Alternative clause model proposal

Partial alternative to proposal from Jason Harrop

Submitted by Peter Meyer (pmeyer@elkera.com.au)
Wednesday, October 15, 2003.

1. Purpose of this document

Jason Harrop was asked by the TC to evaluate clause model proposals from TC members and submit a recommended model for consideration by the TC.

During his work, Jason consulted with Peter Meyer in recognition of his proposed adoption of substantial components of the clause model proposal submitted by Peter Meyer in July 2003. During those discussions both parties attempted to reach a consensus on the model to be presented. The outcome of that process is that Peter Meyer supports the core features of Jason's proposal but opposes some aspects of it.

This document identifies the particular matters on which Jason and Peter have agreed to disagree and recommends that the TC adopts a modified version of the model proposed by Jason.

In addition, topic 8. [Using the clause model in practice] discusses the flexibility of the model and how it can be extended to overcome some practical limitations that may arise because of the desire to make the model as simple as possible for general acceptance. Some users and developers might wish to extend the model to deal with those limitations. These extensions are not necessarily supported by Jason Harrop and are included to provide a full explanation of issues affecting the proposed clause model.

2. Alternative model

2.1 DTD version

The alternative model, ignoring entities, is as follows:

```
<!Element Item (Num?, Title?, Para*, Item*)>
<!ELEMENT Para (Text | Item)+>
<!ELEMENT Text (#PCDATA)>
<!ELEMENT Num (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
```

2.2 XSD Schema version

The proposed Schema modifies the content model of item inside para to prevent direct item recursion inside items as lists. The Schema is set out in Appendix A.

2.3 Summary of key differences to Jason Harrop recommendation

2.3.1 This model differs from the model proposed by Jason in these respects:

(a) The model does not include the `ListItem` element for list items. Rather, the `Item` element is used fully recursively to serve this function.

(b) It is proposed that the `Item` element should be redefined in the XML Schema to avoid the identified problem with the recursive model in list structures.

(c) It is proposed that it is not necessary for DTD support to provide the same functionality as that provided by Schemas. The Schema should be the normative version of the grammar and its full features should be available without regard to the limitations of DTDs.

(d) The model omits the `List` element in the content model for `Para` proposed by Jason.

(e) The `TextBlock` element should be renamed as `Text`.

2.3.2 Jason’s recommendation 4 is opposed. The Architectural guidelines should not be adopted.

2.3.3 Various other recommendations should not be formally adopted.

2.3.4 The reasons for these alternatives are discussed in the following topics.

3. The ListItem element should be omitted

3.1 Background

In the clause model requirements, section 5.3, it was explained that the concept of a list is imprecise in terms of markup. In section 5.5 of the requirements it was stated that:

> “There is no clear distinction between a clause, subclause, list item or list paragraph. Any of these terms might be applied to object 1.3.1 in the example.
>
> Authors should not have to consistently apply markup that makes subtle distinctions as to whether objects are clauses, subclauses or lists since these terms have no clearly defined meaning in structural markup.”

Requirement 8 of the clause model requirements states:

> “The clause model must be as simple as practicable to facilitate user training, support and application development. To make it easy for users, semantic distinctions between generic objects will only be introduced where the benefits are clearly demonstrable.”

3.2 What is a list?

In terms of the proposed clause model one definition of a list would be a series of elements, usually numbered or bulleted, that are contained by a `Para` element. Usually, but not always, the list items will be preceded by introductory text, as in the following example (using simplified markup):

Example 3.2-1

```
<Para><Text>This is a list of spectrum colours:</Text>
<Item>(a)        red</Item>
<Item>(b)        orange</Item>
<Item>(c)        yellow</Item>
<Item>(d)        green</Item>
<Item>(e)        blue</Item>
<Item>(f)        indigo</Item>
<Item>(g)        violet.</Item>
</Para>
```

In that example, the introductory text and the listed items are contained by one `Para` element.

Under the proposed clause model, it would be possible for an author to create this structure (using simplified markup):

Example 3.2-2

```
<Para><Text>This is a list of spectrum colours:</Text></Para>
<Item>(a)        red</Item>
<Item>(b)        orange</Item>
<Item>(c)        yellow</Item>
<Item>(d)        green</Item>
<Item>(e)        blue</Item>
<Item>(f)        indigo</Item>
<Item>(g)        violet.</Item>
```

In this example, the listed items are outside the introductory `Para` element. Particularly if numbered in the same way as example 3.2-1, many people would still regard this as a list.

One of the design philosophies underpinning the clause model requirements is that the clause model should avoid creating named semantic structures such as clause, subclause and list item. As we can see from Examples 3.2-1 and 3.2-2, minds can differ as to what constitutes a list. It is submitted that it is highly undesirable for the markup to use named elements to reflect these sort of subtle distinctions. Under Jason's proposal, the items in example 3.2-1 would be replaced with `ListItem` elements. To what end?

3.3 Stated reasons for the ListItem element

Despite the definition problems and the specifics of requirement 8, Jason proposes use of a `ListItem` element. It would be used in the case of Example 3.2-1 but not in the case of Example 3.2-2. The justification offered is set out in Attachment 2 to Jason's proposal. Essentially, it is to overcome the problem that lists could be created in the way shown above or by allowing items to occur directly inside other items. Jason then says:

> "Note that the possibility of using a W3C XML schema with distinct global and local definitions of <Item> was considered. I reject this possibility since it contravenes the architectural guidelines and has no discernable virtues to redeem it."

3.4 The issues

3.4.1 It is desirable that the problem described in Attachment 2 to Jason's proposal is avoided. It can be avoided simply be re-defining the element in the `Para` context using a Schema. This re-definition would tighten the content model. However, the problem is not particularly serious. It is possible that some users may create list structures incorrectly, if they are working without an XML editor. With an editor, they will soon discover their error because their numbering is unlikely to work correctly. In most XML authoring applications the problem will be avoided because `Item` elements inserted within the `Para` context should also include the contained `Para` element, thus preventing the author from making the mistake.

3.4.2 There are four possible approaches to dealing with the problem described. These are considered in the following table.

Approach to problem	Advantages	Disadvantages
Allow alternative proposal, using DTD only. Content model of `Item` does not change.	Complete element re-use (Requirement 9).	Possibility of authors creating lists in two ways.
Use `ListItem` element (Jason proposal)	Avoids undesired structures.	Content re-use requires element transformation (violates requirement 9).
Re-define `Item` in `Para` context using Schema to restrict direct recursion of `Item` elements.	• no need to transform element name. • Avoids undesired structure. • Item elements from list contexts can always be moved into other contexts. • Item elements can be moved into list contexts unless they are only valid under the loose model for `Item`.	Possible difficulty in moving `Item` elements into list context if the loose model is used.
Re-define `Item` in `Para` context in Schema as for option 3, plus use the "tight" content model for `Item`, as discussed in Jason's proposal and in topic 8.3 of this document.	• no need to transform element name. • Avoids undesired structure. • Complete element re-use (Requirement 9).	None.

3.4.3 It is submitted that of the four options, that using the `ListItem` element is the worst overall. It clearly violates requirement 9.

3.4.4 The `ListItem` element adds nothing to the markup that is of value to a user or to processing systems. It only prevents authors from creating list structures in a way that is not desired when a DTD is used. Using the Schema approach, all processing applications can distinguish the two examples described in topic 3.2 by reference to the containing elements (Are the items within a para element?). This allows applications to deal with discrete content units, control automatic numbering and presentation issues, as desired.

3.4.5 Using a Schema, it is possible to alter the content model of the `Item` element according to whether it is or is not contained by a `Para` element. This is a specific feature of XML Schema that increases their flexibility over DTDs. This eliminates the problem that prompted the introduction of the `ListItem` element into Jason’s proposed clause model.

3.4.6 Jason proposes architectural guideline 2:

> “Where an XML schema is used, an element should not be defined twice with the same name but a different content model.”

The explanation given for this rule is that it inhibits re-usability because the element may not be valid in other contexts and that it may confuse users.

3.4.7 The reasons given in support of architectural guideline 2 do not support a general rule. It may be true that in some cases the re-defining of an element will have undesirable consequences. This is not universally true. As demonstrated by the analysis in the table in item 3.4.2, the content model of `Item` in the list context will be tighter than in other contexts. Only when moving an `Item` from the main hierarchy to a list context using the loose model can the target location be invalid. This will only occur if the content of the item conforms only to the loose model. It is submitted this is by far the lesser evil than requiring a transformation every time an element crosses the list context boundary.

3.4.8 It is highly undesirable to include in the model an element that serves no useful purpose. It is hoped that the TC will produce a standard that will endure for many years without the need to make changes that will invalidate then existing markup. If the `ListItem` element is included, every developer in the future is going to ask "Why is this element here, surely it is redundant?". Will the answer be: "Well, the people that created it knew about XML Schema but wanted the standard to be fully implementable in DTDs and its bad luck hardly anyone uses DTDs anymore"?

3.4.9 In conclusion, it is submitted that the `ListItem` element should be omitted from the clause model and its place taken by the `Item` element.

4. The grammar should not be constrained by

limitations of a DTD

4.1 Background

4.1.1 Jason proposes use of Schema as the normative version of the clause model However, in his proposal, Jason has proposed architectural guideline 1:

> "The grammar should be able to be expressed without error in each of the schema languages the TC adopts."

No specific reasons are given for this guideline.

4.2 The issues

4.2.1 DTDs are a nice, but limited tool. Unlike Schema, they are concise and easy for humans to read. Unfortunately, they have always suffered from the problem encountered in this case. They do not allow the content models of elements to be changed according to context.

4.2.2 It is submitted that increasingly, XML Schema will replace DTDs in content authoring and publishing systems. This process has been relatively slow to date because most of the more powerful XML tools are derived from SGML based tools that we necessarily based on DTDs. Most XML applications now support Schema and it is expected that this process will continue.

4.2.3 While it is early to tell, it is highly likely that other layers of the proposed standard will be best implemented via Schema. This will provide the greater flexibility needed to implement the kinds of data structures that will be needed.

4.2.4 Jason's proposed architectural guideline 1 would limit us for all time to the functionality provided by an obsolete tool.. This is not realistic. Sooner or later, we need to be able to take advantage of the benefits offered by new tools that are developed for the very purpose of overcoming the kind of problems encountered here.

5. Omission of the list container element

5.1 Background

5.1.1 Jason's proposal requires use of a `List` container element. The reasons given for this include:

(a) It will make sense to users (page 12);

(b) It may be useful for XSLT processing (page 12 and Attachment 3, item 5);

(c) People are getting used to a concept of a list (Attachment 3, item 1);

(d) It is convenient for specifying numbering properties, particularly if there are two lists in a `Para`;

(e) It may be useful for vertical applications which we can't predict;

5.2 Issues

5.2.1 It is submitted that elements should be added only if they serve a clear purpose.

5.2.2 Some of Jason’s reasons are irrelevant, particularly (a) and (c) above. Others are speculative, particularly (d) and (e).

5.2.3 The `List` element will be effectively invisible to authors because it does not contain data. Using an XML editor, authors will not normally need to access attributes directly to control list numbering. More likely, this will be done through other interfaces. The author will not miss a non existent list container.

5.2.4 The existence of a `List` container conflicts with the principles set out in section 5.5 of the clause model requirements. It explicitly defines a list structure when this distinction is to be avoided.

5.2.5 There may be some processing advantages from use of a `List` container. It is submitted that these advantages to do not outweigh the inconvenience of adding a container that serves an extremely limited purpose and that adds another step to content re-use at different levels of the document (requirement 9). User convenience should be favoured over processing convenience.

5.2.6 It is submitted that the case for the `List` element is not made out. It appears to be a redundant element.

6. Element name for TextBlock

While only a cosmetic issue, it is significant. The name "TextBlock" is ugly. It seeks to introduce the programmers conception of the element as a block element into the element name. For what purpose? Why use two words when one will suffice?

It is submitted that a shorter, simple name "Text" is sufficient and desirable for a model that we hope will be in front of a large audience of non technical persons.

7. The architectural guidelines

7.1 Jason has proposed 7 architectural guidelines and proposes that these should be formally adopted by the TC.

7.2 It has already been discussed earlier that guidelines 1 and 2 are flawed.

7.3 Guidelines 3 and 4 are not directly relevant to the basic clause model. The principles espoused will need to be considered as part of the work on requirement 11 issues. Some of these issues have already been canvassed, as indicated by Jason. For example, Jason proposes a BlockQuotation element in the discussion under Requirement 11. This approach is vigorously opposed. The proposal may have the effect of foreclosing proper analysis before the analysis is actually done.

7.4 Guideline 5 may be relatively unobjectionable but guideline 6 is a bootstraps argument for creating a `List` container when the `Para` element will do just as well. Guideline 7 could arguably justify anything.

7.5 Overall, there is no reason to formally adopt these guidelines. There is a danger they will limit options on issues yet to be addressed.

8. Using the clause model in practice

8.1 Background

There are two features of the proposed clause model that should be understood by prospective users of the model. In both cases, some users may wish to extend the clause model to enforce stricter markup rules with the objective of providing a more consistent and easily processed data set. The two features are:

(a) The model does not provide a way to ensure that `Items` have `Title` elements where it is desired that contents listings should be consistently generated to organisational standards.

(b) The model allows irregular hierarchical structures by allowing `Para` and `Item`elements at the same level. This can adversely affect generation of contents listings and document processing, particularly for more complex web publishing where it is desired to "chunk" pages based on defined elements of the markup.

In the following topics possible extensions to the clause model are proposed to deal with these issues. It is proposed that these extensions are not part of the core model or exchange standard but that they be officially recognised in the standard as optional extensions. As Jason indicated in his proposal, dealing with point (b) above, it will not be possible to prevent users from adopting the extensions. By officially recognising them and explaining the issues surrounding them the standard will be more fully described and it will be more flexible and useful.

The arguments advanced by Jason for not recognising the tight model for `Item` are not convincing. There is no likelihood of confusion if the issues are squarely addressed in the standard. There is no issue that the loose model would become the defacto standard. It is the standard.

8.2 Enforcing heading consistency

8.2.1 The problem

The item element does not require the use of Titles. Many documents, particularly contracts are expected to use consistent headings or titles for clauses and major hierarchical groups. These titles are important for:

(a) identification of re-usable components in database systems;

(b) readability of the document; and

(c) contents generation in both print and online publications.

If titles are not present on some items, this detracts substantially from publishing and processing functionality if the application expects titles to be present.

If the clause model schema does not enforce the use of titles but they are desired by the application, enforcement rules must be built into authoring applications. This adds unnecessarily to the complexity of those applications.

In addition, applications working with documents have to carry out an analysis of the document to determine the number of levels in the document that have titles and the consistency with which they are provided before they can determine how to extract contents listings.

Overall, these problems result in the data not reliably reflecting a structure that is needed for convenient processing and publishing. They place a significant burden on applications and result in unreliable processing. In many user sites, these issues may not matter. In others they will be significant.

8.2.2 DTD version of the extension

It is proposed that another container element called `Topic` is provided. This element would by recursive and operate in a similar way to the `Item` element except that it would require a `Title` element and it could not exist inside a `Para`.

The content model for the `Topic` element is as follows:

```
        <!ELEMENT Topic (Num?, Title, (Topic+ | Item+ | Para+))>
```

It is part of the flexibility of the proposed clause model that it can be so easily extended in this way.

8.2.3 Issues associated with the Topic element

The Topic element is an alternative to the Item element in the main document hierarchy. The principal reason that it was not included in the clause model is that users will from time to time need to translate between Topic and Item elements where an incorrect choice has been made or content is to be used in the other context. This was in conflict with requirement 9.

Despite this limitation, the selective use of the Topic and Item elements in template documents provides user enterprises with the ability to guide authors down the desired path and to gain substantially higher quality, consistent markup that will support reliable, automated processing.

It would be relatively straight forward for developers who wish to support the Topic extension to provide translation macros or wizards to translate between Topic and Item markup in the authoring environment.

Users of the Topic extension could not expect that other users would necessarily use the same model. The standard for exchange of XML data with other organisations would be based on the proposed Item model. Organisations using the Topic may need to translate data to that model. This is a very simple translation task.

8.3 The tight content model for Item

8.3.1 The problem

The Item element may contain multiple Para elements, followed by multiple Item elements. When this occurs in the main document hierarchy, the following problems can arise:

(a) The content of the Para elements does not appear in a contents listing. Contents listings become misleading.

(b) Large documents cannot be easily chunked into discrete pages for web publications. Applications must deal with content that sits between the hierarchical levels.

Overall, these problems result in the data not reliably reflecting a structure that is needed for convenient processing and publishing. Again, in many user sites, these issues may not matter. In others they will be significant.

8.2.2 DTD version of the extension

The content model for the tight model is as follows:

```
<!ELEMENT Item (Num?, Title?, (Item+ | Para+))>
```

Again, it is part of the flexibility of the proposed clause model that it can be so easily extended in this way.

8.3.2 Issues associated with the tight model

The tight model can be easily implemented by users without affecting compatibility with the standard. All tight model data must conform to the standard.

Use of the tight model also allows users to eliminate any possible disadvantages of re-defining the Item element in the Schema, as discussed in the table in topic 3.4.

It is true that users must be able to process standard data received from other parties. However, this should not inhibit them from using the tight model for their own authoring systems. It would be expected that most organisations, properly advised, would implement the tight model for documents such as contracts and other regularly structured documents.

9. Recommendations

9.1 It is proposed that the TC adopt the clause model proposed by Jason Harrop with the following changes:

(a) Omit the `ListItem` element and replace with the `Item` element in the content model for element `Para`;

(b) Adopt a Schema definition for the clause model that restricts the content model of `Item` inside `Para` to avoid direct recursion of items in that context.

(c) Omit the `List` element at this time.

(d) Rename the `TextBlock` element to `Text`.

9.2 It is proposed that the TC should not adopt the proposed Architectural guidelines (Recommendation 4).

9.3 It is proposed that the TC should not adopt Recommendations 5 to 8 inclusive. These are dealt with by adoption of the overall model, subject to proposed changes.

9.4 It is proposed that the standard officially recognise that use of the extension for the "tight" content model for the `Item` element may be desirable for many users.

This will assist prospective users to better understand the operation and scope of the standard.

9.5 It is proposed that the standard officially recognise that use of the proposed `topic` element extension may be desirable for many users. This will assist prospective users to better understand the operation and scope of the standard.

9.6 It is proposed that the TC review the need for the `List` element after requirement 11 is fully developed.

10. Fit to requirements

The alternative model differs only slightly from that proposed by Jason. It is submitted that the proposed changes will ensure substantially better compliance with requirements 8 and 9.

Clause model Schema and XML markup example

The proposed schema is set out below.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:annotation>
<xsd:documentation xml:lang="en">
Proposed Schema for eContracts TC clause model
</xsd:documentation>
</xsd:annotation>
<xsd:element name="Num" type="xsd:string"/>
<xsd:element name="Title" type="xsd:string"/>
<xsd:element name="Item">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="Num" minOccurs="0" maxOccurs="1"/>
<xsd:element ref="Title" minOccurs="0" maxOccurs="1"/>
<xsd:element ref="Para" minOccurs="0"
maxOccurs="unbounded"/>
<xsd:element ref="Item" minOccurs="0"
maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
<xsd:element name="Para">
<xsd:complexType>
<xsd:choice minOccurs="0" maxOccurs="unbounded">
<xsd:element ref="Text"/>
<xsd:element name="Item">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="Num" minOccurs="0" maxOccurs="1"/>
<xsd:element ref="Title" minOccurs="0" maxOccurs="1"/>
<xsd:element ref="Para" minOccurs="1"
maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:choice>
</xsd:complexType>
</xsd:element>
<xsd:element name="Text" type="xsd:string"/>
</xsd:schema>
```

The markup example from the Clause Model Requirements is set out below.

```
<Item><Title>Markup example</Title>
<Item> <Num>1.</Num><Title>Provisions about the
specification of colours in contracts</Item>
<Item><Num>1.1</Num><Title>Spectrum colours</Title>
<Para><Text>Here is a contrived, complex list structure
using the spectrum colours and one or two others:</Text>
<Item>
<Num>(a)</Num><Para><Text>red,</Text></Para></Item>
<Item>
<Num>(b)</Num><Para><Text>orange,</Text></Para></Item>
<Item>
<Num>(c)</Num><Para><Text>yellow,</Text></Para></Item>
<Item>
<Num>(d)</Num><Para><Text>green,</Text></Para></Item>
<Item>
```

```
<Num>(e)</Num><Para><Text>blue, including:</Text>
<Item>
<Num>(i)</Num><Para><Text>pale blue,</Text>
</Para></Item>
<Item>
<Num>(ii)</Num><Para><Text>dark blue,</Text>
</Para></Item>
<Text>but excluding violet,</Text></Para></Item>
<Item>
<Num>(f)</Num><Para><Text>indigo, and</Text></Para>
</Item>
<Item>
<Num>(g)</Num><Para><Text>violet,</Text></Para></Item>
<Text>from which all colours can be derived.</Text></Para>
</Item>
<Item>
<Num>1.2</Num>
<Title>CMYK colours</Title>
<Para><Text>CMYK colours (cyan, magenta, yellow and black)
are normally specified for inputs to colour printing
processes.</Text></Para>
</Item>
<Item>
<Num>1.3</Num><Title>RGB colours</Title>
<Item>
<Num>1.3.1</Num><Para><Text>RGB colour (red, green, blue)
specifications are used for computer screen displays.</Text>
</Para></Item>
<Item>
<Num>1.3.2</Num><Para><Text>Using only these 3 colours,
you can specify any colour.</Text></Para></Item>
<Item>
<Num>1.3.3</Num><Para><Text>The Number of colours you can
specify depends on the colour depth available. For
example:</Text>
<Item>
<Num>(a)</Num><Para><Text>8 bit colour can render 256
colours;</Text></Para></Item>
<Item>
<Num>(b)</Num><Para><Text>16 bit colour can render 65,
536 colours.</Text></Para></Item>
</Para></Item></Item>
<Item>
<Num>1.4</Num><Title>Using black and white</Title>
<Item>
<Num>1.4.1</Num><Title>Greyscale</Title>
<Para><Text>The Number of greys depends on the available
colour depth, as for other colours.</Text></Para></Item>
<Item>
<Num>1.4.2</Num><Title>Black and white</Title>
<Para><Text>This is really called monochrome. You can
specify either:</Text>
<Item>
<Num>&#x2022;</Num><Para><Text>black, or</Text>
</Para></Item>
<Item>
<Num>&#x2022;</Num><Para><Text>white.</Text></Para>
</Item>
</Para>
</Item></Item>
<Item>
```

```
<Num>2.</Num><Title>Colour profiles</Title>
<Para><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></Para></Item>
</Item>
```