Clause Model Solution Proposal

The underlying pattern

Submitted by Peter Meyer (pmeyer@elkera.com.au)
Wednesday, July 09, 2003.

Introduction

Purpose of this document

This document provides an outline of Elkera's proposal for a clause model that will satisfy the requirements set out in the Requirements for clause model dated 27/05/03.

The model is a simplified extract from the Elkera® Topic DTD which is used by Elkera for the markup of a wide range of documents for content management purposes.

The model is offered as a contribution to the analysis and discussion of possible clause models to satisfy the requirements.

The key points about the model are:

(a) It provides for the markup of the simplest paragraph to very complex clause like structures.

(b) It avoids the need for authors to make semantic distinctions between concepts such as clause, subclause and list.

Intellectual property rights in the Elkera Topic DTD

Elkera is prepared to relinquish copyright in the Elkera® Topic DTD if it or parts of it are adopted as the clause model for the TC.

Content models

The simplified content models are as follows:

```
<Element topic (num?, title, (topic+ | item + | block+))>
<Element item (num?, title?, (item + | block+))>
<Element block (text | item)+>
<Element text (#PCDATA)>
<Element num (#PCDATA)>
<Element title (#PCDATA)>
```

Element names

The proposed clause model is built around `topic`, `item`, `block` and `text` elements. While initially unfamiliar, these document neutral terms are intended to provide a language to allow authors to think about documents in simple, hierarchical terms.

A brief overview of these terms is as follows:

(a) A `topic` is intended to describe a discrete, free standing unit of content that might be re-used from document to document. If each major container is considered as a discrete information topic, this facilitates authoring for re-usability of content. The topic can contain other topics (it is recursive) to form the main document hierarchy.

(b) An `item` can be used as an alternative structure to a topic in the main hierarchy and also as a list item or paragraph. It can function in the role of a clause, subclause or list item.

(c) A `block` is the nearest equivalent to a grammatical paragraph. It may exist as a free standing object, such as a paragraph in a letter or as the main content holder anywhere within the document hierarchy.

(d) The `text` element is the PCDATA element for paragraph content. Generally, it can exist only inside a `block`.

These terms are further described below. They may be considered to be illustrative only of the proposal for neutral element names set out in the requirements.

topic

The `topic` element is a recursive element that may be used to create the main document hierarchy. At any point a topic may be a grouping container such as a chapter or it may contain block content and function as a clause. Contract and other business documents rarely have a fixed depth hierarchy so it is possible that block content can appear at any level in the hierarchy, as shown in the attachment to the Requirements.

Topics must have a title so that, if desired, topics can appear in a contents listing. Topics may be numbered.

item

Overview

The item element is used in two broad situations:

- for any numbered or unnumbered object in the main document hierarchy when topic is not desired (not to appear in the contents listing) or is not available (its parent is an item);
- for all lists, whether numbered, bulleted or unnumbered.

Item elements in the main document hierarchy

The `item` element can be used in two situations in the main document hierarchy:

- Inside a `topic` it is often appropriate to add further headings or titles that are not intended to appear in a contents listing. In that case, the `item` element should be used with a title. It should be noted that there is nothing to prevent an application using `item` titles in the contents listing. It is simply a design philosophy that decisions for authors and application developers will be simpler if they each understand that items are excluded.

This overcomes the common problem of building contents listings for structures that may or may not have a title or heading.

- Often it is desired to create numbered objects in the hierarchy without titles. Some might think of these as subclauses, others as lists. The `item` element satisfies this need.

The proposed model and the standard Elkera Topic DTD does not allow `block` and `item` elements to be mixed at the same level inside a `topic` or another `item`. It is always necessary to create a regular hierarchy. However, if desired, it would be possible to allow them to be mixed by changing the item content model as follows:

```
<Element item (num?, title?, (item | block)+)>
```

This would allow an irregular hexarchy which may not be so problematic when the headings are excluded from the document contents listing. It would be possible to allow this within `topic` also if desired.

The availability of two elements (`topic` and `item`) that can perform essentially the same function in the main document hierarchy to act as grouping containers or clauses might be controversial. The reasons for this are:

(a) It is a difficult exercise for application developers and authors alike to have a common understanding about what will appear in a contents listing. If the heading element on the clause level object is optional, contents generation can be unsatisfactory. There is also an issue of the depth in the hierarchy at which content generation is to stop. Does it go to 3, 4 or more levels? Sometimes the author can signify this in some other way when printing the document but this is extremely difficult in the situation shown in the Attachment to the Requirements. Some objects at level 3 in the hierarchy have titles and others do not.

(b) Content objects appear in a contents listing because of their relative importance as discrete information objects. This is consistent with the concept of a topic discussed earlier.

(c) Frequently, content is created with a title or heading in circumstances where it should not appear in a contents listing and the individual objects are indivisible components of a larger information object (topic).

(d) The `topic` element has a required heading. This disqualifies it from being used as a main object in the hierarchy that may be numbered but not given a title (often called subclauses). A subordinate element of some kind is required to meet this need.

(e) In the author's experience, it is simpler for application development to provide a translator utility in the drafting environment to convert between `topic` and `item` elements than to deal with contents generation where some sequences have headings, others do not and some are mixed. Any distinction between clause, subclause or list item objects creates a need to translate between these object types from time to time.

The consequence of this design is that there is a trade-off between absolute certainty and convenience in contents generation against the occasional need to translate a `topic` to an `item` or vice versa.

Item elements as list paragraphs

The `item` element can also occur within a `block` element. In this context, it is intended to operate as a list.

In the Elkera Topic DTD the type of list to be created (numbering scheme to be used) is determined by an attribute on the first item element in the list. A `numbering.scheme` attribute is provided in the simplified model. It is proposed that a single list element is sufficient to avoid confusion about various list elements and to simplify switching between types. Frequently, authors wish to switch from numbered to bulleted lists and vice versa.

The proposed model does not include a list container. Paragraph level lists are created by including item elements directly inside a block.

There are arguments in favour of including a list container. It can be useful to specify attributes for the list type, for example. This can be more convenient than specifying this on the first item in the list. Disruption might occur when a new item is inserted before the first item without re-specifying the list type.

Rather than specify this information on a list container, it can be specified just as easily on the containing `block`. This is handled by the application and should not involve the author. On this basis, a separate list container is considered to be redundant.

num

The `num` element contains numbers for `topic` and `item` elements.

The proposed clause model makes no assumptions about the numbering scheme to be used or how it will be implemented. This is to be provided by the user's supporting applications. Clearly additional attributes would be required on the `topic` and `item` elements to provide numbering scheme control and to allow use of automatic or fixed numbering.

In the simplified content model the `num` element precedes the `title`. There is no reason why the order cannot be reversed to deal with some structures. This is not considered significant for the current exercise.

title

The `title` element is a heading, caption or title for the `topic` and `item` elements.

The `title` is always within the main container element so the markup always defines the content to which the title relates.

block and text

The `block` and `text` elements are the basic elements of the proposed model. Both of these must be included to create text content for a document. For example:

```
<block><text>The quick brown fox jumps over the lazy dog.</text></block>
```

The `block` element cannot be directly numbered. If a numbered structure is desired, either a `topic` or `item` must be used.

The `block` element may never directly contain text data. The `block` element must contain at least a `text` element which contains the text data. `block` element will also contain tables, block graphics and other similar components. It provides a container for all nested components so they may be manipulated as a group.

It is important that list is contained within the `block` element so that a continuation of the paragraph after the list can be distinguished from a new paragraph or block. This is shown in the following example:

```
<topic><num>1.</num><title>Foxes and lazy animals</title>
<block>
<text>The quick brown fox jumps over the lazy:</text>
<item><block><text>dog;</text></block></item>
<item><block><text>cat;</text></block></item>
<item><block><text>rabbit,</text></block></item>
<text>and any other animal that may be sleeping.</text>
</block>
<block>
<text>More alert animals won’t be caught so easily.</text>
</block>
</topic>
```

The `text` element could be dispensed with as redundant in most situations. The example above could be marked up as follows:

```
<topic><num>1.</num><title>Foxes and lazy animals</title>
<block>The quick brown fox jumps over the lazy:
<item><block>dog;</block></item>
<item><block>cat;</block></item>
<item><block>rabbit,</block></item>
and any other animal that may be sleeping.</block>
<block>More alert animals won’t be caught so easily.</block>
</topic>
```

The `text` element is included to avoid this mixed content and ensure control over chunks of text that occur after other elements for consistent processing. It provides explicit markup of the introductory text of a paragraph before a list and the continuing text of the paragraph after the list. From experience, this substantially assists in application development to have a clear distinction between inline and block content. In some cases, the absence of an explicit element makes it impossible to determine whether to render the chunk of text inline or on a new line. For example, a formula may be followed by the word “where:” to introduce the definitions of formula components. Use of the `text` element allows the data to specify that this continues in line rather than the default position of starting a new line for text objects.

There is a balance between the apparent redundancy of the text element and its convenience for processing and in dealing with unusual situations. In practice, the presence of the text element is rarely, if ever, an inconvenience to authors. XML editors will include the `text` element with the `block` element so the author rarely has to deal with it explicitly.

It is submitted that the balance of convenience is in favour of use of the `text` element.

The `text` element may be repeated inside a `block` to create the equivalent of a new line. This may be used to create lists of objects that are not numbered and align to the left margin for the containing block. It is intended that the `text` element should not be repeated to create a new grammatical paragraph. Some misuse in inevitable.

Recursive vs non recursive model

The proposed model provides for the creation of a document hierarchy using `topic` or `item` recursively. While a recursive model might add some complexity to some applications, it allows for extremely flexible re-use of content at different levels of the hierarchy and it is simple for authors.

Fit to requirements

	Summary of requirements	Fit
1.	Markup the core structures found in documents like Attachment 1 to the Requirements.	Meets
2.	Represent the structured hierarchy of the content.	Meets
3.	Represent terms of benchmark contracts.	To be demonstrated when TC's benchmark contracts are collated.
4.	Define clause objects as self contained objects.	Meets
5.	Self contained markup so that a text file could provide complete contract terms.	Meets. All content, including component numbers are explicitly included.
6.	Must avoid listed terms for element names.	Meets
7.	Must permit markup of contract terms without inclusion of any legal semantic markup or annotation.	Meets
8.	Must be as simple as practicable to facilitate user training, support and application development.	Uses only 6 elements. Authors select elements to satisfy easily understood functional and structural needs (should it be in the contents, should it have a title?"). No need to consider semantic distinctions about clauses, subclauses and lists.
9.	Must be able to re-use content in different levels of the hierarchy, without having to change names of elements.	Meets, subject to possibility that an author may wish to convert between topics and items during authoring. The reasons for this are discussed earlier. All content is located in a `block` element that can be re-used anywhere in the hierarchy.

	Summary of requirements	Fit
10.	Must allow clauses or other content to be incorporated into a document by reference.	Does not address this issue at this stage. A means to do this is easily added to the proposed model without impact on the structure of the proposal.
11.	Other listed features to be determined.	Not considered at this stage. Deferred until requirements are fully developed.

DTD and XML markup example

The simplified clause model and example from the Requirements are set out below. The example is also included in a separate text file.

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE topic [
<!ELEMENT topic (num?, title, (topic+ | item+ | block+))>
<!ELEMENT item (num?, title?, (item | block)+)>
<!ATTLIST item numbering.scheme CDATA #IMPLIED >
<!ELEMENT block (text | item)+ >
<!ELEMENT text (#PCDATA) >
<!ELEMENT num (#PCDATA) >
<!ELEMENT title (#PCDATA) >
]>
<topic> <num>1.</num><title>Provisions about the
specification of colours in contracts</title>
<topic><num>1.1</num><title>Spectrum colours</title>
<block><text>Here is a contrived, complex list structure
using the spectrum colours and one or two others:</text>
<item numbering.scheme="default">
<num>(a)</num><block><text>red,</text></block></item>
<item>
<num>(b)</num><block><text>orange,</text></block></item>
<item>
<num>(c)</num><block><text>yellow,</text></block></item>
<item>
<num>(d)</num><block><text>green,</text></block></item>
<item>
<num>(e)</num><block><text>blue, including:</text>
<item numbering.scheme="default">
<num>(i)</num><block><text>pale blue,</text>
</block></item>
<item>
<num>(ii)</num><block><text>dark blue,</text>
</block></item>
<text>but excluding violet,</text></block></item>
<item>
<num>(f)</num><block><text>indigo, and</text></block>
</item>
<item>
<num>(g)</num><block><text>violet,</text></block></item>
<text>from which all colours can be derived.</text></block>
</topic>
<topic>
<num>1.2</num>
<title>CMYK colours</title>
<block><text>CMYK colours (cyan, magenta, yellow and black)
are normally specified for inputs to colour printing
processes.</text></block>
</topic>
<topic>
<num>1.3</num><title>RGB colours</title>
<item>
<num>1.3.1</num><block><text>RGB colour (red, green, blue)
specifications are used for computer screen displays.</text>
</block></item>
<item>
<num>1.3.2</num><block><text>Using only these 3 colours,
you can specify any colour.</text></block></item>
<item>
<num>1.3.3</num><block><text>The number of colours you can
```

```
specify depends on the colour depth available. For
example:</text>
<item numbering.scheme="default">
<num>(a)</num><block><text>8 bit colour can render 256
colours;</text></block></item>
<item>
<num>(b)</num><block><text>16 bit colour can render 65,
536 colours.</text></block></item>
</block></item></topic>
<topic>
<num>1.4</num><title>Using black and white</title>
<item>
<num>1.4.1</num><title>Greyscale</title>
<block><text>The number of greys depends on the available
colour depth, as for other colours.</text></block></item>
<item>
<num>1.4.2</num><title>Black and white</title>
<block><text>This is really called monochrome. You can
specify either:</text>
<item numbering.scheme="bullet">
<num>&#x2022;</num><block><text>black, or</text>
</block></item>
<item>
<num>&#x2022;</num><block><text>white.</text></block>
</item>
</block>
</item></topic>
<topic>
<num>2.</num><title>Colour profiles</title>
<block><text>One thing to remember is that when working with
colours, always use a colour profile that is available for
your display or output device. This will ensure you achieve
the most consistent results.</text></block></topic>
</topic>
```