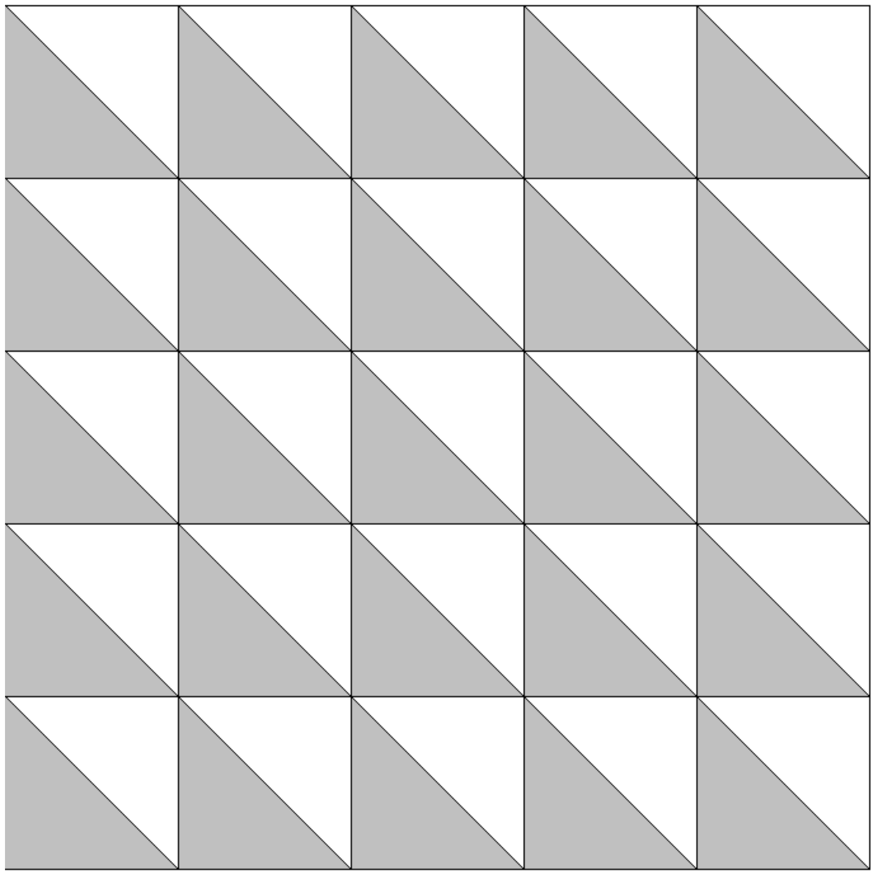
Technical
Report

Research Triangle Park, North Carolina

Architecting Content for DITA

TR 29.xxxx



Julio J. Vazquez
D&ID Marketing & Sales
May, 2002

Abstract

This article describes how an author can apply the DITA architecture when developing information for a product. It lists the information types defined in DITA and a reasonable approach for analyzing content to fit the types. The article does not teach the structure of the language but focuses on the concepts behind the architecture and how to change your paradigm to embrace the architecture. There are a few examples included that are based on the language as it stands today.

ITIRC Keywords

- Darwin Information Typing Architecture
- DITA
- XML
- Information architecture
- Information typing
- Classifying information
- Typing articles
- Architecting content

About the Author

Julio Vazquez (vazquezj@us.ibm.com) has been an information developer with IBM for over 15 years. He has written a few articles about IBM's SGML implementation, IBMIDDOC. He has contributed to the DITA workgroup over the past two years and also actively participated in testing an implementation of DITA. He holds an Associate's degree in Electrical Technology and a Bachelor's degree in Computer and Information Systems.

Contents

I.Introduction 1

II.The information types 1

A.Concept 1

1.What does a concept article look like? 2

B.Task 2

1.What does a task article look like? 3

C.Reference 4

1.What does a reference article look like? 4

III. Why analyze? 5

A.Put yourself in the reader’s place 5

B.Keeps article focused 5

C.Improves writing style 5

IV.How to analyze content 6

A.Read 6

B.Ask questions about material you’re developing 6

C.Strip away what doesn’t fit 6

V. At what cost purity? 6

VI. When should I start analyzing information? 7

VII. What if my information doesn’t fit the basic types? 7

VIII. You mentioned specialization; what’s that? 7

IX.How do I type existing information? 7

X. What comes next? 8

XI. References 8

I. Introduction

The Darwin Information Typing Architecture (DITA -- pronounced dit-uh) is IBM's XML-based solution for the creation of articles about products or services. (In this paper, articles are short, descriptions about tasks, concepts, or other information about hardware or software.) The architecture was developed over the span of two years and the purpose is to classify information articles to produce a cleaner and more efficient writing style. Other goals of the architecture include:

- ✍ The ability to preserve information encoded in legacy SGML
- ✍ Ease of use for the writers
- ✍ The ability to easily extend the base language to support types that are relevant to a writing group
- ✍ The creation of a body of information that can be queried when such technology exists
- ✍ The ability to enable reuse simply
- ✍ The ability to create processing software and style sheets that will support multiple output targets and navigation schemes easily

At the time of this writing, not all of the goals have been realized, but progress is being made to meet those goals. The XML underpinnings of the language show great promise for processing the marked-up content and easily extending the language.

The purpose of this paper is to help you, as an author, evolve your authoring paradigm to encompass DITA. To do this, you must move from a hierarchical development approach to one based on typed articles that can be developed in any order and later assembled using some sort of navigation mapping. The details of the DITA language and navigation mapping are left to other documents that will be mentioned in the References section. The main thrust is to get you to think, "What types of information do I need to develop here?" How to assemble that information becomes a problem to be addressed.

II. The information types

DITA classifies information into three basic types: Concept, Task, and Reference. The design's intent is for the author to use specific questions about the information being developed to determine the correct type for the information. The expectation is that the focus on information type before developing the information will enable the writer to create a succinct, small article that may later be used as part of a larger construct, if needed. By focusing closely on the information type, the author will write articles that are shorter (generally on the level of a browser screen or two) and more easily digested by a reader. Typing information that has already been written is more difficult, but entirely possible. I will start by addressing how to type new information and then discuss how to apply the architecture to existing information to type it correctly.

The following sections describe each information type and the questions a writer should ask to determine which type to use for any situation. Typing may be difficult in some cases, but a concerted effort to type the information can reap many benefits for both the author and the reader.

A. Concept

Basically, the concept article takes the unknown and attempts to make it known to the reader. The author does this by relating the new concept to things the reader already knows, sometimes using analogies to draw the reader to the conclusion.

Concept articles attempt to answer the questions:

- What is this?
- Why is it important to me?
- How does this relate to other things?
- When should I use this?

This paper is an example of a concept article because it does not show you how to perform any actions or contain information that you may want to look up at a later date.

1. What does a concept article look like?

Here's an example of what a concept article would look like if you were to peek at the XML code. Notice that there are a lot of elements and that the information itself doesn't really tell you how to do anything; it just tells you about the subject of the article.

```
<concept id="sqlprofdescription">
<title>SQLJ profiles</title>
<conbody>
<p>In addition to .java files, the SQLJ translator
generates profiles. These files provide details about
the embedded SQL operations in the SQLJ source code,
including types and modes of data being accessed.
Typically, one profile is generated for each connection
context class. A connection context class usually
corresponds to a single database schema. The profiles
are used by the run-time environment to provide details
about the database schema.</p>
<p>Profiles can be customized to make use of vendor-specific
features.</p>
<p>The VisualAge for Java SQLJ tool creates profiles
in your project resources directory. These binary
files have a .ser extension.</p>
</conbody>
</concept>
```

B. Task

Task information tells the reader how to accomplish something. There may be some discussion that puts the task into a context, but the real purpose of a task is to get you from point A to point B with little additional information. The only question that task information answers is, "How do I do this (or get this done)?"

DITA's task type contains elements that support the idea of getting the task done without side discussions. Without going into too much detail, the base structure minimally contains these elements:

- Context - This element contains information that gives the reader an idea of when or why the reader is doing the task. The information contained orients the reader to the task at hand. If there is other information that the reader may need to understand to accomplish the task, that information is referred to but not contained here. The intention is to get readers accomplishing the task as quickly as possible and not to weigh them down with facts not directly related to task completion.

✍ Prereq - This element contains all of the items a reader may need to carry out the steps of the task. Any tools, papers, books, information, or other materials needed to complete all the steps in the task should be listed in this section. The element could also state what other tasks must be completed before this task can be completed successfully. The prerequisites should eliminate the need for the reader to locate items needed to complete the task in the event the reader does not read the entire task first.

✍ Steps (and substeps) -- This is the portion of the article in which you tell the reader what to do. Break the task down into as many steps as you need to ensure the reader is successful. A good rule of thumb is one action per step. If a particular step has a number of actions in itself, break that step down into substeps. It is best to watch someone performing the task and describe the step-by-step process.

✍ Postreq - This element encapsulates things that have to be done when the reader completes the task. Postrequisites may be items that the reader has to do to ensure the task gives the expected results. For example, after installing a specific software application, the reader may have to reboot the machine before using the application. Technically, the reboot is not part of the task of installing this software but a postrequisite for using the application.

✍ Result - The information in this element tells the reader how to determine that the task was completed successfully. There should be a description of any audio or visual clues that would signal success. An illustration of the final state could be appropriate.

1. What does a task article look like?

Here's an example of what a task article would look like if you were to peek at the XML code.

```
<task id="createsqljfile">
<title>Creating an SQLJ file</title>
<taskbody>
<context>Once you have set up SQLJ, you need to create
a new SQLJ file.</context>
<steps>
<step><cmd>In a text editor, create a new file.</cmd></step>
<step><cmd>Add your Java source code and SQLJ statements.</cmd></step>
<step><cmd>Save your file with an .sqlj extension and close the editor.</cmd></step>
</steps>
<postreq>
<p>Once you have created a new .sqlj file, import the file into your project and translate the
file</p>
<p>For information on SQLJ syntax,
contact your database vendor or see <xref href="web.ansi.org/public/std_info.html"
type="external">.
</p>
</postreq>
<related-links>
<linklist type="concept"><title>Related Concepts</title>
<link href="../czover.htm#sqljsupp">
<linktext>SQLJ support in VisualAge for Java</linktext>
</link>
<link href="../czesqlj.htm#sqljemb">
<linktext>Embedded SQLJ</linktext>
</link>
</linklist>
<linkpool type="task" role="child" collection-type="family">
<link href="tzover.htm#accsqlj">
```

```
<linktext>Accessing relational data with SQLJ</linktext>
</link>>
<link href="tzimport.htm#impsqlj">
<linktext>Importing an SQLJ file</linktext>
</link>
</linkpool>
</related-links>
</taskbody>
</task>
```

C. Reference

The reference article is just a factual representation of information that could be used to help someone complete a task. As Joe Friday said, “Just the facts, ma’am.” The presumption is that the article will be accessed at random and the information within it will stand alone to impart only the facts the reader is seeking. (An article in an encyclopedia is a prime example of a reference-type article.)

Some common examples of reference articles in DITA are command syntaxes and software message explanations. Command syntaxes are only read when someone is entering a command; likewise, there is no reason to read a message explanation unless it’s sitting in front of you and you don’t understand what the message is telling you.

Reference articles often refer the reader to other articles that may contain additional information, if needed.

1. What does a reference article look like?

Here’s an example of what a reference article would look like if you were to peek at the XML code.

```
<reference id="hndl0100"> <title>HNDL0100 Format</title><refbody>
<section> <table>
<tgroup cols="4"><colspec colnum="1" colname="col1" colwidth="25*"/>
<colspec colnum="2" colname="col2" colwidth="25*"/>
<colspec colnum="3" colname="col3" colwidth="25*"/>
<colspec colnum="4" colname="COLSPEC0" colwidth="25*"/>
<tbody>
<row>
<entry namest="col1" nameend="col2" align="center">Offset</entry>
<entry colname="col3" morerows="1" valign="bottom">Type</entry>
<entry colname="COLSPEC0" morerows="1" valign="bottom">Field</entry></row> <row>
<entry colname="col1">Dec</entry>
<entry colname="col2">Hex</entry></row>
<row>
<entry colname="col1">0</entry>
<entry colname="col2">0</entry>
<entry colname="col3">BINARY(4)</entry>
<entry colname="COLSPEC0">Bytes returned</entry></row>
<row>
<entry colname="col1">4</entry>
<entry colname="col2">4</entry>
<entry colname="col3">BINARY(4)</entry>
```

```
<entry colname="COLSPEC0">Bytes available </entry></row>
<row>
<entry colname="col1">8</entry>
<entry colname="col2">8</entry>
<entry colname="col3">CHAR(16)</entry>
<entry colname="COLSPEC0">Handle </entry> /row>
</tbody></tgroup></table></section>
<properties>
<property><proptype>Bytes returned</proptype><propvalue>The number of bytes of data
returned.</propvalue></property>
<property><proptype>Bytes available</proptype><propvalue>The number of bytes of data
available to be returned. All data is returned if enough space is provided.</property>
<property><proptype>Handle</proptype><propvalue>The returned handle value to be used
on subsequent API calls when a handle parameter is required.</propvalue></property>
</properties></refbody></reference>
```

III. Why analyze?

Writing to fit the information architecture takes a lot of effort as you have to analyze what you write in terms of what would be the best container for the information. The container itself will restrict you as to the elements permitted (which means less tagging choices) so that an incorrect analysis of your information would create problems (and headaches) when it comes to developing the articles. Is it all worth the bother? The short answer is yes. The next sections give you the long answer.

A. Puts you in the reader's place

There is a lot of information in the world today. The sheer volume that most people deal with is daunting. Mixed types of information tends to be frustrating to the reader because there is generally more information displayed than needs to be read. It's more important that only the pertinent information that the reader requires is presented to the reader. Extraneous information in front of the eyes can cause the reader to miss the message for which the reader started searching. The message is practically lost in the forest of words.

B. Keeps article focused

Writing to a specific information type is liberating, not restrictive. As the writer, you focus precisely on the specifics required of the information type and nothing else. Questions of presentation don't come into play because those things are handled by downstream processes that will format the elements based on a predefined style for the information type. The writer's task now becomes one of making the content fit the information type, not the presentation medium. The type focus eliminates other concerns that may affect the style of writing.

C. Improves writing style

As alluded to in the previous section, honing in on what type of information you are developing helps improve your writing style. As you are working on a single article, you only look at the elements allowed within that particular type and you start thinking in the style dictated by the information type.

As an example, when you are writing a task article you think and write about the steps it takes to accomplish the task -- nothing else should come into play. Any side items that do come up you can note and set aside for a different article and refer to the second article from within the task. You stay focused on the information and so does your reader because they have received precisely what they need.

IV. How to analyze content

For the most part, analyzing content can be easy to do, even if it can be time-consuming. The reality is that when you look at a piece of information and ask certain questions of the information at hand, you analyze the information and decide into which of the types that piece of information fits.

A. Read

Read the information from which you are developing your article. Although this sounds ludicrous, far too often a writer may not pre-read the source that will become the final article. In today's world of copy and paste, it is far too easy to pick up a chunk of information and drop it in place without fully understanding how well that chunk fits the information type that is prevalent to the rest of the article. That one misfit could destroy the entire tone of the article and even make it confusing to your audience. If the chunk of article seems out of place, it most likely does not fit the article.

B. Ask questions about material you're developing

There are two different types of questions you can ask about the information; Is it technically accurate? Does it fit the rest of the article? For the purposes of this paper, I will ignore the technical accuracy question because that is a given; no one wants to produce information that is wrong. The real problem is how does it all fit together.

To get things to fit correctly, you must question every piece you are writing. Ask yourself:

- ✍ What is the purpose of the information?
- ✍ Am I trying to get the reader to do something or understand something?
- ✍ Am I trying to show the reader the way something should look when they use it?
- ✍ Does it fit the type of article?
- ✍ Is the tone consistent with the rest of the article?

The answers to these questions can help you figure out how appropriate it is to add that information to this specific type of article.

C. Strip away what doesn't fit

The heading of this section says it all. If what you have developed does not fit the type of information or the tone of the article, remove it. If the information needs to be somewhere in the body of information you will find another place for it -- it just may not be a fit for the current article. With today's word processing tools, you can put the information aside and bring it into another article where it would fit better with relative ease. Use the flexibility that is now at your fingertips and just say no to bloating an article with information that does not belong in that particular information type. In the long run, you will thank yourself for doing so and your readers will too.

V. At what cost purity?

Generally, most articles you write will fit either the task, concept, or reference types. There may be occasions when a hybrid seems necessary. Resist that temptation as much as possible and make a good business decision on whether you truly need to use a hybrid type. Keeping to the three information types will make it significantly easier to reuse the information. Mixing the information types together may be easier in the short term, but will make the information more difficult for the reader and that person is the final arbiter as to how effective your information is.

Having said that, realize that developing information also requires sound business judgment and if you are spending a lot of time trying to figure out where a particular piece of information fits, it is best to develop the information as best you can in one type and revisit the information when you get time to perform an appropriate analysis. Trying to complete an analysis of information should not affect a production schedule. At the same time, it is important to correctly type all the information you have developed as soon as you can after the first edition of that information so that the original intent is fairly fresh in your mind.

VI. When should I start analyzing information?

Yesterday. Seriously, there is no reason not to analyze your information to determine to which type an article belongs. Even when developing a traditional hierarchical deliverable, take the time to try and make each section a single information type. It will benefit you and your reader both now and in the future. The razor-edge it will bring to your writing will help you get your point across to the reader faster. (It will also make it easier for you to migrate to DITA when that time comes.)

Every time you write something, instead of setting up an outline, try to break down your thoughts into the different types of information described. Make a conscious attempt not to write a hierarchical structure, but a collection of separate articles. After you've written those articles, then think about how to navigate through the information. Let the information drive the navigation and not the navigation drive the information. In fact, after you've written the information, the navigation task becomes one of defining an outline to assemble the pieces together into a flow. If you really get your creative juices flowing (and think outside of the box occasionally), you can see different flows for the information and assemble your content from different readers' points of view.

VII. What if my information doesn't fit the basic types?

What I didn't mention is that there is another type. It's a generic topic (which I'll just call topic from now on) from which the other types are specialized. The topic type can be used as a wrapper for other types of information or you can specialize the topic type to reflect your information needs. (In fact, you can specialize any DITA article type.) Most of your information should fit the basic three types, but when you're stuck, use the topic type; it's generic enough to meet all your needs.

VIII. You mentioned specialization; what's that?

Specialization is taking an existing information type in DITA and using the inheritance properties of the architecture to meet your needs. The benefit of specialization is that you can leverage existing formatting rules and tool sets by having your information type resemble an existing one. The ability comes to you through the power of XML.

Specialization is the sort of thing an author wouldn't do without the help of someone who understands the architecture and the DTDs. Using specialization to create your own type is an area that is discussed at the developerWorks web site (see references) in more detail. If you're interested in specialization, take a look there and examine some of the samples that exist.

IX. How do I type existing information?

Typing existing information is a challenge. The problem lies in the fact that information developed without an architecture in mind tends to bleed types together and excising the information that doesn't fit the type may be difficult. That said, the exercise is worth the benefits you can derive from the resulting clarity of the information. So how do you go about it?

Look at the information at hand. Drill down to the lowest section in a piece and ask yourself the what the purpose of the section is. Look at the information with the following questions in mind:

- Does this information tell me how to do something? Is it just a series of steps to be performed? (Task)
- Is the section showing the reader syntax? Is the section a list of features and functions? Is it something that a reader will come back to without caring about any other information? (Reference)
- Does the section give background or overview information? Does it answer questions about the environment in which something exists? Does it expand the reader's knowledge base? (Concept)

You may not get pure answers to these (or similar) questions. If not, you should look at how to rewrite the section such that you can get the section as close to those answers as possible. It may not be possible to get existing content to fit into a single specific type, as you could if you started writing in that mode, but you should be able to get closer, which will still improve the content's clarity.

X. What comes next?

This is a good question. At the time of this writing, the XML elements that encapsulate DITA are still being fine-tuned, so they are not ready for prime-time yet. However, there is a package that contains the latest version of the language at www.ibm.com/developerworks. Download a copy of it and take a look at the samples provided and, if you're into reading these sorts of things, look at the DTDs. Glean as much information as you can from the samples.

Other missing pieces at this time deal with navigation and processing. Currently, there are a few things missing that would enable ease of printing (yes, everyone still prints what they read -- admit it and move on) and navigation through the morass of information. Those pieces will come soon enough, but depend on additional stability in the language. The good news is that the language is almost there so the focus can turn to these other items in the near future. Also, the XSL style sheets that represent the on-screen presentation are in place and look pretty good. The expectation is that at the end of 2002 IBM will have a workable XML solution for information development.

One thing that this paper has not addressed yet is meta data -- the information that describes information. At this writing, there are some initial ideas on meta data and how to enable it such that the requirements on the writer are not as daunting, but closing that hole will probably take a little more time. Once the meta data is described more clearly, navigation will be simpler because the reader would just be able to query the information to retrieve it. (Of course that search engine also has to be defined, but that will happen.)

For those who are reading this and wondered why XML schemas were not used to develop the language, there is a simple answer: the schema standard was not fully developed at the time DITA was being developed and the group who worked on developing the DTD felt it best to get the language out there and convert it from a DTD to a set of schemas when the schema standard (and the language) stabilized. This allowed the team to get a working prototype out for evaluation and correction.

XI. References

For more details about DITA and XML, look at the following:

www-106.ibm.com/developerworks/library/x-dita1/ -- this link is an introduction to DITA from an architecture viewpoint. In this article, you'll also find links to FAQs, the DITA forum and other articles about the architecture and language.

North and Hermans, *Teach Yourself XML in 21 Days* -- while not about DITA, this book gives a solid foundation on XML and understanding the relationship between the markup and the presentation. It also gives you enough information so that you should be able to interpret any DTD.

If you can get in touch with any of the following people (maybe through the DITA forum), they are extremely knowledgeable and glad to help you understand whatever is bothering you about DITA:

- Don Day, IBM Austin - I consider Don the DITA god. He's done the majority of the implementation and is an overall good guy to talk with. The best way to contact him is through the DITA forum (which is pointed to from the first reference).

- Mike Temple, IBM Rochester -- one of the team members who had the nerve to try to write about how to use the language that developed from the architecture. He's done a good job on the initial documents about the XML implementation.

- Michael Priestley, IBM Toronto -- one of the chief architects of DITA, Michael tried to keep things moving during the development. Unfortunately, he had people like me to contend with who kept wanting things to work perfectly.

- Nacia Owens, IBM Raleigh -- is writing the DITA User's Guide and the DITA Language reference with help from the DITA team. She was also kind enough to look at this article and help with examples.

Index

A

analyze information
- how much, 6
- how to, 6
- reason, 5
- when to start, 7
- why, 5

C

concept
- example, 2
- information types, 1

E

example of
- concept article, 2
- reference article, 4
- task article, 3

H

how to
- analyze information, 6

I

information types, 1
- concept, 1
- non-match, 7
- reference, 4
- task, 2

N

non-match
- information types, 7

R

reason
- analyze information, 5

reference
- example, 4

S

specialization defined, 7

T

task
- context, 2
- example, 3
- postreq, 3
- prereq, 2
- result, 3
- step, 3

typing existing information, 8

W

why
- analyze information, 5