Augmenting XHTML for Help and Documentation

Erik Wilde
Computer Engineering and Networks Laboratory (TIK)
Swiss Federal Institute of Technology (ETH Zürich)

Abstract

Providing users with help and other documentation is essential for any software targeted at end users. Authoring help and documentation in a platform-independent way is hard, because different help systems have different conventions for structuring and organizing the documents. The Help System Generator (HSG) *presented in this paper provides an easy and platform-independent way of preparing and publishing help and documentation. Using HSG, software creators can easily author, reuse, and publish help and documentation for different platforms.*

1 Introduction

The Web as an ubiquitous infrastructure makes it possible to implement an increasing number of applications as client/server-based implementations. However, it may be required to have online as well as standalone versions of an applications, which can then be used when working over the Web or being offline. Toolkits are becoming available which support the development of this kind of software [10]. If software is developed this way, help and documentation should also be available online as well as offline.

Creating help and documentation is a complex task, which often requires the collaboration of different contributors, which means that some form of coordination is required. As an added complication, it is increasingly required to create this documentation in a way which supports different publication channels, for example as a Web-based help system, as an integrated help system of the target platform, or as printed publication.

While the majority of help and documentation formats today are HTML-based (because HTML browsers are becoming a commodity in almost all computing environments), the exact way of packaging the HTML files and describing their dependencies differs between different formats. The most common formats on the operating system level are *Microsoft HTML Help* and *Apple Help* (Linux does not have such a standard help format). On the software platform level, the Java platform has two different formats, one is *JavaHelp* [9], and the other is the *Eclipse Help System* [6] for Eclipse-based applications.

If there is no help system available for an application or service being deployed, it may make sense to use static HTML pages, which can be made available on any Web server or even be browsed locally with a Web browser.

In this paper, we describe a system which solves the problem of different publication channels by using a commonly known document format (HTML) and augmenting it with a small set of documentation-specific features. The main idea behind this design is to not reinvent the wheel by designing a completely new documentation format, but to reuse the existing and widely known HTML format and only extend it where it is required. The goal of this design is to make it easy for documentation writers to create documentation, because they can leverage their existing HTML knowledge and only have to learn a very small set of documentation-specific elements.

The system described in this paper is used for the help and system documentation of the *Shared References (ShaRef)* project, where HTML and Eclipse Help documentations are generated from the source documents. The reason for this is that ShaRef provides a Web-based client as well as an Eclipse client, and for both client implementations the same help and system documentation should be available.

2 Concept and Design

The concept of the *Help System Generator (HSG)* is to use XHTML [11] (the XML-compliant version of HTML) as the basic documentation format, and then use XML Namespaces [1] to augment the XHTML structures with additional elements required for the documentation (these elements are described in Section 2.2). Because different projects may have different documentation requirements, introducing new elements is easy, as described in Section 2.3.

In addition to the documents, there must be a *Table of Contents (ToC)*, describing in which way the documents must be put together for the documentation. Starting from the documents (augmented XHTML) and the ToC (a simple XML document), different output formats can be created. For example, all help systems mentioned above need some ToC description as input, but they all require slightly different formats, which can be automatically created from the original ToC.

All help systems mentioned above provide expand/collapse features for the ToC. According to CHIMERA and SHNEIDERMAN [3], ToC views of structured content are easier to use if they provide expand/collapse features. For this reason, even the HTML version is using a JavaScript-based expand/collapse presentation of the ToC.

2.1 XHTML Restrictions

In principle, HSG does not impose any limitations on how to use XHTML, but in practice, there are two reasons why restrictions on the XHTML should be imposed in many application scenarios. The first reason is that some XHTML features may make it impossible to create self-contained documentation (a typical example are links to resources which are not part of the documentation itself, for example links to images on the Web). The second reason is that it often is a goal to have the documentation follow a number of guidelines, so that the final version has a consistent design.

In its basic setup, HSG issues warnings for elements which may conflict with the first reason above (links to external resources), so that users can check whether these links should be removed. For the layout consistency of the documentation it is impossible to define general rules, which is why they are not part of the basic setup. Adding additional rules, however, is part of HSG's principal design (as describe in Section 4.1), and adding very basic rules such as disallowing (or silently removing) all `class` and/or `style` attributes is easy to implement in HSG.

The most basic limitations are that a `<head>` element is not allowed (it will be generated if the target format is HTML-based), and that links to external resources (mainly `<a>` and `<img>`) are deprecated (it still may make sense to include a link to a Web-based FAQ or discussion forum, but this link will only work if the resulting documentation is used when being online).

For further issues regarding the question of consistent document design and related topics, Section 4.1 describes more methods how documents can be validated before processing.

2.2 XHTML Extensions

In principle, HSG pages are XHTML pages, which means they can be imported from existing (X)HTML documentation, authored using standard HTML authoring tools, or created by anybody having HTML knowledge. HSG defines additional elements, which augment XHTML in a way which is better suited towards creating help and documentation.

The following elements are supported in our prototype. It is important to keep in mind that introducing new elements (Section 2.3) is easy and an important aspect of HSG's design. Figure 1 shows examples for all elements described (all elements from the `http://dret.net/xmlns/hsg` namespace are HSG elements).

- *Cross-References:* References between documents are created using a custom element (which also supports fragment identifiers). The reason why XHTML's `<a>` element is not used is that the target names must be used for calculating the actual links for a given output format (for example, adding "`.html`" when creating HTML output). HSG provides a simple 1:1 model of embedded links because this is easy to understand with existing HTML knowledge.[1] The text of a cross-reference is generated from the target's title, which means that the link text and the target title text will automatically match.

- *Images:* Images also need special treatment, because they refer to external resources, too. Therefore, images must be included using a custom element, which makes it possible to collect information about images, create an image repository, and in the final representation generate links pointing to that repository.

- *Inclusions:* HTML does not support inclusion,[2] but for modularizing the documentation, inclusion is a very useful functionality. Inclusions can be on the document level (which means that XHTML/HSG fragments can be reused), or on the text level (including documents as pure text, for example for listings).

- *Index Entries:* Some of the target formats support searching, but for the others, having an index is essential for making the information easier accessible. Therefore, index terms can be marked up in the documentation, and this will either be mapped to a special index format for the target format (for example, JavaHelp supports indices), or to a generated index document in the target format (for example, Eclipse Help does not support indices).

[1] If required, a more powerful link model such as XLink [5] could be adopted, but then mapping it to HTML and other formats would become considerably harder.

[2] Technically, HTML (or rather SGML/XML) supports inclusion through the internal subset and external entities, but this is hardly used and only poorly supported by today's browsers.

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:doc="http:/dret.net/xmlns/hsg"><body>
  <h1>Sample Document</h1>
  <p>There must be one <code>h1</code> element giving the title. The following list describes all
     features of the current HSG documentation implementation:</p>
  <ul>
    <li>Here is a link to the complete documentation of HSG itself, demonstrating
         <code><doc:index>doc:a</doc:index></code>: <doc:a href="hsg-documentation"/>.</li>
    <li>Images may be included with the <code><doc:index>doc:img</doc:index></code> element:
        <doc:img src="vxhtml10.gif" alt="Valid XHTML Logo"/>.</li>
  </ul>
  <p>This is the document source (demonstrating the literal inclusion of text with
     <code>&lt;<doc:index>doc:include</doc:index> mode="text"&gt;</code></doc:a>):</p>
  <pre><doc:include href="sharefdoc-sample.xml" mode="text"/></pre>
</body></html>
```

Figure 1. Example HSG Document (XHTML with additional HSG Elements)

2.3 Adding Extensions

The basic set of HSG elements is rather small, but it already supports the basic requirements for creating help and documentation. If additional information needs to be included in the documents, new elements may be introduced, which can then be used to implement new functionality.

For every extension, it must be defined how it has to be used, and how it is mapped to the required output formats. For example, an extension to add margin notes for printed documentation may be added, but since margin notes are not that well-known in HTML-based formats, it may be decided to simply ignore these notes for HTML-based outputs.

HSG is currently implemented in XSLT, which is is ideally suited to support the addition of extensions. Figure 2 shows how document processing is implemented in HSG and can be easily extended, and this is described in the overall context of the HSG implementation in the following section.

3 System Architecture

The XML documents consisting of XHTML and HSG extensions are the main source of documentation for the generated output. In addition, images and included text files are also part of the documentation. These three types of documents (XML, images, text) are collected in different repositories, and since they are individual resources, they can be managed using a standard access and version control system, for example CVS or Subversion. Working on the documentation then is done by a standard workflow of checking out documents, updating them, and then committing them (Subversion is the best choice here because it supports atomic commits for multiple resources).

When documentation is generated from the resources, the XML documents are processed, it is checked that the referenced images and text files actually exist, and then the resulting documentation is written to a separate location. While the included text documents have been integrated into the generated documentation (and thus need not be shipped as individual resources), the images are still referenced in most output formats (most notably in any HTML-based format) and thus have to be included in the generated documentation as individual resources.

As a result of this architecture, integrating HSG-based documentation into existing access and version control systems can be done easily, and working on the documentation can be done consistently with working on other resources (such as program code). Generating the documentation can be included as one step in building a release, but also should be done periodically to check the documentation files for errors and inconsistencies.

4 Implementation

XHTML as the foundation of HSG has been chosen because it can be processed using XML technologies, in particular *XML Transformations (XSLT)*, which for HSG is used in its most recent version XSLT 2.0 [8]. Generally, XML is the foundation for a large variety of document-oriented applications, and using an XML-based format is the best way to keep the system as open as possible.

When switching to HSG, it is possible to migrate existing documents. If they use some kind of XML-based vocabulary, they can be mapped to XHTML. If they use HTML (i.e., they are not XML), tools like *HTML Tidy* can be used to clean up the HTML code and transform it into XHTML.

4.1 Validation

As a first step before transforming document to a target format, they are validated. Validation in this case does not mean validation against a DTD or an XML Schema, it simply refers to the general process of testing whether the documents conform to a given set of constraints.

As a first step, the documents must adhere to the XHTML restrictions described in Section 2.1, and all HSG elements found in the documents must be well-known and supported. Also, any rules for these elements must be adhered to, as an example it is required that a cross-reference points to an existing document. These checks are necessary to make sure that the set of documents can be consistently processed.

As a second step, additional constraints may be enforced. These can either be formulated in additional XSLT code, or it is possible to use a higher-level language, such as *Schematron* [7], which would be well-suited to define additional constraints. An example for a complete set of constraints are the *Web Content Accessibility Guidelines (WCAG)* [4], which are increasingly asked for when creating documentation. It should be kept in mind, however, that WCAG cannot be checked in a fully automated process, but at least parts of the checking can be automated.

4.2 Transformation

After validation of the input documents, they are transformed into the required output format(s). XML Namespaces and XSLT provide excellent support for this kind of processing, and Figure 2 shows the core fragment of the transformation code.

In this example, it is shown how XHTML elements are simply copied (this is the code for the HTML output format), while unknown HSG elements trigger an error message and are ignored. HSG's output formats are mapped to XSLT *modes*, which provide a natural way for dealing with different ways of treating the same kind of nodes. The code for processing the supported elements described in Section 2.2 is not shown in this example, but it follows the pattern of the template processing unknown HSG elements.

Since XSLT 2.0 has powerful new functions for accessing input files (in particular, any kind of text file can be included and even processed with tokenization functions), the transformation step is not limited to XML resources only, but can include any kind of resource which is available through XPath's and XSLT's functions for interacting with the execution environment.

For the HSG elements described in Section 2.2, there are templates which implement the mapping of these elements to the supported output formats. For any additional HSG elements (as described in Section 2.3), templates have to be implemented which transform these elements. Because of XSLT's template matching algorithm, this kind of behavior can be very elegantly implemented in imported stylesheets, without the need to touch any of the original HSG code.

The only problem discovered with XSLT 2.0 so far is that there is no function to return the current mode, so there is no easy way of implementing templates for two modes which only in few places implement different behavior depending on the current mode.

```
<xsl:template match="html:*" mode="html">
  <xsl:copy>
    <xsl:apply-templates select="node() | @*"
                         mode="html"/>
  </xsl:copy>
</xsl:template>
<xsl:template match="@*" mode="html">
  <xsl:copy/>
</xsl:template>
<xsl:template match="doc:*" mode="#all">
   <xsl:message>unknown element</xsl:message>
</xsl:template>
```

Figure 2. XSLT Code for Processing XHTML and HSG

5 Discussion

The system presented in this paper has been designed and implemented because of the documentation needs of a concrete project. The design is generic, so that it can be used for a variety of help and documentation applications. The current design and implementation supports a small set of documentation-related elements, but because the design is extensible and extensions can be implemented using a widely known language, it is easy to adapt the system to new requirements.

5.1 Related Work

There are many commercial products for the creation of concrete output formats for various help systems. Many of these products also support printed documentation, which HSG only addresses as future work (Section 5.3). The limitations of many products are that they are not portable across development platforms, are not extensible with additional documentation elements, and are not extensible with project-specific documentation constraints for producing validated documentation.

From the open source software side, the most popular projects are *Texinfo* [2] and *DocBook* [12]. Texinfo uses a text-based input format (no markup language) and C programs for converting the text-based input into a number of different output formats, most notably emacs info files, HTML, and $\TeX$ for printing documentation.

DocBook is an XML-based documentation format. There are stylesheets available for producing different help system formats from DocBook documents. DocBook is a rather complex format, and while many software developers have at least a working knowledge of HTML, the same cannot be said about DocBook.

5.2 Contributions

The major features of HSG are its extensibility in terms of additional documentation elements, and in terms of document validation. While these features require a certain amount of coding, they fit well into the design and implementation of HSG, and are rather easy to implement for experienced XSLT developers.

The advantage of HSG over Texinfo is that HSG is built on existing standards such as HTML, XML Namespaces, and XSLT, which makes it easy to extend HSG. The advantage of HSG over DocBook is its much simpler and widely known document model, which is only extended with a few documentation-specific elements.

5.3 Future Work

While HSG currently only targets HTML-based output formats, producing printed documentation could also be implemented. Using XSLT for transforming the documents to XSL-FO and then produce PDF would be the most obvious way, and there are a number of existing implementations for this kind of document preparation.

Creating documentation currently is done with no specific editing tool or support (we do use an XML editor, though), and while basic (X)HTML editors could be used, it would probably make more sense to use an extensible XHTML or XML editor, which could be extended to provide editing support for the HSG documentation elements as well.

6 Conclusions

The system presented in this paper can be used to produce help and documentation based on XHTML and a small number of additional documentation elements. Using this system, it is easy to create platform-independent documentation, which can then be used to produce different output formats for existing help systems.

References

[1] T. Bray, D. Hollander, and A. Layman. Namespaces in xml. World Wide Web Consortium, Recommendation REC-xml-names-19990114, January 1999.

[2] R. J. Chassell and R. M. Stallman. *Texinfo: The GNU Documentation Format*. GNU Press, Boston, Massachusetts, September 1999.

[3] R. Chimera and B. Shneiderman. An exploratory evaluation of three interfaces for browsing large hierarchical tables of contents. *ACM Transactions on Information Systems*, 12(4):383–406, October 1994.

[4] W. Chisholm, G. Vanderheiden, and I. Jacobs. Web content accessibility guidelines 1.0. World Wide Web Consortium, Recommendation WAI-WEBCONTENT-19990505, May 1999.

[5] S. J. DeRose, E. Maler, and D. Orchard. Xml linking language (xlink) version 1.0. World Wide Web Consortium, Recommendation REC-xlink-20010627, June 2001.

[6] K. L. Halsted and J. H. Roberts. Eclipse help system: An open source user assistance offering. In K. Haramundanis and M. Priestley, editors, *Proceedings of the 20th Annual International Conference on Computer Documentation*, pages 49–59, Toronto, Canada, October 2002. ACM Press.

[7] International Organization for Standardization. Information technology — document schema definition languages (dsdl) — part 3: Rule-based validation — schematron. ISO/IEC 19757-3, February 2005.

[8] M. Kay. Xsl transformations (xslt) version 2.0. World Wide Web Consortium, Working Draft WD-xslt20-20050915, September 2005.

[9] K. Lewis. *Creating Effective JavaHelp*. O'Reilly & Associates, Sebastopol, California, June 2000.

[10] R. Merrick, B. Wood, and W. Krebs. Abstract user interface markup language. In K. Luyten, M. Abrams, J. Vanderdonckt, and Q. Limbourg, editors, *Proceedings of the Workshop on Developing User Interfaces with XML: Advances on User Interface Description Languages*, Gallipoli, Italy, May 2004.

[11] S. Pemberton. Xhtml 1.0: The extensible hypertext markup language (second edition). World Wide Web Consortium, Recommendation REC-xhtml1-20020801, August 2002.

[12] N. Walsh and L. Muellner. *DocBook: The Definitive Guide*. O'Reilly & Associates, Sebastopol, California, July 1999.