Single-source Publishing using Starfish UPTEP Method: Applications in Multilingual Web Site Development and Courseware Content Management

Vlado Kešelj
Faculty of Computer Science
Dalhousie University
6050 University Ave, NS, Canada
vlado@cs.dal.ca

***Abstract*—Single-source publishing has been known for decades as a method of creating and maintaining published content from a single source document, while generating different content forms. It has usually been associated with the maintenance of a system documentation and it has been implemented in specialized systems around certain particular document forms. We present a case that such an approach would be useful in an open-ended general area of text-based document management applications, such as creation and maintenance of multi-lingual web sites and courseware material maintenance in LaTeX. Such application areas are very dynamic and open in types of target text, which is used for further processing and document rendering, so a closed system for single-source publishing would not be appropriate. We demonstrate an approach using universal text preprocessing and text-embedded programming (UPTEP) that can be effectively used for this task. This solution approach is validated through the use of open-source Starfish system, which is a general UPTEP system. We show how the UPTEP approach to single-source publishing can be readily implemented using Starfish on the two case studies.**

***Index Terms*—Preprocessing and Text-Embedded Programming (PTEP); Universal PTEP (UPTEP); Web Engineering; Courseware Management; Single-source Publishing; Text Processing;**

I. Introduction

Single-source publishing has been known and was in use for many years. It was explicitly discussed since at least 30 years ago [1], [2], [3], when it was particularly useful in generating and maintaining system documentations. The main advantage of a single-source publishing system is that it is easier and less error-prone to maintain documentation in a single source file or files, where we have only one place to generally update any change to the documentation content. From this single source, different documents in form, modality, length and similar, are generated. In the context of this line of research, a specific system would usually be built for a particular use, with pre-specified language of the source documentation, and conversion branches would be implemented which generate different output document formats. One of the earliest strong advocates for single-source publishing in software development was Donald Knuth in 1980's with his approach to program writing using the Literate Programming approach, as implemented in his early system WEB for the Pascal programming language, and the in an immediately following version of CWEB for the C programming language. [4], [5], [6]

A. Closed-system vs. Open-system Single-source Publishing

We refer to this traditional approach to single-source publishing as a *closed-system single-source publishing*. Implementing such a system is a relatively obvious and straight-forward, and these systems were usually based on defining a unified SGML or XML based source markup language, which is then compiled and translated into some target formats, such as HTML, PDF, and similar. A disadvantage of this approach is that it is relatively rigid, providing typically a predefined template about how documents will look like, and it cannot be used in a fast prototyping approach, where we often develop the target document, and then we want to generalize it into multiple document modalities. Two examples of such situations are development of a web site, and development of a course content.

B. Web Site Example

Let us assume that we have developed a prototype of a web site with several pages, such as `index.html`, `a.html`, and `b.html`, and others such as images, CSS and JavaScript files; and we would like to mirror it in another natural language, in files with slightly different names, such as `index-en.html`, `a-en.html`, and `b-en.html`. We choose language extensions for file names such as `-en` for English, or `-sr` for Serbian. We will focus on HTML files, but the approach is also applicable to other textual files. Creating and maintaining this system would be made easier by having a single-source files where we keep sentences or paragraphs in different languages in the same file in a close proximity. We could then imagine extending such system into multiple languages, multiple language encodings or alphabets, and even incorporating an automated translation function which would keep aligned texts synchronized. To make this work, we would need a system to process the single-source documents into documents with different modalities; i.e., languages.

C. Courseware Example

The second example is a courseware content management system, where we may be developing lecture notes in LaTeX for students, and we want to maintain a set of slides in parallel with those notes, which are also created in LaTeX. Alternatively, we may be developing slides first, and follow with their inclusion in notes, and gradual morphing of those notes into a textbook-style narrative of lecture notes. The target typesetting language does not have to be LaTeX, we could imagine the same scenario with HTML, Markdown, or other languages that can generate slides and documents, or even a mixture of different languages, one for slides and another one for lecture notes. Additionally, we can add more modalities other than just slides and lecture notes—there could be course notes as a more textbook-like set, or instructor notes as larger-font notes for use by an instructor in class.

Both of these examples could be implemented using a customized text processing systems, but that would lead to developing one-off customized system each time we generalize our documents into a single-source publishing format. Our objective is to show that a standardized universal preprocessing and text-embedded programming system (UPTEP), such as Starfish, can readily be used for this task by the standard brief customized modifications envisioned by such system.

The contributions of this paper can be summarized in three main items:
(1) We propose a solution to the important problem of open-system single-source publishing using the universal preprocessing and text-embedded programming (UPTEP) approach.
(2) The UPTEP approach to open-system single-source publishing is implemented as a set of functionalities in the software system Starfish—an open-source implementation of UPTEP in Perl.
(3) The implementation is validated in two practical case studies: multi-lingual web site development and maintenance, and courseware content management.

The rest of the paper is organized as follows: Section II presents related work in the problem of single-source document publishing. Section III describes the UPTEP methodology and its Starfish implementation, which is applied as a solution approach to our research problem. Section IV presents an evaluation of our approach through two validation cases in multi-lingual web site development and courseware content development. Finally, we conclude the paper in section V by summarizing the paper and giving some ideas for future research.

II. Related Work

The foundation ideas of single-source document publishing can be linked to the invention of the TeX typesetting system [7], with the first appearance in 1978, and the concept of Literate Programming [4], [6] in 1980s by Donald Knuth. Literate programming was initially implemented in the WEB system, with Pascal as one of the target languages, and later reimplemented with C as CWEB, which had its initial release in 1987. The Literate Programming methodology was published and introduced as a better way to write programs and well-typeset program documentation in the same time. Although specialized in this way for software development, from a more general point of view it can be regarded as single-source publishing system, with C as a target programming language, and TeX as another target format, used as typesetting language for documentation.

Later developments led to explicit closed-system single-source document publishing systems, one of which is WinHelp, which was released as a part of the Windows 3.0 system in 1990. [3] The SAS Institute described their single-source publishing system based on SGML in 1996 [2]. Later and up to this date, there were a number of single-source publishing systems, some of the most popular ones are DocBook [8] and Oxygen [9], which are all closed-system type systems.

The UPTEP universal preprocessing and text-embedded programming methodology was introduced in 2001 [10] and later further described and formally defined [11], [12], [13], [14].

III. UPTEP and Starfish Methodology

The goal of Universal Preprocessing and Text-Embedded Programming (UPTEP) is the create a standardized, universal, and uniform framework for text preprocessing and text-embedded programming, which could be easily and effectively used in generic textual contexts [14]. Starfish is one actual implementation of this approach [10].

The UPTEP framework is implemented in the Starfish open-source system. The system is available in the CPAN [10] Perl software repository and it is easily installed in the Linux and similar Unix-like system, as well as Windows systems with Perl language installed. The starting idea of the Starfish approach is based on inserting short snippets of Perl code between delimiters `<?` and `!>` in an arbitrary text. If the `starfish` command is executed on such file, the snippets are executed and their input appended to the snippets themselves. This is called the *update* mode of computation. The *replace* mode of computation works by treating the file with the snippets as a source file, and writing the output as the target file. In this way, the snippets in the source file are replaced with their output in the target file. The replace mode is similar in a way to the concept of notebooks in R or Jupyter, where the code is immediately followed by its output. In order for the replace mode to work in direct textual context, Starfish allows snippet code to be protected by commenting it out from the surrounding text context.

In these two case studies, we will use the replace mode of Starfish. The replace mode is similar to PHP in the sense that the code snippets are delimited by the escape sequences `<?` and `!>` by default. The Starfish system is flexible in the sense that it used different espace sequences in some file types, and it allows the user to redefine these espape sequences.

Table I shows some examples of escape sequences in Starfish, and compares them to some other text-embedded programming systems. The reason why these other text-embedded programming systems cannot be easily used in our two case examples of single-source publishing is that the escape sequences

TABLE I
ESCAPE STRINGS IN SOME SYSTEMS

System	Escape Strings	
	Begin	End
PHP	<?	?>
	<?php	?>
ASP	<%	%>
JSP	<%	%>
ePerl	<?	!>
Text::Template	{	}
Text::Oyster	<?	?>
HTML::EP	<ep-perl>	</ep-perl>
Starfish (default)	<?	!>
Starfish (HTML style)	<!--<?	!>-->
Starfish (user defined)	<?sfish	!>
Starfish (user defined)	*any string*	*any string*

and semantics of code snippets are fixed, and they do not allow as direct and easy implementation of markup needed for single-source publishing shown in our examples.

The flexibility of Starfish which is used is two-fold:
1) the escape sequences can be redefined, and defined in more forms, including regular expressions; and
2) the semantics of the snippet can be changed not only to execute embedded language and use predefined output routine, but to for example selectively include or ignore text, or to have some other special semantics.

This pair of escape sequences or other form of code activation and the semantics of the activation is called a *hook* in the Starfish context. The hooks can be added or removed in the snippets themselves, or defined in the Starfish configuration files, which makes it highly flexible to easily implement short and convenient markup conventions.

IV. SINGLE-SOURCE PUBLISHING IN STARFISH

Both examples of single-source systems, that we mentioned, can easily be implemented in Starfish, due to its flexibility. Namely, the escape sequences <? and !> can be redefined to define arbitrary so-called *hooks*, which can be activated in a number of ways. For example, in the web multilingual example, we can define in the source file index.html types of hooks such as shown in Listing 1 or we can use multi-line hooks such

```
:sr tekst na srpskom jeziku
:en text in the English language
```

Listing 1. Example of multi-lingual one-line hooks to use text versions in Serbian (*transl.* "text in the Serbian language") and English.

as shown in Listing 2. When producing a target file such as index-sr.html only the lines tagged as 'sr' are selected, and for index-en.html only the lines tagged as 'en' are selected. The other untagged lines are output in both cases.

```
<sr>
tekst na srpskom jeziku
</sr><en>
text in the English language</en>
```

Listing 2. Example multi-line hooks to use text versions in two languages.

This example shows how to maintain a site in two languages, but it can directly be generalized to multiple languages by introducing multiple tags. One such example would be official web sites in Bosnia and Herzegovina where frequently versions in three (BCS), or four (BCS and English) languages are presented, with addition of two alphabets (Latin and Cyrillic) for some languages.

Similarly in the courseware example, we would define tags such as 'sl' for slide content, and 'l' for lecture notes content to separate target content for these two types of documents.

A. Starfish Multi-lingual Web Site Implementation

The Starfish can process any HTML document with predefined hooks <!--<? and !>-->. In order to setup hooks for language separation in the output files as describe above, we simply need to add new hooks depending on the target output files. For the Serbian and English languages, with on-line tags :sr and :en and general tags <sr>..</sr> and <en>..</en, we need to insert the code at the beginning of an HTML document as shown in Listing 3.

```
<!--<?
if ($Star->{OUTFILE} =~ /-sr.html$/) {
    sfish_add_tag('sr', 'echo');
    sfish_add_tag('en', 'ignore');
elsif ($Star->{OUTFILE} =~ /-en.html$/) {
    sfish_add_tag('en', 'echo');
    sfish_add_tag('sr', 'ignore');
}
!>-->
```

Listing 3. Initial Starfish code to enable one-line and multi-line hooks

The above code is Perl snippet code in Starfish, which uses pattern matching to decide which parts of the source file to use based on the name of the output file. The translation of the single-source file into the target file is done with the simple starfish commands in a command-line interface as shown in Listing 4.

```
starfish -replace -o=t/index-sr.html \
                              index.html.sfish
starfish -replace -o=t/index-en.html \
                              index.html.sfish
```

Listing 4. Starfish commands in shell (command-line interface) to produce language variations of page.

The -replace option indicates a replace rather than an update mode. We would typically use also a -mode=644 option to make output files readable for a Web server.

The initial code snippet with setting up new hooks, shown above, may still be too much of a hassle to be inserted in

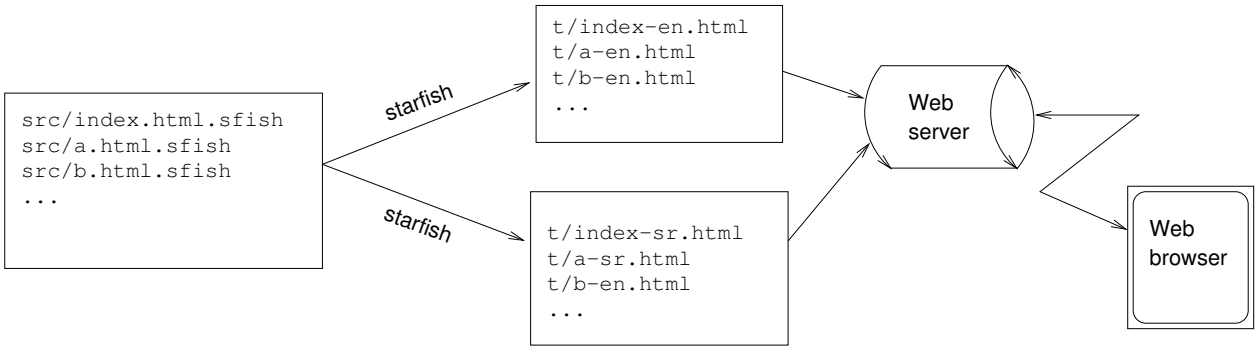

Fig. 1. System Overview of the Multi-lingual Web Site Example

each HTML source file, so we can add it to the special `starfish.conf` file, which is typically executed at the beginning of each Starfish file.

B. Starfish Courseware Implementation

The courseware single-source publishing solution is implemented in a very similar way as the multi-lingual Web site. However, in practice it does use somewhat different tags. First, most of the content material prepared goes into lecture notes, and we would prefer to include it without any particular tags. On the other hands, the slides material is only sometimes added at some particular places, so we will use the tag `:slide` for one line, or the general tag `<slide>...</slide>` for material to be included exclusively in slides. We would also define tag `:sl,l/<sl,l>...</sl,l>` for material to be included in both slides and lecture notes. The Starfish snippet which should be added to the beginning of the `.tex.sfish` file, or in the configuration `starfish.conf` file, to set up these kind of tags is shown in Listing 5.

```
<? if ($Star->{OUTFILE} =~ /-notes.tex$/) {
    sfish_add_tag('sl,l',  'echo');
    sfish_add_tag('slide', 'ignore');
elsif ($Star->{OUTFILE} =~ /-slides.tex$/) {
    sfish_add_tag('sl,l',  'echo');
    sfish_add_tag('slide', 'echo');
    sfish_ignore_outer;
} !>
```

Listing 5. Starfish code for setting up tags for slide-notes separation in LaTeX courseware contents.

Sometimes we may want to be able to include slide LaTeX content which shows in slides, and in the notes it is nicely framed as a slide, and not included simply as itemized content. We may want also to introduce only a general tag `<slf>...</slf>` to indicate such content. In order to frame and properly set up the content in the lecture notes, we define LaTeX environment `\begin{slide}...\end{slide}`, and add a hook using a lower level function in Starfish based on regular expression matching as shown in the snippet in Listing 6.

```
<? if ($Star->{OUTFILE} =~ /-notes.tex$/) {
    $Star->addHook(qr/%?<slf>\n?((?:.|\n)*?\n)
                      %?<\/slf>\n?/x,
      sub{"\\begin{slide}$_[2]\\end{slide}\n"});
elsif ($Star->{OUTFILE} =~ /-slides.tex$/) {
    $Star->addHook(qr/%?<slf>\n?((?:.|\n)*?\n)
                      %?<\/slf>\n?/x,
                   sub{$_[2]});
} !>
```

Listing 6. Use of regular-expression based hooks in Starfish to create a customized `slf` tag.

The above hook setup is based on using regular expression to define a hook to capture text excerpts between tags `<slf>...</slf>`, which may or may not be prefixed with a percent sign (`%`), which is normally a comment character in LaTeX. The coursenotes hook will produce the text excerpt within the environment `\begin{slide}...\end{slide}`, while in the slides there is no need for this environment. The slide environment in LaTeX can be defined to make a note "Slide notes:" and put the slide content in a framed box of width 10cm, using the LaTeX code as shown in Listing 7.

```
% Showing slide within notes
\newsavebox{\myslidebox}
\newenvironment{slide}{\begin{lrbox}
{\myslidebox}
\begin{minipage}{10cm}\raggedright}
{\end{minipage}\end{lrbox}
\shortstack[l]{{\it Slide notes:}\\%
\fbox{\usebox{\myslidebox}}}}
```

Listing 7. Example of LaTeX code that would include slides in coursenotes associated with Listing 6.

C. Overall Architecture

The overall architecture of the web single-source publishing system is shown in Fig. 1. When moving from a typical web site into single-source publishing, it is a good practice to separate the actual source files that are directly edited from the static or dynamic files read by the web server. The site would be moved into a source directory `src/` and we would rename the files by adding the extension `.sfish`. Using a `Makefile`, we prepare recipes to generate target files in the target public directory `t/`. (Of course, the names of these directories can be arbitrarily renamed.) Based on whether the target file name is ending with `-en` or `-sr`, the text with appropriate language would be produced from the source `.sfish` files.

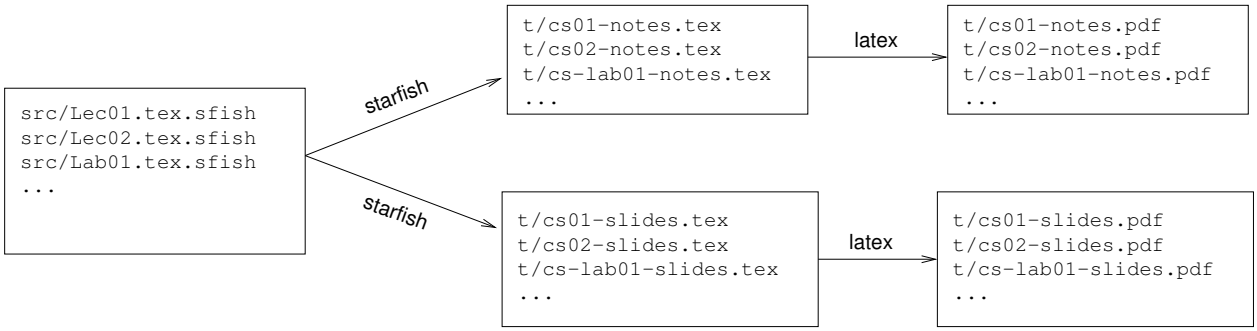

Fig. 2. System Overview of the Courseware Example

The overall architecture of the courseware example is quite similar to the web example and it is shown in Fig 2. The LaTeX starfish files do not need to be separated necessarily in a different directory, but it is a good practice to put target generated files in a temporary directory such as `tmp/` to avoid accidentally editing them instead of the source files. The single-source files have typically an extension `.tex.sfish`, and the final rendered files are PDF files produced form the target LaTeX files.

V. Conclusion

In this paper we first show by two examples, one in a multi-lingual web site creation and maintenance, and another in a courseware content development, the need for an open-system single-source document publishing system. While the single-source document publishing has been known for a number of years, even tracing to Knuth's work in 1980's, so more than 40 years ago, it has been mostly implemented as a closed-system with specific target formal languages. In our examples, we show that some modern applications, such as web site development for several languages in parallel, or courseware development for multi-modal content delivery, require a more flexible open-system approach.

Next, we show how an existing framework aimed at the universal preprocessing and text-embedded programming (UPTEP) can be used to effectively put together such open-system single-source publishing systems. Starfish is an actual open-source implementation of an UPTEP system, and using its standard hooks we show how it is used in the two examples of web site development and courseware.

As the future steps, we are looking at extending the shown examples, and into further improvement of implementation the make the process of development more efficient. The proposed UPTEP system can also be further extended using Large Language Models (LLMs) for semi-automated development of content. In particular, in the case of a multi-lingual web site, an automated translation functionality through the use of API with the services such as Google translate, would make the process even more efficient.

References

[1] S. Neilson, "Single-sourcing text — managing thoughts in multiple documents," 2015 (accessed Jan 2025), https://wordpress.stuartneilson.com/single-sourcing-text-managing-thoughts-in-multiple-documents.

[2] C. Roposh and H. Schoenrock, "Developing single-source documentation for multiple formats," in *Proceedings of the 14th Annual International Conference on Systems Documentation*, 1996, pp. 205–212.

[3] Wikipedia.org, "Single-source publishing," 2025 (accessed Jan 2025), https://en.wikipedia.org/wiki/Single-source_publishing.

[4] D. E. Knuth, "Literate programming," *The Computer Journal*, vol. 27, no. 2, pp. 97–111, 1984.

[5] J. Bentley and D. Knuth, "Programming pearls: Literate programming," *Communications of the ACM*, vol. 29, no. 5, pp. 384–369, 1986.

[6] Wikipedia.org, "Literate programming," 1984 (accessed Jul 1, 2020), https://en.wikipedia.org/wiki/Literate_programming.

[7] D. E. Knuth, *The TEXbook*. Reading, MA, USA: Addison-Wesley, 1986.

[8] N. Walsh and R. L. Hamilton, *DocBook 5: The Definitive Guide: The Official Documentation for DocBook*. "O'Reilly Media, Inc.", 2010, http://www.cs.unibo.it/~cianca/wwwpages/dd/Docbook.pdf.

[9] oxygenxml.com, "oxygen xml editor," 2025, https://www.oxygenxml.com/.

[10] V. Kešelj, "Perl module Text::Starfish and starfish: A Perl-based system for preprocessing and text-embedded programming," 2001–20 (accessed Jul 1, 2020), https://metacpan.org/pod/Text::Starfish.

[11] ——, "Perl module Text::Starfish and starfish: A Perl-based system for preprocessing and text-embedded programming," 2001–20 (accessed Jul 1, 2020), http://vlado.ca/starfish.

[12] ——, "Starfish: A Perl-based framework for text-embedded programming and preprocessing," *The Perl Journal*, June 2005.

[13] ——, "A prototype for universal preprocessing and text-embedded programming," *arXiv preprint arXiv:2007.02366*, 2020, https://arxiv.org/abs/2007.02366.

[14] ——, "A proposal for universal preprocessing and text-embedded programming (ptep) system," in *Proceedings of 21th International Symposium Infoteh-Jahorina 2022*. IEEE, March 2022.