Pragmatic DITA on a Budget

Joaquim Baptista
Altitude Software
Alameda Fernão Lopes, 16 – 4 andar
1495-136 Algés, Portugal
+351-21-412-9800
px@acm.org

ABSTRACT

A small documentation team working on a tight budget can now use the tool ecosystem enabled by the DITA standard to create the sophisticated content that previously required long and expensive projects. The author spent just nine person-weeks over three years to replace a custom XML system with a DITA system based on a combination of off-the-shelf software, authoring conventions, and custom scripts.

Key factors to minimize the required effort were the adoption of authoring conventions instead of specializing DITA topics and the use of folders instead of elaborate metadata. These decisions avoided the need to customize the editor and the toolchain, and simplified the creation of style sheets for the required output formats.

A small documentation team of five writers maintains over 6300 topics and publishes documents, online helps, and training materials. Open formats allow a custom script to generate reference topics from commented C# code.

Categories and Subject Descriptors

H.5.2 [**Information Interfaces and Presentation**]: User Interfaces – *Training, help, and documentation.* I.7.1 [**Document and Text Processing**]: Document and Text Editing – *Document management.* I.7.2 [**Document and Text Processing**]: Document Preparation – *Index generation, Markup languages.*

General Terms

Design, Documentation, Economics, Management.

Keywords

DITA, XML, Authoring conventions, Version control, Reuse, Topic-based writing, API reference.

1. INTRODUCTION

Over the years, Altitude Software products and documents have been localized to Portuguese (both Brazilian and European), French, Spanish, Japanese, Mandarin, and Polish.

The repeated costs charged by the translation companies for desktop publishing justified the adoption of XML, which also held the promise of avoiding the retranslation and repeated revision of the same materials.

Due to the previous experience with LaTeX[10] and Linuxdoc[22, 18], the author also knew that separating contents from presentation made writers more productive.

1.1 Proprietary XML System 2000–2004

The author considered the widely available DocBook[21] as too complex for adoption by a team without experience in XML applications, because it offered too many options for writers. Instead, Altitude Software hired a consultant in January 2000 to study and develop a custom XML system called XDoc.

The team started to use XDoc in November 2000, and the project entered maintenance in May 2001. XDoc created outputs in multiple formats, namely Microsoft Word, PDF, HTML, and Windows 95 Help, and the team enjoyed the advantages of open access to the contents.

By 2004, the team was outgrowing XDoc as the documentation set of the largest product grew to 130 documents and 16 help files. Due to initial architectural decisions, HTML outputs were taking 44 hours to generate.

In September 2004 the team was looking for alternatives. The existing solutions in the market required a significant customization effort, but the team could not afford to hire another consultant.

Table 1. Cost of the XML tools, in 8-hour days

Year	Consultant	XDoc	C# APIs	DITA	Total
2000	125†	44‡			169
2001	25	21			46
2002	14	4			18
2003		7			7
2004		3	15		18
2005				20	20
2006				19	19
2007			3	5	8
2008				1	1
Total	164	79	18	45	306

†Proposal ‡Estimated

1.2 DITA Adoption 2005–2008

In August 2005 the team started to use DITA as all the required pieces seemed to be available at a reasonable cost. Adopting DITA in 2005 was frustrating because the DITA standard de-

scribed features that the toolkit did not implement, or implemented differently in various outputs.

To avoid extensive customization efforts, the team refrained from specialized DITA topics. Instead, the team adopted writing conventions inspired by wiki systems[6] and by the team experience with XDoc.

As an upcoming standard, DITA[12, 14] enabled the development of a tool ecosystem where the team could combine off-the-shelf tools with custom-built scripts. The team expects to benefit from new DITA tools as they mature while avoiding the cost of migrating the contents to new formats.

Up to April 2008, the effort required to adopt DITA tools has taken about 9 person-weeks, which compares favorably with the estimated 14 person-weeks required just to manage XDoc between 2000 and 2002 (Table 1).

2. DOCUMENTATION SET

Documents are published monthly as PDF, HTML, and CHM helps. Documents are available on different Web sites to employees, partners, and customers. Customer Assistance burns CDs with the documentation on request.

2.1 Technical Documents and Helps

The documentation set is composed of several large books aimed at different audiences, namely contact center supervisors, system administrators, solution designers, and developers. Developers use different APIs or the Altitude Scripting Language. Large documents are complemented by many smaller documents for optional components of the product and for integrations with third-party products (Table 2).

Table 2. Documentation set

Kind	Documents	Pages	Topics
Main	31	1540	2199
Developer	14	2101	3448
Training	18	1410	1297
Helps	18	—	1937
Total	100	5041	8881

Online helps are delivered as part of the software and often reuse topics from the documents. API references are also delivered as autocompletion and embedded help in integrated development environments.

2.2 Training Documents

The team started to develop training materials in 2007 following the guidelines of Ruth Clark[2].

The team developed a set of DITA topics for lesson plans, trainer instructions, slide presentations, exercises, and solutions to selected exercises. These topics are published in a set of 17 documents that support 6 standard training courses, with materials for trainers and trainees (Table 3).

Trainers receive handbooks with everything except the reference materials. Omitting the reference materials from the handbook cuts the size of the printed document. Auxiliary books contain the lesson plans and instructions for setting up the software environment.

Table 3. Topics in training materials

Topic kind	Topics	Slides	Student	Trainer
Trainer plans	65		—	handbook
Lesson plans	15		guide	handbook
Presentations	105	723	guide	handbook
Reference	657		guide	—
Exercises	99		exercises	handbook
Solutions	39		solutions	handbook
Total	960	723	10 books	7 books

Opening the HTML table of contents of a trainer handbook is a convenient way to deliver courses. Handbooks allow trainers to show slides (using Slidy), exercises, and solutions.

Trainees receive course guides that include slide presentations supported by topics from documents, with matching documents for exercises and solutions. Exercises printed as A5 booklets are less intimidating and far more convenient than hunting for exercises inside 300-page student guides.

Although trainees receive printed books, they also receive the same materials as PDF and CHM helps. Both formats offer convenient ways to search for specific information and convey color in a few exercises where color is important.

3. OFF-THE-SHELF SOFTWARE

The DITA Open Toolkit[5] is a mandatory component of DITA[12, 14]. It is just too difficult to re-implement an evolving toolkit for an evolving standard, meaning that the team had no choice but to cope with the complexities of Java distributions, Java build systems, and miscellaneous libraries.

The toolkit provides the means to transform DITA maps into PDF documents, CHM help files, and HTML files. However, the team still needed an authoring editor, a better PDF output, a means to store and share DITA files and, later, a means to generate slides.

3.1 Authoring Editor

The authoring editor is the factor that most influences the productivity of writers.

There are many advanced text editors aimed at developers with features such as syntax highlighting, navigation, and text templates. These editors were adequate for SGML[8] applications, such as Linuxdoc[18, 22], where many tags could be omitted. With XML applications that use complex element structures such as DITA, writing using a text-based editor becomes painful. Adopting such an editor would have hindered the productivity of writers and might even have prevented the less technically minded writers from writing.

To write new text as opposed to encoding existing text into XML, writers need an authoring editor that shields them from the need to manually manipulate most tags. The editor must support writing in terms of sections, paragraphs, and lists. Writers should only notice tags for more complex structural manipulations and, ideally, authoring editors should help writers with these tasks as well.

Authoring editors have been in the market since 1988[19], but these editors require extensive customization to support a specific DTD. Adopting such an editor requires either hiring experienced

consultants or a significant investment in learning and customizing the editor.

Syntext Serna[20] was one of the first editors to support WYSIWYG[1] authoring of DITA topics out-of-the-box, thus avoiding the cost of customization. The team has also benefited from an evolving product because of the increasing adoption of DITA.

3.2 PDF Output

Version 1.2 of the DITA Open Toolkit included a proof-of-concept PDF transform using the freeware Apache FOP as a XSL-FO[11] processor. However, the PDF transform did not satisfy the needs of the team and would have required a significant effort to improve, in the form of weeks or even months spent tinkering with XSLT[7] and XSL-FO, in addition to the cost of becoming proficient with both technologies.

RenderX had a demo PDF transform that required their proprietary XSL-FO processor XEP. The RenderX PDF transform was good enough to use as a starting point, although it still required some customization. In a sense, the XEP processor was the non-free part of the DITA Open Toolkit.

3.3 Source Control

In XDoc, writers worked with a file at a time and kept dated backups of important milestones in a shared folder.

Migrating to DITA and topic-based writing meant that writers would manage more files, would work with several XML files at a time, and that conflicts between writers trying to update the same file would become common. Therefore, the team needed to adopt a better form of source control.

The combination of the open-source tools Subversion[3] and TortoiseSVN[9] has allowed less technically-minded writers to successfully use source control. As a front-end, Tortoise\-SVN instantly shows the changed files and, with a menu selection, the changes within each file (Table 4).

Subversion seamlessly merges the work of different writers in different files and even in different parts of the same file. When Subversion signals a conflict, writers typically have to manage a real writing conflict.

Subversion manages versions of whole trees of files instead of individual files, and thus it is especially suited to capture the state of the documentation set at different points in time. Subversion allowed the team to recover lost files in a few cases where writers overwrote files by mistake.

Subversion is also used to manage the changes to the DITA Open Toolkit and to distribute among team members the toolkit, Java libraries, and custom scripts.

3.4 Slides Output

In 2007, training delivery required slides. The team wanted to author simple slides in DITA instead of using proprietary tools and opaque file formats.

To deliver the slides, the team selected the W3C package Slidy[16, 17]. Trainers only need a Web browser to display slides built for Slidy.

[1] What you see is what you get.

Table 4. Documentation set in Subversion

Kind	Item	Subtotal	Total
DITA files	Topic	113	
	Concept	361	
	Task	179	
	Reference	5571	
	Ditamap	131	6355
Image files	GIF, JPEG	1894	
	SVG	78	
	Visio	245	2217
Other files	HTML	11	
	Textual	47	
	Binary	2	60
Folders	Main topic	170	
	API reference	224	
	Image	114	
	Other	27	535
Total			9167

4. AUTHORING CONVENTIONS

DITA elements[13] are classified as structure, block, or inline. Structure elements contain block elements, which contain inline elements. Text is only allowed inside block or inline elements.

The DITA `conref` mechanism is used sparingly to avoid maintenance issues created by *spaghetti*[4] text. Also, the use of `conref` can affect the output.

4.1 Structural Markup

For historical reasons the team writes most topics as DITA references instead of plain topics or concepts. Tasks are also rare in the Altitude Software documentation.

The team ensures that elements `<body>` and `<refbody>` contain a sequence of sections or examples, which in turn contain a sequence of block elements (and never text directly).

Topics also feature `<shortdesc>` descriptions when desirable and these double as the first paragraph of a topic. Topic metadata is restricted to `<indexterm>`, but the team prefers to use inline index entries when possible.

Having topics in separate files simplifies their reuse in different maps. Also, files with nested topics generate different navigation structures for online and PDF outputs.

4.2 Block Markup

Block markup always starts a new line, and the team avoids nested block elements such as lists inside lists.

The allowed block elements are paragraph, note, ordered and unordered lists, description list, table, pre-formatted lines and code block. Description lists do not have headers nor sequences of terms or descriptions. The element `<lines>` is used in table cells for visual formatting.

Display images appear inside figure elements. Writers specify image variants for online and PDF outputs using the attribute `otherprops` with the values `"online"` and `"print"`.

4.3 Inline Markup

Inline markup only appears inside block elements. The allowed inline elements are `<codeph>`, `<apiname>`, `<filepath>`, `<userinput>`, `<systemoutput>`, `<uicontrol>`, and `<image>`. The team writes interface paths using phrases `<ph>` with embedded text and `<uicontrol>` elements.

Inline `<indexterm>` elements allow page-level precision in the PDF output.

Some linking structures cannot be captured by related links and require `<xref>` elements. For example, expert tables that link to a hundred topics or links that require a contextual description.

4.4 DITA Maps

DITA maps have metadata, nested topic references, and relationship tables. The metadata includes the dates of first publication and last revision, the product name, the product version, and the document series.

Writers often create DITA maps for closely coupled sets of topics and then reuse these auxiliary maps in document-level maps.

4.5 Slide Markup

Slide presentations are DITA reference topics. The title and the first section become the first slide. The remaining sections become slides or handout notes, depending on whether they have a title. Slides mostly have paragraphs, bullet lists, and images.

Having a single topic with all the slides of a presentation avoids the need to simultaneously name and edit multiple files, as well as the corresponding Subversion overhead as slides are split, merged, and reordered.

4.6 Folders and Filenames

Writers use folders instead of metadata to group topics by audience or by component.

To avoid issues with case-sensitive filesystems, the names of files and folders are restricted to lowercase letters, digits, and hyphens. Prefixes group files in larger folders, filenames with *intro* are introductory topics, and filenames ending with *slides* contain slide presentations. Folders named *ref* typically contain reference information, such as API functions.

5. SOFTWARE CUSTOMIZATION

Writers work on a local copy of the documentation set, using Syntext Serna to edit topics and maps. Generating outputs is a simple drag and drop operation. Writers use Subversion to commit completed work and retrieve the work of other writers.

Writers can also use text editors or custom scripts to perform special work. Throwaway scripts were created to convert XDoc content to DITA and are still created for one-off tasks.

Adopting DITA required about 10 weeks of software customization between May 2005 and April 2008 (Table 1). As the writing conventions limited the allowed input they also limited the customization effort.

5.1 DITA Toolkit Customization

For online outputs, post-processing avoids the cost of fully understanding the DITA Open Toolkit and the supporting technologies. Online outputs are post-processed to create HTML frames and HTML slides for Slidy.

PDF output required the customization of the XSLT style sheets and some understanding of XSL-FO to modify cover pages, headers, footers, section headings, and description lists. Further customization was required to handle related topics, cross-references with page numbers, and slides.

5.2 Build With Conventions

Adopting a set of conventions for top-level folders and maps simplifies the build process. The custom script *build* runs the DITA Open Toolkit over the *dita-tmp* folders and copies the processed outputs to folders under *dita-output*.

The first word of a DITA map filename becomes a folder. For example, processing the map *altitude-vbox* to PDF generates the file *dita-output/pdf/altitude/vbox.pdf* (Table 5).

Table 5. Conventional top-level folders

Path	Description
71-dita	Source files
dita-bin	Scripts, DITA Open Toolkit
dita-tmp	Temporary DITA toolkit files
– log	Log files of DITA toolkit
– out	Output folder for DITA toolkit
– tmp	Temporary folder for DITA toolkit
dita-output	Outputs from *build*
– help	CHM helps from *build*
– html	HTML files from *build*
– pdf	PDFs from *build*
– zips	Archives from *build-zips*
dita-output-fr	French outputs from *build*

To generate a single output, writers use drag and drop to a shortcut, such as *build-chm.bat*. To generate outputs in batches, writers use command lines.

After running *ant* with the *ditaval* filtering for online or PDF output, the script *build* post-processes the outputs. For CHM helps, *build* replaces the CSS style sheet and removes the index tab if there are no index entries. For HTML, *build* replaces the CSS style sheet, creates frames, and processes filenames ending in *slides* to become valid Slidy.

The script *build-zips* creates ten archives that simplify the distribution of documents and training materials.

5.3 C# APIs as DITA Topics

To support autocompletion in integrated development environments, C# requires that classes are documented in comments next to the code.

The script *dotnet-to-dita* parses the commented C# source code of two sets of source files and generates reference topics for classes, methods, and properties. Topics include the formatted code signature as well as descriptions lifted from comments in the code.

The script illustrates the major advantage of having contents in an open format by generating 1683 topics (973 pages) of reference information describing four API variants in C#, Java, and C.

The script exploits the structure of the specific APIs to explain auxiliary types within the API elements that use them, reducing the number of pages that readers must consult to understand each

feature. This processing resembles function inlining as performed by compilers[1].

The script generates indexes in a semi-automated process. First, the script generates possible 1-level or 2-level index entries corresponding to API elements. Writers then edit the index by maintaining lists of exceptions detailing which 1-level entries should become 2-level entries and which index entries should be left out of the index.

Developing the script took three weeks in 2004, followed by three days in 2007 to generate DITA topics and improve the indexes from online-level to book-level quality. The time taken to edit the indexes is not included.

5.4 Auxiliary Scripts

A set of small auxiliary scripts compensate for the lack of a content management system that maintains the integrity of the documentation (Table 6).

Table 6. Custom and auxiliary scripts

Script	Lines	Language
build	328	Perl
build-zips	172	Perl
dotnet-to-dita	2541	Python
check-hrefs	71	Python
check-ditamap	15	Perl
find-ditamap	15	Perl
ditamap-refs	21	Perl

The script *check-hrefs* validates references in DITA topics to images and other elements (`<xref>` element).

The script *check-ditamap* validates whether a DITA map references existing topics, or whether a set of topics is referenced in a specific map.

The script *find-ditamap* lists all the DITA maps that reference a set of topics.

The script *ditamap-refs* lists all the topics referenced in a DITA map, either directly or indirectly through referenced maps.

6. DITA MATURITY MODEL

The six levels of the DITA Maturity Model[15] illustrate the sophistication of the team achievements while using DITA on a budget.

6.1 Level 1: DITA Topics

The initial objective in 2005 was to convert the existing documents to DITA, escaping the limitations imposed by XDoc. However, online outputs treated nested topics as a single unit, which limited the usefulness of searching and was not convenient for online delivery.

6.2 Level 2: DITA Maps

The team broke nested topics into smaller units, which required rewriting large parts of the documentation set during 2006. However, the restructured topics could be reused in different books and helps, thus lowering the overall maintenance effort.

Table 7 shows that 22% of all topics are used in more than one output (excluding outputs that, by their nature, reuse many topics). Excluding most topics aimed at developers, 46% of the remaining core topics are used in more than one output.

Table 7. Cost of the XML tools, in 8-hour days

Usage	All topics		Core topics	
1	4677	78%	1600	54%
2	1024	17%	1019	35%
3	258	4%	256	9%
4–14	61	1%	61	2%
Total	6020	100%	2936	100%

Figure 1 shows how topics are reused across all published outputs except embedded API helps. Although 322 help topics are published only once, one topic manages to get published in 56 different outputs!

6.3 Level 3: Specialization and Customization

The specialization of DITA topics proposed by Level 3 is not really required for a small team, at least in the software domain.

Teams must have a critical size to justify the investments needed to customize the editor and the toolchain. Also, the team has learned from XDoc that excessive customization hinders the ability of the toolchain to tackle new challenges as the documentation set evolves.

Instead, editors enforce authoring conventions, thus compensating for the lack of specialization. Over time, the team has refined its structural and quality guidelines to improve different parts of the documentation set. These practices correspond to the process part of Level 3.

6.4 Level 4: Automation and Integration

Although the team has only made small investments to complement Subversion with auxiliary scripts, Altitude Software is already following several practices classified as Level 4.

In 2007, the team reused document topics as supporting materials for the student guides (Table 3). Also, the marketing department reuses two specific documents, using XSLT transformations to integrate information into a larger Marketing document.

Since 2004, the script *dotnet-to-dita* reads commented C# code to create regular API documents.

Training presentations were translated from the DITA source using off-the-shelf translation memory tools. Slides of the most

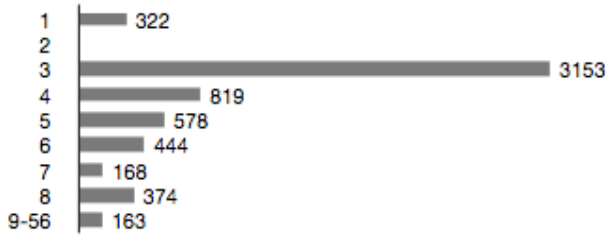

Figure 1. Reuse of topics in all outputs

popular courses are currently translated to French, and translations to other languages are planned.

6.5 Level 5: Semantics on Demand

Reaching Level 5 seems to require significant effort in integrating disparate information sources in an automated way. In the future, the team will consider whether to integrate the documentation set with non-DITA information sources, such as the company wiki[6].

7. CONCLUSIONS

A team with the discipline to follow strict authoring guidelines can limit the customization effort required to effectively use the DITA standard.

Using the DITA Open Toolkit, a suitable editor, and Subversion, a small team on a budget can aspire to the kind of content sophistication that ten years ago was limited to experimental environments or very large companies that could pay for custom projects.

8. ACKNOWLEDGEMENTS

The author would like to thank Ricardo Amador for developing XDoc and all current and past members of the documentation team for their support.

9. REFERENCES

[1] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 1986.

[2] R. C. Clark. *Developing Technical Training*. Addison-Wesley, Reading, MA, 1989.

[3] B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. *Version Control with Subversion*. O'Reilly, first edition, June 2004.

[4] E. W. Dijkstra. Go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, March 1968.

[5] DITA Open Toolkit Project. DITA open toolkit. http://dita-ot.sourceforge.net/.

[6] N. Drakos. Altitude's user-centered wiki becomes a mission-critical tool. Toolkit Case Study G00150519, Gartner, August 2007.

[7] M. Fitzgerald. *Learning XSLT*. O'Reilly, November 2003.

[8] C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, Oxford, 1990.

[9] S. Küng, L. Onken, and S. Large. *TortoiseSVN: A Subversion client for Windows*, version 1.4.7 edition, November 2006.

[10] L. Lamport. *LaTeX User's Guide and Document Reference Manual*. Addison-Wesley Publishing Company, Reading, Massachusetts, 1986.

[11] D. Pawson. *XSL-FO: Making XML Look Good in Print*. O'Reilly, August 2002.

[12] M. Priestley. DITA XML: A reuse by reference architecture for technical documentation. In *Proceedings of the 19th annual international conference on Computer documentation*, pages 152–156. ACM SIGDOC, October 2001. Santa Fe, New Mexico, USA.

[13] M. Priestley and J. Hackos, editors. *Darwin Information Typing Architecture (DITA) Language Specification v1.0*. OASIS Open, May 2005.

[14] M. Priestley, G. Hargis, and S. Carpenter. XML-based technical documentation authoring and publishing architecture. *Technical Communication*, 48(3):352–367, August 2001.

[15] M. Priestley and A. Swope. DITA maturity model. Whitepaper, JustSystems, 2008.

[16] D. Raggett. HTML Slidy: Slide shows in XHTML. http://www.w3.org/Talks/Tools/Slidy/, 2005.

[17] D. Raggett. Slidy: A web based alternative to Microsoft PowerPoint. In *Proceedings of XTech 2006*, Amsterdam, Netherlands, May 2006.

[18] C. Schwarz. Flexible formatting with linuxdoc-sgml. *Linux Journal*, 1(18), October 1995.

[19] B. Shiff, P. Sharpe, and R. Spencer. *SoftQuad Author/Editor: user's manual*. SoftQuad Inc., Toronto, Canada, 1990.

[20] Syntext. Serna. http://www.syntext.com/.

[21] N. Walsh and L. Muellner. *DocBook: The Definitive Guide*. O'Reilly, first edition, October 1999.

[22] M. Welsh and G. Hankins. *SGML-Tools User's Guide*, November 1996.