Design patterns for information architecture with DITA map domains

Defining a type for collections of topics

Erik Hennum (ehennum@us.ibm.com)
Advisory Software Engineer
IBM

Don Day (dond@us.ibm.com)
Lead DITA Architect
IBM

John Hunt (john_hunt@us.ibm.com)
User Assistance Architect
IBM

Dave Schell
Chief Strategist and Tools Lead
IBM

28 September 2005
(First published 27 September 2004)

The Darwin Information Typing Architecture (DITA) provides maps for assembling topics into deliverables. By specializing the map elements, you can define a formal information architecture for your deliverables. This architecture provides guidance to authors on how to organize topics and lets processes recognize your organizing principles, resulting in a consistent, clear experience for your users.

When a Web site or help system lacks definition and structure, readers can get lost in the content. Information architecture is the practice of organizing and interrelating content so the reader remains oriented and gets answers. By defining formal design patterns for information architecture, content providers can apply tested architectures to improve the user’s experience. Using DITA map domains, you can express these design patterns in XML so authors can reuse them consistently for many collections of content.

This article explains the design technique for creating a DITA map domain. Before delving into the technique, however, you should have a solid understanding of design patterns and how they apply to information architecture.
Design patterns
A design pattern provides a formal definition of a solution and of the problems to which it applies. The goal of design patterns is to avoid approaching each situation as a problem that has never been seen before and, instead, to make it possible to repeat solutions that have worked. In particular, a design pattern distills the experience of an expert or the best practices of a community so everyone can apply that expertise. While the approach originated in architecture and has seen great success in software engineering, design patterns apply equally well to information architecture.

Design patterns are usually expressed as a structured description with formal headings. For instance, a basic template for design patterns might have the pattern name, the contexts in which the problem appears, and the solution to the problem.

Information architecture
Information architecture can be summarized as the design discipline that organizes information and its navigation so an audience can acquire knowledge easily and efficiently. For instance, the information architecture of a Web site often provides a hierarchy of Web pages for drilling down from general to detailed information, different types of Web pages for different purposes such as news and documentation, and so on.

An information architecture is subliminal when it works well. The lack of information architecture is glaring when it works poorly. The user cannot find information or, even worse, cannot recognize or assimilate information when by chance it is encountered. You probably have experience with Web sites that are poorly organized or uneven in their approach, so that conventions learned in one part of the Web site have no application elsewhere. Extracting knowledge from such information resources is exhausting, and users quickly abandon the effort and seek the information elsewhere.

The same issues apply with equal force to other online information systems, such as help systems. The organization and navigation of the information has a dramatic impact on the user's ability to acquire knowledge.

The design pattern for the how-to collection
An example of a design pattern for information architecture might be a how-to collection of topics. A basic design pattern for a how-to collection might be described as follows.

Pattern name
How to

Context
The user needs to accomplish a goal but may not know some of the background or steps. Some users will need to learn the entire activity while others will only need to refresh their knowledge. The goal can be described as a distinct activity.

Solution
Provide a collection of topics that explains the concepts the user must understand, lists the tasks the user must perform, and provides examples of how the goal is accomplished. The tasks are the required core of the how-to collection. The collection sequence is:
• An introductory topic
• Conceptual topics as needed for orientation
• The tasks
• Examples as needed for detail
• A summary topic

A user might have several goals that would benefit from a how-to: for instance, setting up Web authentication, reading a database from a Web application, and so on. By applying the design pattern for the how-to collection, writers provide well-structured and consistent information.

Figures 1-3 illustrate the gain in clarity and consistency for the information architecture by using the design pattern for the how-to collection:

**Figure 1. Casual architecture**

**Figure 2. Design pattern**

**Figure 3. Formal architecture**
In short, the design pattern acts as a kind of blueprint that's fulfilled by the writer. An information architecture might provide a repertoire of such design patterns, each solving different problems.

Even with a design pattern as straightforward as the how-to collection, authors can forget about or misunderstand the pattern. As the number and complexity of design patterns increases, the patterns become increasingly difficult to apply. Such authoring challenges can pose an enormous obstacle to large-scale adoption of design patterns.

**Defining the design pattern in XML**

When defined only through a description, a design pattern has to be promoted and enforced through the perpetual vigilance of editors. A better approach is to represent the design pattern as an XML design that can be validated by an XML editor or parser. Defining the design pattern in XML has the following benefits:

- Markup gives authors guidance while they're working
- Ensures consistent use of the design pattern
- Ensures complete use of the design pattern
- Processing can recognize the design pattern and operate on its declared semantics; for instance, it can provide a special index for how-to collections

Note that defining an XML design for a collection doesn't prevent topic reuse, but instead guides topic reuse so that appropriate types of topics are used at positions within the collection. For example, in the how-to design concept topics appear only as background before the tasks rather than in the middle of the how-to.

**Specializing topics and maps**

DITA supports the definition of a formal information architecture through topic and map types. The topic type defines the information architecture within topics (the micro level) while the map type defines the information architecture across topics (the macro level).

The base topic and map types are general and flexible so they can accommodate a wide variety of readable information. You specialize these general types to define the restricted types required for your information architecture.

- **Topic:** The topic type mandates the structure for the content of a topic. For instance, the DITA distribution includes a task type that provides a list of steps as part of the topic content. This specialized topic type provides guidance to the author and ensures the consistency of all task topics. Processing can rely on this consistency and semantic precision. For instance, the processing for the task type might format the task steps as checkable boxes.

- **Map:** The map type mandates the structure for a collection of topics. A map can define the navigation hierarchy for a help system or the sequence and nesting of topics in a book. For instance, the DITA distribution includes a bookmark demo that mandates a sequence of preface, chapter, and appendix roles for the top-level topics. This specialized map type ensures that the collection of topics conforms to a basic book structure.
Without formal types that represent the information architecture, the information architecture is defined only through editorial guidelines. Different authors may interpret or conform to the guidelines in varying degrees, resulting in inconsistency and unpredictability. By contrast, the formal types can ensure that the design can be repeated for many deliverables.

**Map specialization**

DITA maps have added many capabilities including specialization through map domains. However, instead of packaging specializations of elements for topic content, you specialize elements for map content -- typically the `topicref`. The specialized `topicref` element lets authors specify semantics or constraints on collections of topics. By packaging the `topicref` specializations as a map domain rather than as a map type, you can reuse the design pattern in many different map types.

A specialized `topicref` can be used for the following purposes:

- To restrict references to topics of a specialized type. For instance, a `conceptref` refers only to concept topics (including specialized concepts).
- To assign a role within the collection to a topic. For instance, the topic identified by a `summaryref` element might provide the concluding explanation for a collection.
- To restrict the contents of the collection, requiring specific topic types or roles at specified positions within the collection.

Drawing on all of these capabilities, you can create an XML definition for the design pattern of a how-to collection.

**Implementing a map domain**

A map domain uses the same DTD implementation technique as a topic domain. See "Specializing domains in DITA" for details on the domain implementation technique, which aren't repeated here. Instead, this article summarizes the application of the domain DTD implementation technique to maps:

1. Create a **domain entities file** to declare the elements that extend the `topicref` element.
2. Create a **domain definition module** to define the elements including their element entities, content, and attribute definitions, and the architectural class attribute.
3. Create a **shell DTD** that assembles the base map module and the domain entities file and definition module.
4. From the shell DTD, author a **map** that collects and organizes your topics.

**Declaring the map domain entities**

The entities file for the how-to domain defines the `howto`, `conceptref`, `taskref`, and `exampleref` extensions for the `topicref` element and defines the how-to domain declaration for the domain attributes entity:

```xml
<!ENTITY % howto-d-topicref "howto">
<!ENTITY howto-d-att "(map howto-d)" >
```
Defining the map domain module

The definition module for the how-to domain starts with the element entities so the new elements can, in turn, be extended by subsequent specializations. Of these new elements, only howto has been declared in the entities file because the other new elements should only appear in the child list of the howto element. (In fact, reference typing elements such as conceptref and taskref may also be defined in the entities file for reuse in other specialized child lists.)

Listing 1. Default element entities

```xml
<!ENTITY % howto       "howto">
<!ENTITY % conceptref  "conceptref">
<!ENTITY % taskref     "taskref">
<!ENTITY % exampleref  "exampleref">
<!ENTITY % summaryref  "summaryref">
```

The definition module goes on to define the elements. The definition for the howto element restricts the content list for the collection to the following items:

- The metadata for the overview topic
- References to any number of concept topics
- References to any number of task topics and optional example topics
- A topic acting in the role of a concluding summary

In addition, the howto element refers to the topic that provides an overview of the contents.

Listing 2. Collection structure definition

```xml
<!ELEMENT howto ((%topicmeta;)?, (%conceptref;)*, ((%taskref;), (%exampleref;)?)+, (%summaryref;))>
<!ATTLIST howto
  navtitle   CDATA   #IMPLIED
  id         ID      #IMPLIED
  href       CDATA   #IMPLIED
  keyref     CDATA   #IMPLIED
  query      CDATA   #IMPLIED
  conref     CDATA   #IMPLIED
  copy-to    CDATA   #IMPLIED
  %topicref-atts;
  %select-atts;>
```

The conceptref and taskref elements have a restricted type. When an editor or validating process follows the reference, and the referenced topic doesn't have the declared type (or a specialization from the declared type), the processor reports an error.

Listing 3. Reference type definition

```xml
<!ELEMENT conceptref ((%topicmeta;)?, (%conceptref;)*>
<!ATTLIST conceptref
  href       CDATA   #IMPLIED
  type       CDATA   "concept"
  ...>
<!ELEMENT taskref ((%topicmeta;)?, (%taskref;)>}
<!ATTLIST taskref
  href       CDATA   #IMPLIED
  type       CDATA   "task"
  ...>
```
The `exampleref` and `summaryref` elements don't restrict the type, but instead assign roles to the referenced topics. Because the content list of the `howto` collection allows a topic to act as an example and requires a topic to act as a summary, the author is prompted to create topics in those roles, and the roles can be used in processing -- for instance, to add a lead-in word to the emitted topic titles.

**Listing 4. Reference role definition**

```xml
<!ELEMENT exampleref ((%topicmeta;)?, (%exampleref;)*)>
<!ATTLIST exampleref ...
<!ELEMENT summaryref ((%topicmeta;)?)>
<!ATTLIST summaryref ...
```

On closer investigation, either or both of these particular roles may turn out to reflect a persistent topic structure or semantic, in which case it is appropriate to define topic types and limit the corresponding `topicref` specialization to topics of those types. However, the general technique of assigning a role to a topic in the context of a collection remains valid.

Finally, the definition module sets the class attribute to declare that the new elements derive from `topicref` and are provided by the `howto` package:

**Listing 5. Element ancestry definition**

```xml
<!ATTLIST howto %global-atts;
class CDATA "- map/topicref howto/howto ">
<!ATTLIST conceptref %global-atts;
class CDATA "- map/topicref howto/conceptref ">
```

**Assembling the shell DTD**

As with topic domains, a shell DTD assembles the base map module with the entities file and definition module for the how-to domain:

**Listing 6. Shell document type**

```xml
<!--vocabulary declarations-->  
<!ENTITY % howto-d-dec PUBLIC "-//IBM//ENTITIES DITA How To Map Domain//EN"
"howto.ent"
%howto-d-dec;
...

<!--vocabulary substitution (one for each extended base element, with the names of the domains in which the extension was declared)-->  
<!ENTITY % topicref "topicref | %mapgroup-d-topicref; | %howto-d-topicref;">

<!--vocabulary attributes (must be declared ahead of the default definition) -->
<!ENTITY % included-domains "&mapgroup-d-att; &howto-d-att;">

<!--Embed map to get generic elements -->
<!ENTITY % map-type PUBLIC "-//IBM//Elements DITA Map//EN" "../...dtd/map.mod">
```
Creating a collection with the domain

Using the shell DTD, a map can include one or more how-to collections, as in the following example:

**Listing 7. Map instance document**

```xml
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE map PUBLIC "-//IBM//DTD DITA How To Map//EN" "howtomap.dtd">
<map>
  <!-- how-to clusters can appear anywhere in a map hierarchy but always follow a consistent information pattern within the how to -->
  <howto href="dita-mapdomains.xml">
    <conceptref href="designPatterns.xml"/>
    <conceptref href="informationArchitecture.xml"/>
    <conceptref href="formalCollection.xml"/>
    <conceptref href="xmlDesign.xml"/>
    <conceptref href="mapBackground.xml"/>
    <conceptref href="mapSpecialization.xml"/>
    <taskref href="implementDomain.xml"/>
    <exampleref href="declareEntities.xml"/>
    <exampleref href="domainModule.xml"/>
    <exampleref href="assembleDTD.xml"/>
    <exampleref href="domainInstance.xml"/>
    <summaryref href="summary.xml"/>
  </howto>
</map>
```

In fact, this example is the map for the article that you’re reading right now -- that is (as you may well have noticed), this article conforms to the design pattern for a how-to collection. Here’s the list of topics in this how-to article along with the topic type or role and title:

**howto: Design patterns for information architecture with DITA map domains**

- concept: Design patterns
- concept: Information architecture
- concept: The design pattern for the how-to collection
- concept: Defining the design pattern in XML
- concept: Specializing topics and maps
- concept: Map specialization
- task: Implementing a map domain
- example: Declaring the map domain entities
- example: Defining the map domain module
- example: Assembling the shell DTD
- example: Creating a collection with the domain (this topic)
• summary: Summary

For this article, the how-to collection constitutes the entire content set, however that arrangement isn't required. A how-to collection might be part of a larger content set. For instance, in a help system, you might include multiple how-tos as part of a navigation hierarchy. Similarly, you might use how-to collections in books by creating a new shell DTD that combines the bookmark map type with the how-to map domain.

As you explore design patterns and collections, you'll find that, in addition to topics, a design pattern can aggregate smaller collections. For instance, you can create design patterns for a how-to collection, a case study collection, and a reference set collection. A product information design pattern might then require a product summary topic and at least one of each of these subordinate collections in that order. You can implement the design in XML by creating map domains for each of the low-level design patterns and, in addition, the high-level design pattern.

You'll also find that, to represent a high-level relationship with a collection, you can create a relationship to the root topic for the collection. As the introduction and entry point for the collection, the root topic provides the most general statement for the content of the collection. That is, you can treat the set of topics as a collective content object, using the root topic to represent the collection as a whole for navigation and cross references.

Summary

In this article, you've learned how to specialize the topicref element to mandate a specific collection of topics. For complete, single-purpose collections such as functional specifications, information plans, and quick reference guides, you might package these specialized topicref elements with a new map type. For design patterns such as how-tos or case studies that can appear within a larger content set -- especially when different designers might create different collection types -- you might want to package the specialized topicref elements as a map domain.

By specializing a DITA map in this way, you can implement a formal information architecture that applies design patterns through the XML definition. In particular, information architects can block out the plan for a content set using these large-scale design patterns as collective objects. By taking this approach, information architects can provide guidance to authors and declare collective semantics for processors with the end result that users receive consistent, complete, and well-organized information.
## Downloads

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x-dita7-howto.zip</td>
<td>15KB</td>
</tr>
</tbody>
</table>
Resources

- IBM donated DITA to the OASIS standards organization in March of 2004, where it is now managed by the OASIS DITA Technical Committee (http://www.oasis-open.org/committees/dita/). In April of 2005, OASIS approved Version 1.0 of the DITA specification, which consists of the following documents:
  - A consolidated .zip file with all specifications, DTDs, and Schemas is publicly available in the documents section of the OASIS DITA Technical Committee site: http://www.oasis-open.org/committees/download.php/12091/cd2.zip
- A reference implementation toolkit for both the developerWorks and OASIS 1.0 versions of the DITA DTDs/Schemas is available at the DITA Open Toolkit project site on SourceForge: http://dita-ot.sourceforge.net. The DITA Open Toolkit supercedes all previous versions published on developerWorks, the last version of which was commonly called "dita132".
- Download the definition for the how-to map domain and the DITA source files for this article as a sample for the use of the domain. Unzip the example in the root directory for the dita13 distribution of the public DITA toolkit on developerWorks.
- Learn more about DITA from the OASIS Cover page.
- Read the developerWorks article "Introduction to the Darwin Information Typing Architecture" (updated September 2005).
- Learn how to implement DITA specialization with "Specializing topic types in DITA" by Michael Priestley (developerWorks, updated September 2005).
- Read Erik Hennum’s article "Specializing domains in DITA," which shows you how to leverage the extensible DITA DTD to describe new domains of information (developerWorks, updated September 2005).
- Find out how to join the discussion in the DITA forum, moderated by Don Day and Michael Priestley.
- Discuss DITA on the DITA user’s group.
- Look at a specific catalog of design patterns, primarily for user interfaces but with some applicability to information structure.
- Follow the trends in information architecture.
- Get a more complete background on design patterns for many disciplines.
About the authors

Erik Hennum

Erik Hennum works on the design and implementation of User Assistance for the IBM Storage Systems Group. For DITA, he has helped shape the principles of domain specialization.

Don Day

Don Day designs and supports publishing tools for IBM's Information Development community and has represented IBM on the W3C XSL and CSS Working Groups. For the past three years, Don has led the workgroups that developed and now maintain the DITA DTDs and specification. He has B.A.s in English and Journalism and an M.A. in Technical and Professional Communication from New Mexico State University.

John Hunt

John Hunt charts the overall technical direction and strategy for product user assistance for IBM Lotus Software. He has designed award-winning help systems and spearheaded the move to a topic-based, modular, and layered information architecture. He's currently planning the transition to XML and DITA at Lotus, and how best to leverage the two for providing user assistance for Lotus Workplace.

Dave Schell

Dave A. Schell is IBM's chief strategist and tools lead in support of its technical writing (User Technology) community.

© Copyright IBM Corporation 2004, 2005
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)