An OASIS DITA Adoption TC White Paper

DITA 1.2 Feature Description: Domain and topic integration

Tony Self
On behalf of the OASIS DITA Adoption Technical Committee

Date: 11 December 2009

OASIS

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. The restrictions on module inheritance have been relaxed enabling vocabulary elements to be shared between different types of modules.
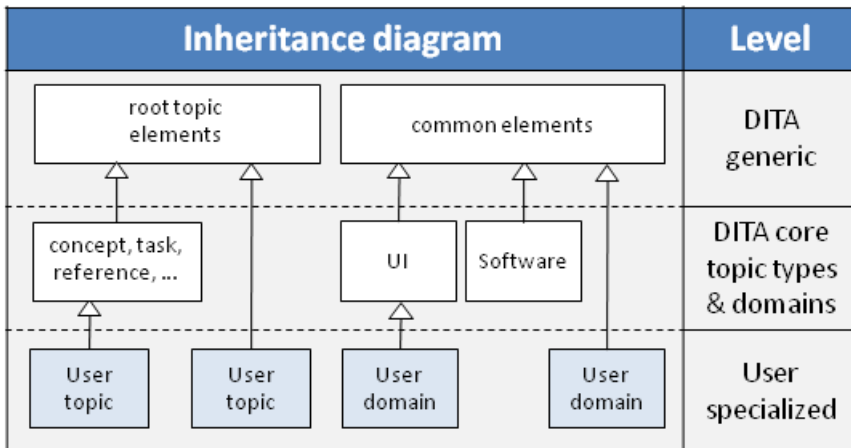
Specialization in DITA 1.1

Specialization is an important feature of the Darwin Information Typing Architecture (DITA). It 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. It is intended for creating new document types for specific industry sectors (machine industry, pharmaceuticals, etc.) or a specific purpose (learning and training).
- Domain specialization defines new markup for use in many structural types, such as new kinds of keywords, tables, or lists, or new attributes such as conditional processing attributes. It is intended for creating vocabularies for subject areas such as user interfaces or programming language constructs.

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:

> 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. [1]

The following table summarizes the inheritance architecture in DITA 1.1. Note that topics are also created from common elements but this "use" relationship is not shown in the diagram.

Legend

☐ Module DTD or schema

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

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 specific user interface. Let's call this task: <uiTask>. In addition the user interface has a special type of widget and it is important

---

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

to properly distinguish it from other user interface elements (see Figure 1: Example of a uiTask instance (see page 5).

```
<uiTask id="t1">
<title>Using page grids</title>
<uiTaskbody>
<context>Turn the page grid on or off </context>
<steps>
<step>
<cmd>On the <widget><image href="actionbar.png" /></widget> click <uicontrol>Grid</uicontrol></cmd>
<stepresult>The page grid is shown. Remove the grid by selecting this command again.</stepresult>
</step>
</steps>
</uiTaskbody>
</uiTask>
```

Figure 1: Example of a uiTask instance

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 widgets domain specialization. The situation is given in the following figure.

The process of creating a new document type in DITA is called integration. The complete steps for integration are beyond the scope of this document and are not listed here. Instead only the definitions for setting up the inheritance hierarchy are given in Setting up inheritance in DITA 1.1 (see page 5).
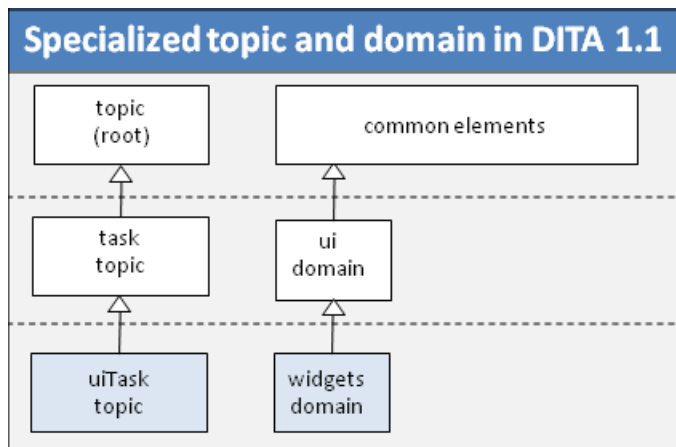
Setting up inheritance in DITA 1.1

The procedure below contains the settings that must be made for creating a domain specialization in DITA 1.1. It is not meant as detailed procedure for creating a domain specialization.

Creating a domain specialization in DITA 1.1

> **Note:** The uiTask topic specialization which is assumed the following steps has been included to give the domain specialization a context. All of the steps below also apply if the <widget> element is added to one of the base topic types.

- Set the class attribute of the <widget> element in the widgets domain module file (`widget.mod`) to identify the inheritance from the UI domain
  The widget domain module file should have something like the following attribute declaration (other attributes have been omitted):
  ```
  <!ATTLIST widget class CDATA "+ topic/ph ui-d/uicontrol widgets-d/widget ">
  ```
- Define the element extension entities in the widget domain entity file (`widget.ent`)

The element extension entities are used when integrating the different domains in the shell DTD

The widgets domain entity file should have the following entity declarations:

```
<!ENTITY % widgets-d-uicontrol "widget">
<!ENTITY % widgets-d-ph "widget">
```

- Define the domain identification entity in the widget domain entity file (`widget.ent`)

  The domain identification entity will be used to set the value of the `domains` attribute of the <uiTask> element to the domain inheritance hierarchy.

  The widgets domain entity file should have the following entity declaration:

```
<!ENTITY widgets-d-att "(topic ui-d widgets-d)">
```

- Set the `domains` attribute of the <uiTask> element in the uiTask module file to the included domains
  The uiTask module file should have something like the following attribute declaration(other attributes have been omitted):

```
<!ATTLIST uiTask domains CDATA "&included-domains;">
```

- Add the previously defined domain extension elements to the base element extension entity in the uiTask shell DTD file

  Extension entities are needed both for <ph> as well as for <uicontrol>. This is to make the <widget> available within topic structures everywhere <ph> is allowed and not just in topic structures where <uicontrol> is explicitly allowed (e.g. <menucascade>).

```
<!ENTITY % ph "ph | %pr-d-ph; | %sw-d-ph; |  %ui-d-ph; | %widgets-d-ph;">
<!ENTITY % uicontrol "uicontrol | %widgets-d-uicontrol; ">
```

- In the uiTask shell DTD file define which domains are included (other domains have been omitted)

  Since the widgets domain extends the UI domain both must be included.

```
<!ENTITY included-domains "&ui-d-att; &widgets-d-att">
```

Relaxation of the inheritance restriction in DITA 1.2

Explanation of inheritance relaxation in DITA 1.2 with examples

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.

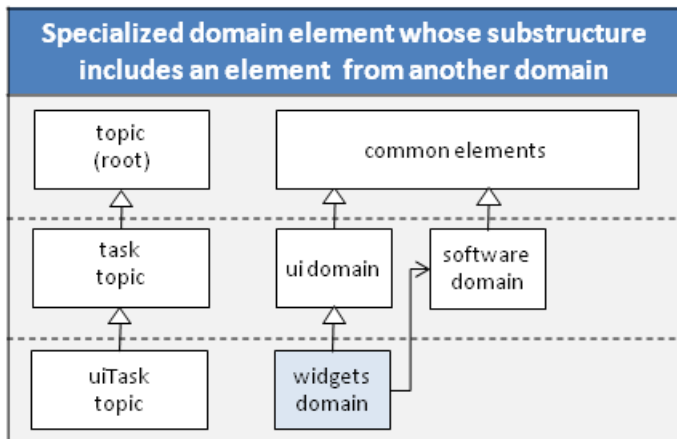## A specialized domain extension whose substructure includes a preexisting domain element

Suppose that in the *DITA 1.1 specialization example* actions invoked through 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 had to define a new element in the widgets domain with the same content model as the <cmdname> element and call it something like <widgetCmdname> (note that element names must be unique).

uiTask topic using multiple domains

Here is an example uiTask instance:

```
<uiTask id="t2">
<title>Using page grids</title>
<uiTaskbody>
<context>Turn the page grid on or off </context>
<steps>
<step>
<cmd>On the <widget><image href="actionbar.png" /></widget> click <uicontrol>Grid</uicontrol> or
enter the command: <widget><cmdname>grid on</cmdname></widget> in the command window</cmd>
…
</step>
</steps>
</uiTaskbody>
</uiTask>
```

DITA 1.2 allows the substructure of a specialized element in a domain to use another domain as illustrated in the following figure.

The architectural attribute declarations (omitting class attributes of <uiTask>) for setting up the inheritance hierarchy 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*
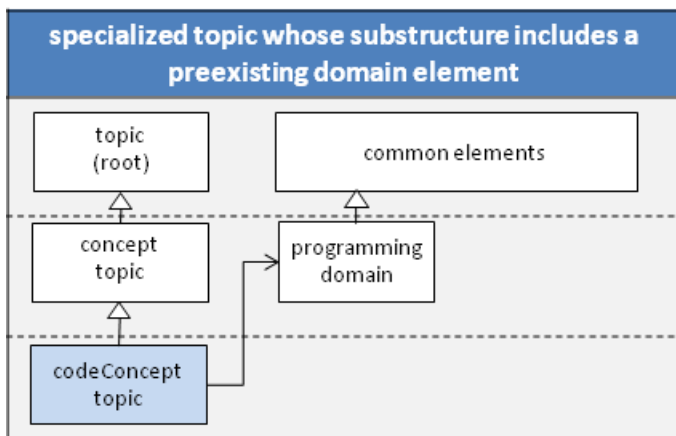- *topic and software*
- *topic and UI*
- *topic*

Note:

- *In DITA 1.1 a document type can be created that would allow similar uiTask instances but there is no support for an architectural attribute declaration that specifies a dependency on a combination of modules.*
- *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 in the shell document type.*
- *The explicit "use" relationship is shown to make the inclusion of another domain clear. Topics also make use the common elements but this relationship is not shown in the diagram.*

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, i.e. want to directly reference an element from a preexisting domain in our topic structural specialization.

Specialized codeConcept topic with its body element referencing a domain element

Let's call the topic <codeConcept>. The body of this topic <codeConBody> lists a <codeblock> from the programming domain which is possible in DITA 1.2 and illustrated in the following figure.

specialized topic whose substructure includes a preexisting domain element

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*

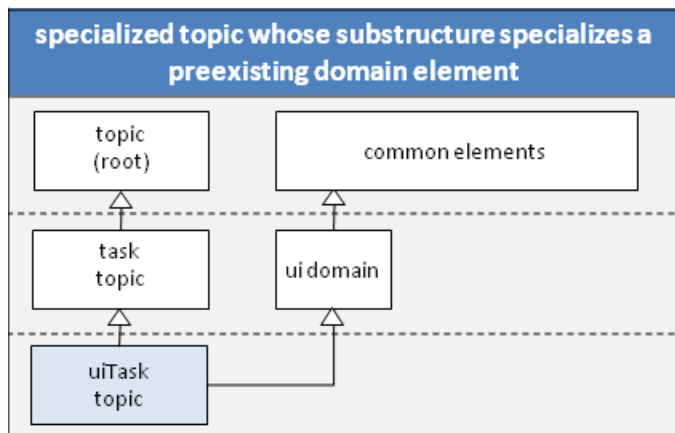# A specialized topic whose substructure specializes a preexisting domain element

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 by means of a <uiMenuContext> element which has a similar content model as <menucascade>.

uiTask topic with an element specialized from a preexisting domain element

Here is an example uiTask instance:

```
<uiTask id="t4">
<title>Using Ruler Guides</title>
<uiTaskbody>
<uiContext>To perform the steps in this task first select from the main menu
<uiMenuContext>
<uicontrol>View</uicontrol>
<uicontrol>Show</uicontrol>
<uicontrol>Guides</uicontrol>
</uiMenuContext>
</uiContext>
</uiTaskbody>
</uiTask>
```

In DITA 1.2 we can specialize <uiMenuContext> from <menucascade> of the User Interface domain (which is not allowed in DITA 1.1 because elements in a structural specialization cannot specialize 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:
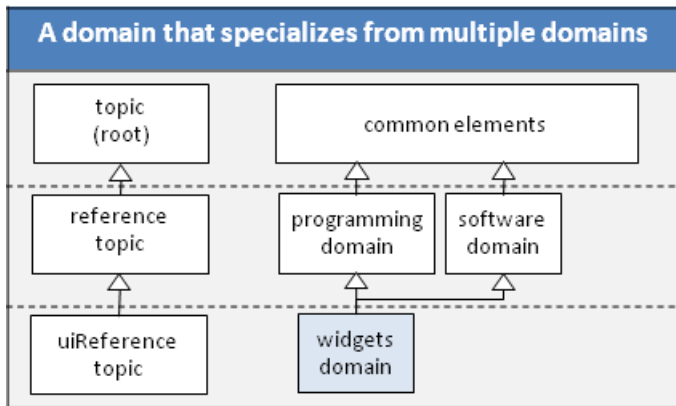
- *topic and task and UI*

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

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 the <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.

Widget domain specializing from programming and software domain

The following figure shows specialization from two different domains:[2]

A domain that specializes from multiple domains

Legend

☐ Module DTD or schema

─▷ Inheritance relationship

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*
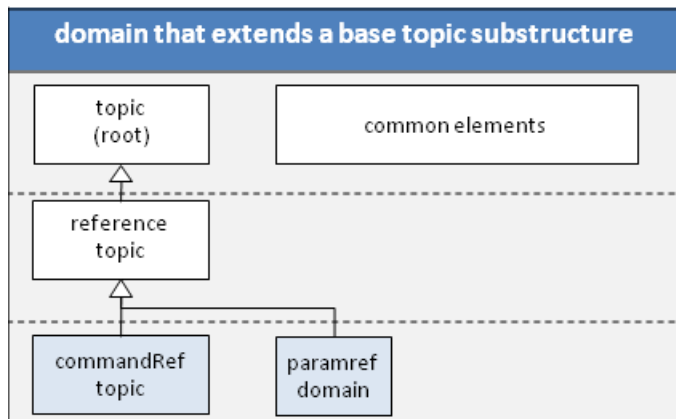- *topic and programming*
- *topic*

---

[2]

Note: *uiReference topic in this example has been included to show the attribute declarations for the* `domains` *attribute.*

A specialized topic whose substructure requires a domain that extends the substructure of a 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>.

A domain that extends the subtstructure of a base topic

We have to create a domain that specializes a topic which is possible in DITA 1.2 and 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.