Transcluding the Invariants

Eric Armstrong

Having settled on a strategy of composition, the next question is whether to transclude the parts of the text that change from version to version (the *variants*) or whether to transclude the *invariants*, instead—the boilerplate and other things that don't change. This article lists the advantages of the latter course of action.

> **Note:**
> Thanks to Ted Nelson, we have the word "transclude" in our vocabulary, which means "take something from somewhere else and include it in-line here". So a DITA conref is a way of transcluding material into a topic from somewhere else.

Introducing the Idea

It's second nature, I guess, to consider the small differences in things as part of the "foreground", and assume that the larger, invariant pieces form the "background" (backdrop, context) against which the actors move in which the foreground. In this case, the foreground is a conref, while the background is the context into which the conref fits.

So when there is a small bit of boilerplate that needs to be included in multiple files, it's easy to consider that text the foreground, make a conref out of it, and transclude it wherever it's needed.

Most of the time, the bits and pieces that change (the *variant* data), are very small compared to the topic in which they fit--a single path, for example, or a command. That fact makes it somewhat natural to look at those small variants as the "foreground", conrefing them into the larger, "background" context.

But there is a good case to be made for the opposite approach: Encoding the variable information in a topic of its own, and transcluding chunks of background material around it to create context.

> **Note:**
> My thanks to Deborah Pickett to pointing out the value of that approach, and to Sowmya Kannan for recognizing it as a potential solution for a key design problem we were facing at Sun Microsystems (a recognition that was aided by the multiple alternatives she explored).

So rather than having a single topic with contents that look like this:

```
Install.dita
  ...lots of text...
  <ph conref="metadata_platform.dita#install_path"/>
  ...lots more text...
```

We can turn things around 180 degrees and have multiple topics that look like this:

```
Solaris_Install.dita
  <ph conref="base_topic.dita#lots_of_text"/>
  <ph>...Solaris installation path...</ph>
  <ph conref="base_topic.dita#lots_more_text"/>

Linux_Install.dita
  <ph conref="base_topic.dita#lots_of_text"/>
  <ph>...Linux installation path...</ph>
  <ph conref="base_topic.dita#lots_more_text"/>
```

> **Note:**
> The key enabler that makes such an arrangement possible is an authoring tool that shows transclusions in context. As long as your editor does that, it's a pretty viable solution that writers can deal with. Without that behavior, the solution would be pretty darn ugly, and unlikely to get much traction in the real world.

Making it Work

To make such topics work, we need a base topic that has the parts we need to transclude. Then we need to make a topic for each variant we want to produce, transclude all of the parts, and add the variant information.

One downside to this approach is that it's clearly a lot of work. Another is that, like conditionals, the information needed to create a complete document is once again scattered throughout the topic set. That makes it harder, for example, to construct a complete list of everything you need to know to produce documents for a new platform.

But the approach does have a significant advantage--it lets you create links to the topic variants, give them titles of their own, and include them in a DITA map--all without having to do any file swapping.

That advantage could have been the one that decided our final design, at Sun, because it would let us create a DITA map that produces a document like this, where two variants of a single topic are needed in the final document:

```
Applet Developer's Guide
   Developing an Applet
   Deploying an Applet
      Making a JAR File (Solaris)
      Making a JAR File (Windows)
```

That sort of thing is pretty darn hard to do if there is only one topic called "Making a JAR File". It could conceivably be done, but it would take a lot of work. (solution below). But even when it worked, the resulting DITA map could only be used for generating HTML pages. It wouldn't be useful for generating PDFs, DocBook files, help files, or anything else.

Alternative Solution Using a Shared File

a) Generate those pages separately from the main DITA map.
Since they'll have the same name, they'll need to be in separate directories.
Say:

```
solaris/making_jar.html
windows/making_jar.html
```

b) Use "peer"-scoped topicrefs in the DITA map. That causes the links to be generated in the TOC, but doesn't cause the topics to be generated.

a) Since the generated links would be invalid the way they are, add an outputclass attribute in the DITA map to guide downstream processing, and then add a plugin to the DITA OT to modify the links appropriately.

Note:
I'm not taking credit for this solution. I'm only reporting it.
:_)

Conclusion

There is a lot to be said for composition as an architectural strategy. It's worth exploring. Having said that, the key decision is whether to transclude the variable information or the invariants. Whichever way you decide to go, you have a lot of power at your disposal.