An OASIS White Paper

DITA 1.2 Feature Overview: Domain and topic integration

Marc Speyer

12 November 2009

OASIS (Organization for the Advancement of Structured Information Standards) is a not-for-profit, international consortium that drives the development, convergence, and adoption of e-business standards. Members themselves set the OASIS technical agenda, using a lightweight, open process expressly designed to promote industry consensus and unite disparate efforts. The consortium produces open standards for Web services, security, e-business, and standardization efforts in the public sector and for application-specific markets. OASIS was founded in 1993. More information can be found on the OASIS website at http://www.oasis-open.org.

The OASIS DITA Adoption Technical Committee members collaborate to provide expertise and resources to educate the marketplace on the value of the DITA OASIS standard. By raising awareness of the benefits offered by DITA, the DITA Adoption Technical Committee expects the demand for, and availability of, DITA conforming products and services to increase, resulting in a greater choice of tools and platforms and an expanded DITA community of users, suppliers, and consultants.

DISCLAIMER: All examples presented in this article were produced using one or more tools chosen at the author's discretion and in no way reflect endorsement of the tools by the OASIS DITA Adoption Technical Committee.

Table of contents

Domain and topic integration

In DITA 1.2 the integration of domains and topics has been unified to improve design flexibility and to simplify the DITA specialization constructs. By relaxing the restriction from what modules may inherit from, vocabulary elements can now be shared between different kinds of modules.

Overview

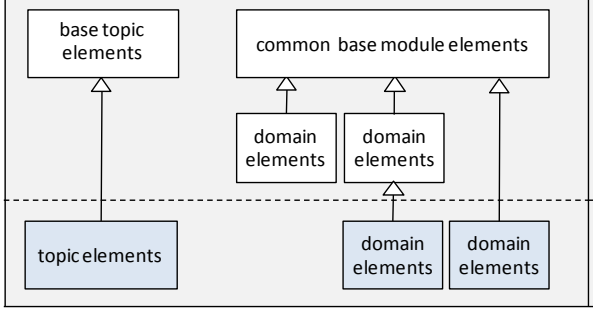
Specialization is an important feature of the Darwin Information Typing Architecture (DITA) that has many benefits such as the ability to quickly define more appropriate information types, better interoperability, increased consistency and reduced learning time for users. Specialization can be achieved at two levels:

- Structural specialization defines new types of structured information, such as new topic types or new map types.
- Domain specialization creates new markup for use in structural types, such as new kinds of keywords, tables, or lists, or new attributes such as conditional processing attributes

Until DITA 1.2 only single inheritance was allowed: topic elements could be specialized off elements in their parent topic and domain elements could be specialized off elements in their parent domain or elements in the common base module. Put in another way[1]:

When you define new types of topics or domain elements, remember that the hierarchies for topic specialization and domain specialization must be distinct. A specialized topic cannot use a domain element in a content model. Similarly, a domain element can specialize only from an element in the base topic or in another domain. That is, a topic and domain cannot have dependencies. To combine topics and domains, use a shell DTD.

The following table summarizes the inheritance architecture in DITA 1.1.

Inheritance diagram	Level	Scope
	base topic and topic domains specialized topics and domains	global local to topic

Legend

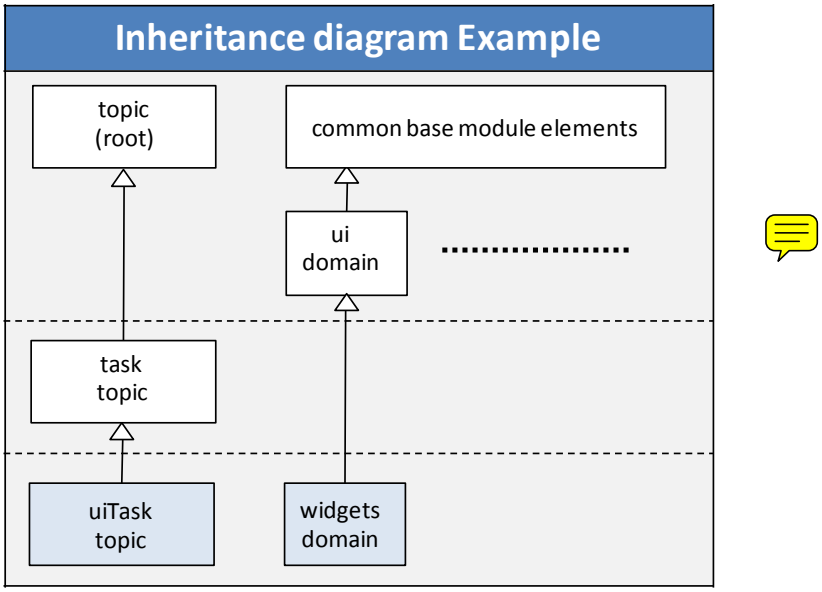
Module DTD or schema

Inheritance (Generalization) relationship through class attribute of elements in module

[1] See Specializing domains in DITA, Eric Hennum, September 2005, IBM developerWorks

Example 1: Specialized topic and domain in DITA 1.1

Suppose we have identified a need for a new task model that will instruct users how to perform a task using a special specific user interface. Let's call this task: <uiTask>. The user interface has a special type of widget and it is important to proper distinguish it from other user interface elements. After examining what is available in the User Interface domain of the DITA standard we decide that the <uicontrol> is the most appropriate element to use as a basis for our domain specialization. The situation is given in the following figure.



The process of creating a new document type in DITA is called integration. ~~Briefly when~~ In the case of a DTD, the steps involved are:

For inheritance (class attribute)

1. Set the class attribute of the <uiTask> element in the uiTask to "- topic/topic task/task uiTask/uiTask"

2. Set the class attribute of <widget> element in the widgets domain module to "+ topic/ph ui-d/uicontrol widgets-d/widget"

For domain inclusion

1. Set the domains attribute of the <uiTask> element in the uiTask module to "&included-domains;"

2. Define the element extension entities in the widgets domain entity file
   <!ENTITY widgets-d-uicontrol "widget">
   <!ENTITY widgets-d-ph "widget">

3. Define the domain identification entity in the widgets domain entity file
   <!ENTITY widgets-d-att "(topic ui-d widgets-d)">

4. Add the domain extensions elements
   <!ENTITY % uicontrol "uicontrol | widget-d-uicontrol;”>

5. Make the following declaration in the uiTask document type shell
   <!ENTITY included-domains "&widgets-d-att">

Relaxation of the inheritance restriction in DITA 1.2

DITA 1.2 relaxation of the inheritance restrictions addresses sharing limitations between vocabulary elements. What limitations are removed and how it works is best ~~be~~ explained by the examples later in this section, but first we need to introduce some terminology.

.

Extension element
An extension element is an element that - under the control of a document type shell - can appear as an alternative or replacement for its base element in contexts where its base element can appear.

A specialization of <topic> is always an extension element. For instance, an information architect can allow the <reference> to appear instead of <topic> in all <topic> contexts including the top context in a document as well as nested contexts in the content models for the <dita> and <topic> elements.

Domain modules always provide one or more extension elements. For instance, the programming domain supplies the <apiname>, <codeblock>, <codeph>, <option>, <parml>, <parmname>, <synph>, and <syntaxdiagram> extensions of the base <dl>, <fig>, <keyword>, <ph>, and <pre> elements.

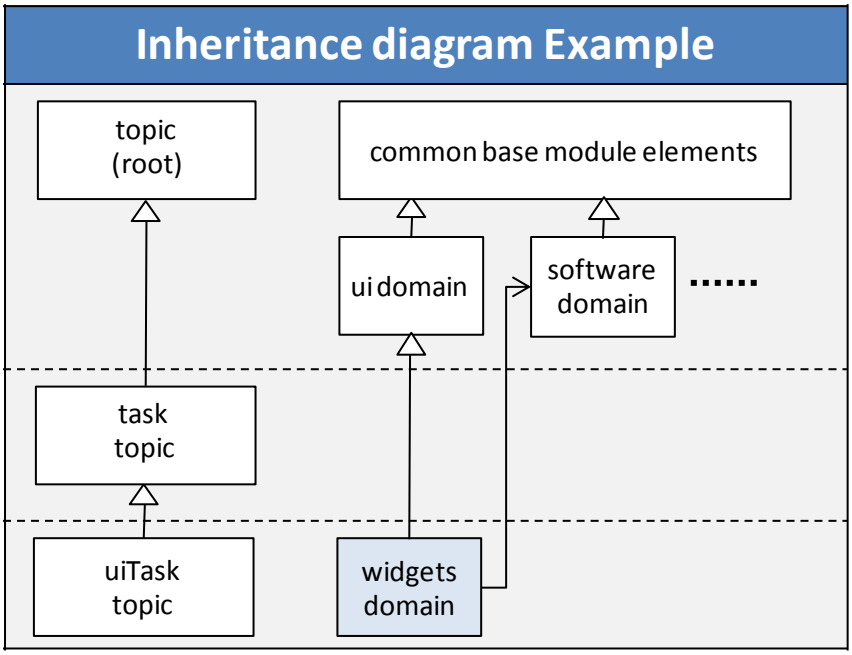
Substructure element
A substructure element can only appear within an extension element.

For instance, the <properties> element from the reference vocabulary module can appear only in the <refbody> under the <reference> extension element. Similarly, the <plentry> element from the programming domain can appear only as a sub-element of the <parml> extension element.

Example 2: A specialized domain extension whose substructure includes a preexisting domain element

Suppose that in example 1 of the previous section actions invoked by widgets can also be invoked by commands. After examining the standard DITA domains we find the <cmdname> element of the Software domain to be the most appropriate. Previous to DITA 1.2 we could only specialize from a single domain inheritance path (see the domain identification entity in example 1) and therefore we must define a new element in the widgets domain with the same content model as the <cmdname> element and call it something like <widgetCmdname> (element names must be unique).

In DITA 1.2 the substructure of a specialized element in a domain may use another domain. This is illustrated in the following figure.



Legend

Module DTD or schema

Inheritance relationship

Usage of domain, inclusion in domain identification entity

The architectural attribute declarations (omitting class attributes of <uiTask>) are:

uiTask/@domains:	(topic	ui-d+sw-d	widgets-d)
widget/@class:	"+ topic/ph	ui-d/uicontrol	widgets-d/widget "
cmdname/@class:	"+ topic/keyword	sw-d/cmdname "	

Instances of <uiTask> generalize to any of the following combinations of modules:

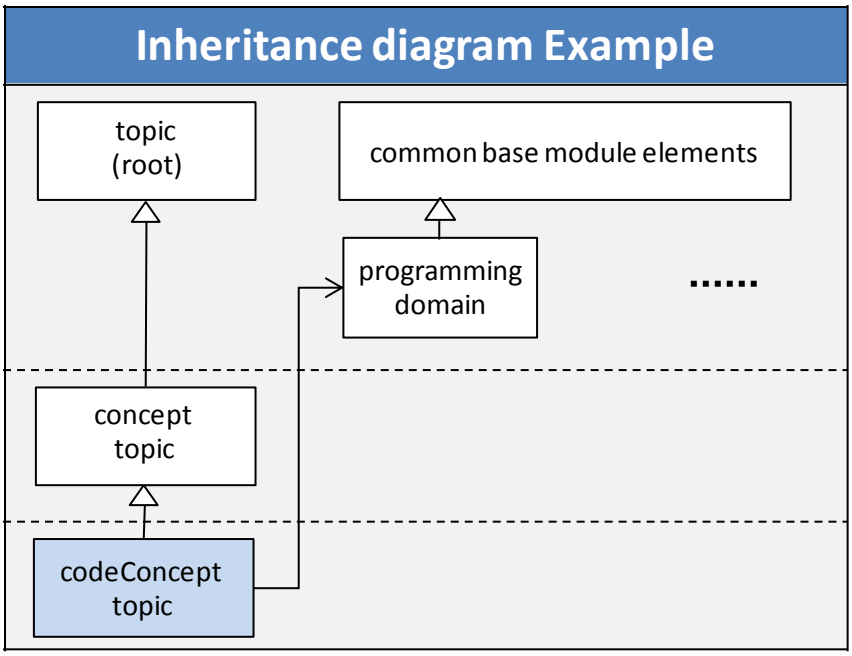
- topic and UI and software
- topic and software
- topic and UI
- topic

Notes:

- Parenthetical expressions are used for both the domains and class attributes which is a recommendation in DITA.1.2.
- The domains attribute could imply that some elements from the User Interface and Software domain will not be part of the uiTask, but these elements will be declared any way since the domain must be referenced the shell document type.

Example 2: A specialized topic whose substructure includes a preexisting domain element

Suppose that we want to explain a programming technique and have identified that the concept topic satisfies our need for topic specialization and that <codeblock> from the programming fits the requirements for code listings. Let's call the topic <codeConcept>. The body of this topic <codeConceptBody> lists a <codeblock> from the programming domain which is possible in DITA 1.2. This is illustrated in the following figure.



Legend

- Module DTD or schema
- Inheritance relationship
- Usage of domain, inclusion in domain identification entity

The architectural attribute declarations:

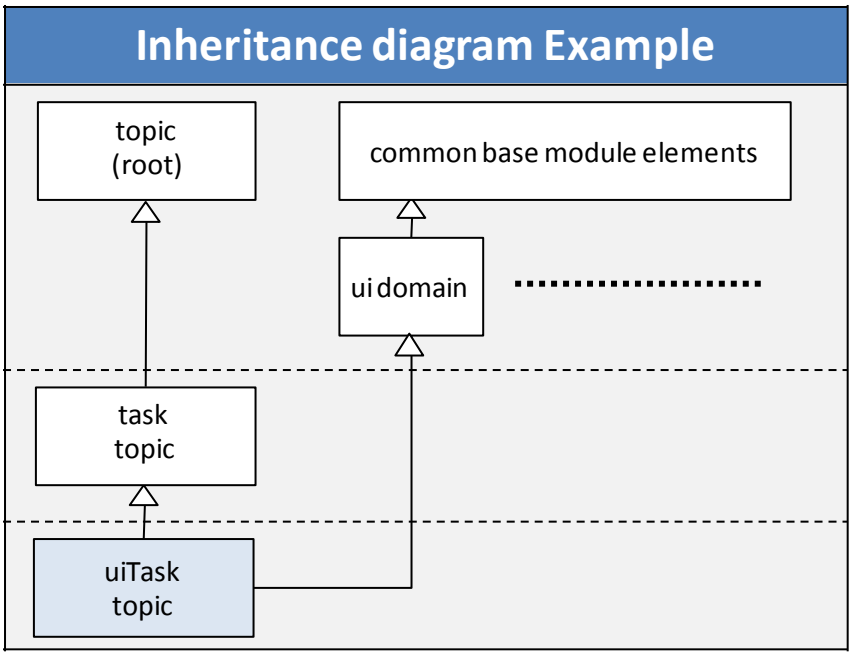
codeConcept/@domains:	(topic	concept+pr-d	codeConcept)
codeConcept/@class:	"- topic/topic	concept/concept	codeConcept/codeConcept "
codeConBody/@class:	"- topic/body	concept/conbody	codeConcept/codeConBody "
codeblock/@class:	"+ topic/pre	pr-d/codeblock "	

Instances of <codeConcept> generalize to any of the following combinations of modules:

- topic and concept and programming
- topic and concept
- topic and programming
- topic

Example 3: A specialized topic whose substructure specializes a preexisting domain element

Suppose that the <uiTask> topic specializes the task topic and has <uiTaskBody> and <uiContext> substructure. In the <uiContext> we want to identify the menu item for the task. This can be achieved by a <uiMenuContext> element as a specialization of <menucascade> of the User Interface domain (which is not allowed in DITA 1.1 because elements in a structural specialization cannot be specialized from domain elements). This is illustrated in the following figure.



The architectural attribute declarations are:

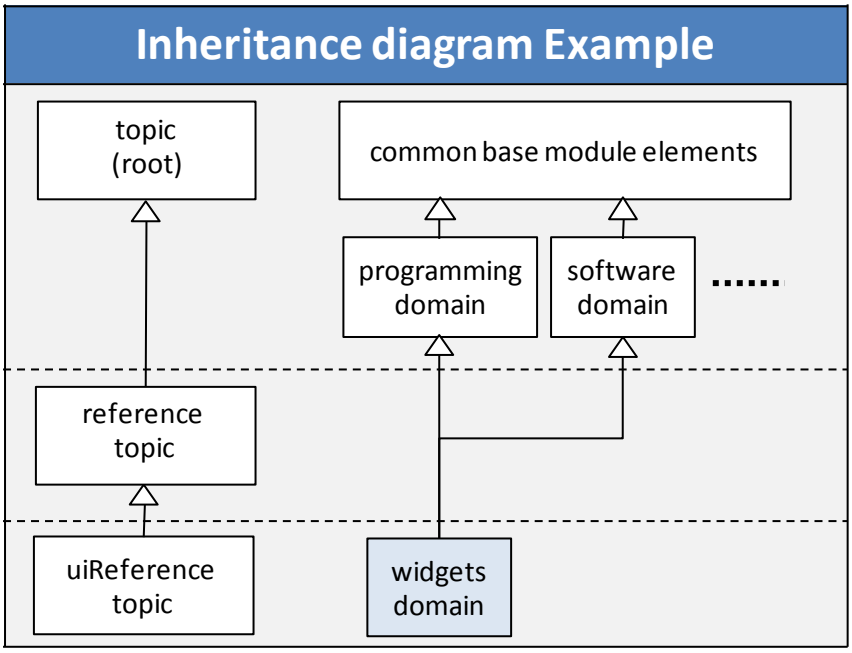
uiTask/@domains:	(topic	task+ui-d	uiTask)
uiTask/@class:	"- topic/topic	task/task	uiTask/uiTask "
uiTaskBody/@class:	"- topic/body	task/taskbody	uiTask/uiTaskBody "
uiContext/@class:	"- topic/section	task/context	uiTask/uiContext "
uiMenuContext/@class:	"+ topic/ph	ui-d/menucascade	uiTask/uiMenuContext "

Instances of <uiTask> generalize to any of the following combinations of modules:

- topic and task and UI
- topic and task
- topic and UI
- topic

Example 4: A domain with extension and substructure elements that specialize different domains

Suppose that we want to create a library of UI controls and their associated class names using <uiReference> topics. We can use <widget> element which specializes <uicontrol> from the UI domain and has the <widgetName> specialization of the <apiname> extension element from the programming domain that identifies the control. This would require a specialization from two different domains as illustrated in the following figure.



The architectural attribute declarations:

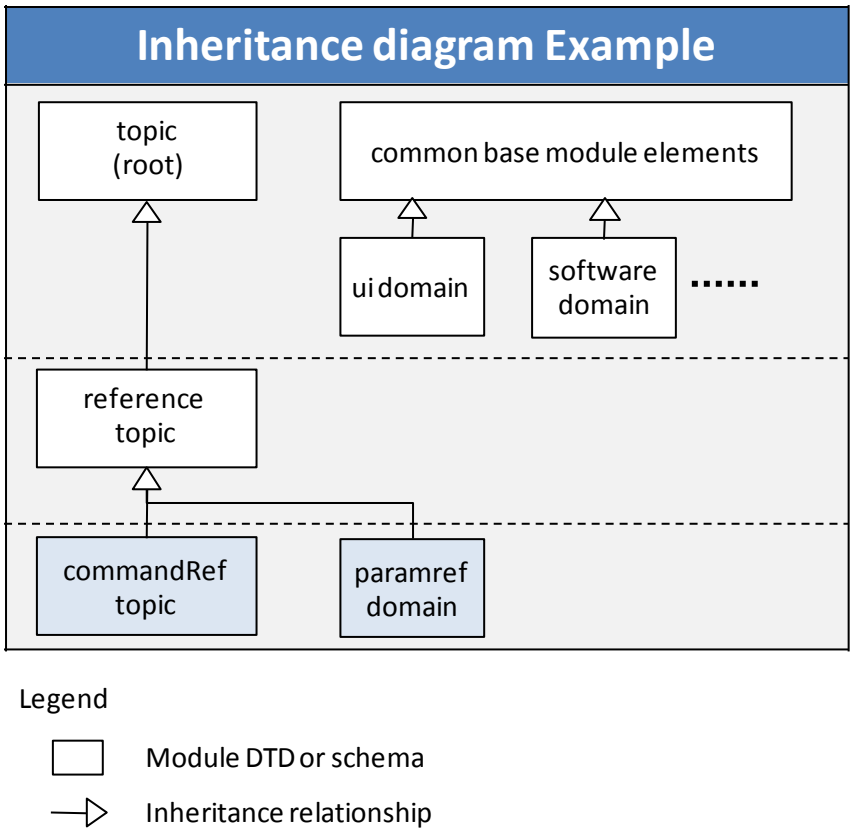
uiReference/@domains:	(topic	ui-d+pr-d	widgets-d)
widget/@class:	"+ topic/ph	ui-d/uicontrol	widgets-d/widget "
widgetName/@class:	"+ topic/keyword	pr-d/apiname	widgets-d/widgetName "

Instances of <uiReference> generalize to any of the following combinations of modules:

- topic and UI and programming
- topic and UI
- topic and programming
- topic

Example 5: A specialized topic whose substructure requires a domain that extends the substructure of the base topic

Suppose that we want to markup the parameters as part of the reference for a command, function, or statement using <commandRef> topics. We find that the <properties> is the most suitable and introduce a <parameters> extension element that specializes <properties> from the reference topic and contains the <paramtype> and <paramdesc> specializations of <proptype> and <propdesc>.This means that we have to create a domain that specializes a topic which is possible in DITA 1.2 This is illustrated in the following figure.



The architectural attribute declarations:

commandref/@domains:	(topic	reference	paramref-d)
	(topic	reference+paramref-d	commandref)
commandref/@class:	"- topic/topic	reference/reference	commandref/commandref "
commandBody/@class:	"- topic/body	reference/refbody	commandref/commandBody "
parameters/@class:	"+ topic/simpletable	reference/properties	paramref-d/parameters "
paramtype/@class:	"+ topic/stentry	reference/propvalue	paramref-d/paramtype "
paramdesc/@class:	"+ topic/stentry	reference/propdesc	paramref-d/paramdesc "

Instances of parReference generalize to any of the following combinations of modules:

- topic and reference and paramref
- topic and reference
- topic

Summary

DITA 1.2 allows much more flexibility for combining vocabularies. This is achieved through relaxation of the inheritance restrictions from previous versions. This new architecture will let designers create new document types from domains and specialized topic types that extend or reuse elements from single or multiple domains and topics. The methods for determining compatibility for generalization and conref are described in the DITA 1.2 Architectural Specification.