) hyperlinked multiple pages.

The former viewing is useful for saving the document for later time and printing the document for detailed examination. This is like Weaving of the program document. The consolidated document format is suitable for this single page viewing. To view distributed format document as a single page requires collecting all of relevant pages and merge them as a single HTML document. And this task is carried out by the document server, or the client browser. For this purpose some page index database may be prepared on the server beforehand.

The consolidated format document is not adequate for multiple page viewing. To view a program document as multiple pages may be best suited for the distributed document format. Here again, as described above, the name of a fragment and hyperlink index database should be prepared on the server in advance.

C. Source code distribution

The program document publishing facility includes distribution of its source code, or the assembled executable application. This requires the CodeML to text translation and source text integration, and is regarded as document Tangling.

The process is similar to that of viewing the distributed format document as a single page. This could be achieved by both of the server and client browser. If this process is to be achieved by the server, it may extend its job further to make the executable application.

APPENDIX A.

DOCML DATA TYPE DEFINITION OUTLINE

<!DOCTYPE docml>  
<!ELEMENT docml sect+>  
<!ELEMENT sect (name, subs+)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT subs (name, code)>  
<!ELEMENT code (refer+)>  
<!ELEMENT refer (#PCDATA)>  

APPENDIX B.

CODEML DATA TYPE DEFINITION OUTLINE

<!DOCTYPE codeml>  
<!ELEMENT codeml (vars|class|function)+>  
<!ELEMENT vars (name, type, expression?)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT type (#PCDATA)>  
<!ELEMENT class (name, (vars|function)+)>  
<!ELEMENT function (name, params?, vars?, block)>  
<!ELEMENT params (name)>  
<!ELEMENT block (expression|control|break|return)>  
<!ELEMENT control (name, default?, (cond, block)+)>  
<!ELEMENT break EMPTY>
The next file contains...

And so on.

REFERENCES


