Report to OASIS LegalXML eContracts TC

Structural Markup – "Basic" Clause Model Recommendations

Author: Jason Harrop, SpeedLegal (jharrop@speedlegal.com)

Status: Draft 1.0 dated Tuesday, October 14, 2003

Acknowledgements

Peter Meyer (pmeyer@elkera.com.au), Andrew Squire (asquire@elkera.com.au), and Justin Lipton (justin@speedlegal.com) contributed substantially in deliberations leading to the development of this report, though they may or may not agree with the recommendations.

Table of Contents

Purpose of this document

This document is my report to the Technical Committee recommending a "basic" clause model, which I was charged with producing as part of the process described in more detail below. It takes into account the proposals contributed via the TC's mailing list, as described below.

Once a "basic" clause model has been agreed, the next steps are to extend it into a "complete" clause model, and then into a grammar suitable for the entirety of a contract document.

The solution space to the clause model problem is large. The advantages and disadvantages one solution may have over another are not always obvious, and can only be weighed subjectively. Consequently, the recommendations in this document reflect the judgement of its author, which is likely to be different to that of some other TC members.

Summary of Recommendations

Recommendation 1: The TC develop its own grammar for representing the clause model, rather than use an existing DTD.

Recommendation 2: The TC publish its grammar as a W3C XML schema, in Relax NG, and (subject to recommendation 12) as a DTD.

Recommendation 3: The TC publish the normative version of its grammar using either W3C XML schema, or Relax NG.

Recommendation 4: The TC adopt each of the Architectural Guidelines specified herein.

Recommendation 5: The TC represent clause structures using a recursive hierarchy, with one and only one container at each level (but permitting a List hierarchy inside each level in a manner which does not allow the main recursive hierarchy to be resumed from inside a List).

Recommendation 6: The concept of a List be included in the grammar.

Recommendation 7: The TC adopt the suggested grammar (using its preferred schema language).

Recommendation 8: The TC should not officially sanction the so-called "tight" model as an alternative to the "loose" model recommended here.

Recommendation 9: Consistent with UBL, names for XML constructs *must* use camel-case capitalization, such that each internal word in the name begins with an initial capital followed by lowercase letters (example: **TextBlock**). All XML constructs other than attributes *must* use upper camel-case, with the first word initial-capitalized, while attributes *must* use lower camel-case, with the first word all in lowercase (example: **unitCode**), with the exception of ID.

Recommendation 10: Incorporation of Items by reference be defered to requirement 11 stage, pending further discussion.

Recommendation 11: Whether to markup inline lists, and how, to be considered as part of requirement 11.

Recommendation 12: Namespace handling and extension mechanism be considered at the requirement 11 stage.

Background

On 1 May 2003, Dan Greenwood posted an email entitled "Proposed Process", which among other things proposed that the requirements for the clause model be documented and agreed: http://lists.oasis-open.org/archives/legalxml-econtracts/200305/msg00001.html

A requirements document was produced and posted to the list on 27 May 2003:

http://lists.oasis-open.org/archives/legalxml-econtracts/200305/msg00027.html

and approved in the 18 June 2003 teleconference, as documented in the minutes:

http://lists.oasis-open.org/archives/legalxml-econtracts/200307/msg00002.html

The clause model requirements document set out a staged approach to development, so that the issues to be considered at each stage are limited. The stages are:

(a) Develop “basic” clause model (ie excluding objects identified in requirement number 11, other than stub or placeholder elements).

(b) Once the basic clause model is agreed, develop requirements for issues set out in requirement number 11.

(c) Develop “complete” clause model (ie covering the requirements developed for the issues set out in requirement number 11)

(d) Develop requirements for markup of complete contract documents using the clause model.

A process for delivering on the clause model requirements document was proposed in the agenda for that teleconference:

http://lists.oasis-open.org/archives/legalxml-econtracts/200306/msg00016.html

That process was as follows:

"that the TC develop a Clause Model addressing those requirements, according to the following process:

> (a) interested TC members to forward proposals to the mailing list which satisfying all or part of the requirements (a 3 week period concluding 9 July).
>
> Such proposals could be complete solutions, or contribute partial or point solutions, intended to be used as part of a complete solution.
>
> (b) Jason Harrop to prepare a report to the TC recommending a clause model

which takes those contributions in to account. The report is to include the clause model and a summary of the contributions received. (it is expected this report could be completed in 4 weeks from 9 July, namely 6 August).

(c) draft report to be circulated to TC members who had contributed proposals as input to that work, for comment and fine tuning, before surfacing on the mailing list

(d) mailing list and teleconference discussion, and possible further fine tuning

(e) TC to vote to accept the report. "

The 18 June 2003 teleconference also approved that process.

This document is the report referred to in (b). A preliminary draft has already been circulated as per (c).

We are currently up to (d).

Methodology

Contributions Received

Proposals were received from the following TC members as input into the process:

- John McClure: http://lists.oasis-open.org/archives/legalxml-econtracts/200306/msg00019.html , http://lists.oasis-open.org/archives/legalxml-econtracts/200307/msg00016.html
- Jason Harrop: http://lists.oasis-open.org/archives/legalxml-econtracts/200307/msg00004.html
- Peter Meyer: http://lists.oasis-open.org/archives/legalxml-econtracts/200307/msg00005.html

Dr Leff referred us verbally to his previous proposals.

The contributions submitted by Jason Harrop and Peter Meyer were quite similar. Since the recommendations made in this report draw heavily on these proposals, I do not discuss them further here.

The contribution submitted by John McClure suggested an RDF based approach.

The clause model requirements document contained the following requirement:

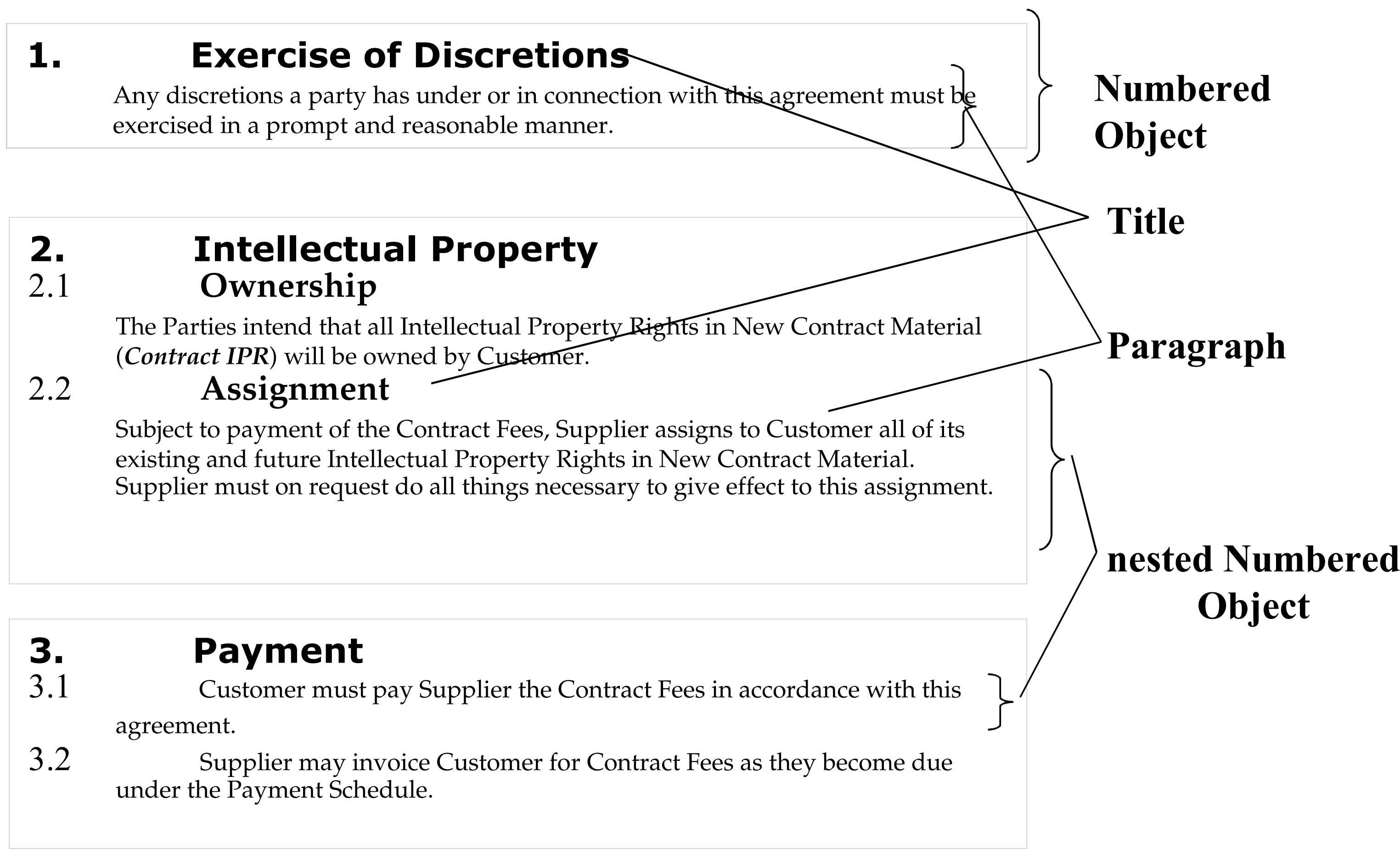
> 8. The clause model must be as simple as practicable to facilitate user training, support and application development.

I was unable to see how an RDF based approach addressed this requirement, as compared to more traditional approaches to document-centric design. Indeed, it raised a number of problems. In summary:

- no discernable advantages over traditional approaches to document-centric design
- the model is much looser than it needs to be to model the content, which means it is not suitable for use in popular XML editors (eg XMetaL, Altova's Authentic, Microsoft Word 2003) without more customisation than the other proposals required
- it uses rdf:Bag, for example, as a possible container for list items. In the rdf:Bag container, ordering is not significant. In a contract, the ordering of paragraphs is significant.
- the proposal explains that the convention for including a caption (ie a title) within a "Block" element is to place it in an rdf:li in an rdf:Bag element. This means a clause can have multiple titles, which I have not seen in contract documents, nor would I expect stylesheets to generally anticipate.
- it appears to require a document author to specify formatting semantics which are to apply to its "Block" element (candidates suggested in the proposal included "TitledParagraph", and "IndentedParagraph"). The document author does not need to worry about this using the other proposals.

Underlying Clause Pattern

Here are some examples of clauses:



1. Exercise of Discretions

Any discretions a party has under or in connection with this agreement must be exercised in a prompt and reasonable manner.

2. Intellectual Property

2.1 **Ownership**

The Parties intend that all Intellectual Property Rights in New Contract Material (***Contract IPR***) will be owned by Customer.

2.2 **Assignment**

Subject to payment of the Contract Fees, Supplier assigns to Customer all of its existing and future Intellectual Property Rights in New Contract Material. Supplier must on request do all things necessary to give effect to this assignment.

3. Payment

3.1 Customer must pay Supplier the Contract Fees in accordance with this agreement.

3.2 Supplier may invoice Customer for Contract Fees as they become due under the Payment Schedule.

As the proposals from Harrop and Meyer recognised, the underlying clause pattern we need to model looks like:

Numbered Object container
- Title
- Paragraph
- (List)
- nested Numbered Object<u>**s**</u>

Existing DTDs

A first step was to consider whether any existing DTD was a suitable solution to the clause model requirements.

I considered the following:

- DITA
- DocBook
- OpenOffice
- UBL
- WordML
- XHTML

I asked whether the DTD in question could model the underlying clause pattern identified above, and also considered the following non-functional requirements:

- End user skill set
- Developer skill set
- Tool support - editing environment
- Tool support - stylesheets
- Extension mechanism
- Additional elements supplied
- Position in lifecycle
- Development process
- Licence terms

None of the DTDs I looked at model the pattern satisfactorily. For example, the XML to reflect the fact that a list in a particular sentence is part of a paragraph.

In addition, the DTDs are in general large and complex – so customizing down to what we need is not a good idea, since:

- our work would look harder to understand
- we would get pigeon-holed as eg "part of DocBook"

The conclusion was that we need to develop our own grammar to represent the clause model. See http://www.oasis-open.org/archives/legalxml-econtracts/200307/msg00006.html for details.

Recommendation 1: The TC develop its own grammar for representing the clause model, rather than use an existing DTD.

Evaluation Process

The evaluation process focussed on a detailed examination of the ideas contained in the Harrop and Meyer proposals.

The process involved lengthy discussion with Peter Meyer, Andrew Squire and Justin Lipton about certain details.

The net result is that this recommendation reflects our shared view of the best way to do parts of the clause model. There are however certain points on which we differ, and where we do differ, these recommendations reflect my own views and my justifications for those views.

Terminology

In this document, the concepts "block" or "block-level" and "inline" or "inline-level" are used in much the same way as they are used in CSS http://www.w3.org/TR/REC-CSS1#formatting-model Extensible Stylesheet Language: http://www.w3.org/TR/xsl/ and XHTML http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410/xhtml-modularization.html#s_textmodule

A Word about DTDs and Schemas

The TC is yet to make a decision about which of DTD, W3C Schema, Relax NG or other grammars it will support, and which of these would be normative.

Recommendation 2: The TC publish its grammar as a W3C XML schema, in Relax NG, and (subject to recommendation 12) as a DTD.

For convenience only, this report expresses things using DTD syntax.

Recommendation 3: The TC publish the normative version of its grammar using either W3C XML schema, or Relax NG.

Architectural Guidelines

During the evaluation process, several architectural guidelines suggested themselves.

These guidelines help to ensure the resulting grammar handles issues which call for a judgement call in a consistent, sensible way.

The guidelines:

1. The grammar should be able to be expressed without error in each of the schema languages the TC adopts
2. Where an XML schema is used, an element should not be defined twice with the same name but a different content model

3. A given element is expected to always present as a block, or always as an inline. It must not be ambiguous.
4. Inline elements must not have block type descendants.
5. Encapsulation principle 1: Where markup is required in order to represent material which when presented will form a block, that markup should be encapsulated within a container which maps to the block.
6. Encapsulation principle 2: Where attributes are required to direct certain behaviour, these should live on the element most closely associated with the behaviour (eg a list container would contain the attributes which goven the behaviour of the list)
7. Provide containers where the container represents a concept a user can understand, and it would be useful for XSLT

Some of these guidelines warrant a bit of explanation.

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

For example if the TC chooses to publish the clause model as a W3C XML schema, a Relax NG schema, and also as a DTD, then the model must be such that a parser comes to the same conclusion as to the validity of the document irrespective of whether a W3C XML schema, a Relax NG schema, or a DTD is used.

As suggested by recommendation 2, my working hypothesis has been that the structural grammar will be published as a W3C XML schema, a Relax NG schema, and also as a DTD.

Note: namespace recommendations are outside the scope of this report. If we allow people to extend our work with unknown elements and attributes from unknown namespaces, this would not work well with DTDs (hence the "preferably").

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

In a data oriented XML schema, it makes sense to define say "party" globally, and then sub-class it locally where required.

This is a bad approach in a document-centric schema, since authors using an XML editor expect to be able to move say a party element to anywhere a party is allowed. If the content model for party is different depending on where in the document you wish to use it, re-use may or may not work, and users end up confused, which contravenes our ease of use requirement.

A given element is expected to always present as a block, or always as an inline

This greatly simplifies development, and is also a simple conceptual model for a user.

This principle ensures that our clause model can be rendered easily using CSS and XSL:FO, without the need to look-ahead, since there is a stable 1-1 mapping between our elements and those rendering models.

Note: this is a processing expectation. Organisations are free to force a different presentation (as for example is sometimes done with list items using CSS).

It also facilitates conversion to and from XHTML and WordML, since those grammars follow the same principle.

This principle does have a modest cost associated with it, in that artifacts which could occur inline or as a block (eg quotes, probably lists and images, and possibly equations and notes) need one element for their block form and another for their inline form (assuming we choose to make it possible to markup the inline form – hence the "probably" and "possibly").

Inline elements must not have block type descendants

See above for rationale.

The implication of this principle is that the content model for a block list must be different from the content model for an inline list (should we choose to support inline lists).

Encapsulation principle 1: Encapsulate markup within containers which map to blocks.

For similar reasons to the previous two principles, any collection of element content which is to be rendered as a block or blocks should be encapsulated within an element which maps to that block or blocks.

An example of the application of this principle is to the handling of block quotations (admittedly a requirement 11 item, but one which Peter Meyer and I have had to discuss). The contents of each of the blocks comprising the block quotation and any associated text rendered in those blocks are completely contained in an element which represents the block quotation. See page 22 below.

Encapsulation principle 2: Attributes should live on the element most closely associated with the relevant behaviour

Where attributes are convenient to assist renderers to present some artifact, those attributes should appear on an element directly associated with the artifact, not something further away.

For example, attributes for various properties related to list handling could be placed on a list container, the parent paragraph, or the first list item. This principle would dictate that the attributes go on the list container, not the first list item, nor the containing paragraph.

Useful Containers which make sense to users

This principle is closely related to the previous one.

Where a container represents a concept which makes sense to a user (eg a list), then other things being equal, that container should be included in the grammar since it is so useful to XSLT processing.

Recommendation 4: The TC adopt each of the Architectural Guidelines specified herein.

Requirement 11

Requirement 11 is outside the scope of the "basic" clause model, and not specifically addressed by it.

Some thoughts regarding some of these matters are nevertheless captured later in this document (see page 22 below).

For the convenience of readers, the contents of requirement 11 are reproduced here:

(a) the numbering of objects and the means by which numbers are specified in the markup;

(b) inclusion of distinct content such as quotations, examples and explanatory notes that are part of the contract document;

(c) inclusion of tables with table cells that may contain other clause model structures;

(d) images;

(e) equations (mathematical formulae);

(f) footnotes / end notes;

(g) internal cross references;

(h) citations to other works;

(i) defined terms;

(j) references to parties;.

Recommended Solution

Basic Description

The model encapsulates each clause within an element called <Item> (see DTD below).

The model uses the same name at each level of nesting (ie it is hierarchical and recursive). An important simplifying feature is that only one element is available at each level in the main hierarchy. Inside each level, subsidiary list hierchies are available, but these are terminal in the sense that you can't resume the main hierarchy from inside a List.

Recommendation 5: The TC represent clause structures using a recursive hierarchy, with one and only one container at each level (but permitting a List hierarchy inside each level in a manner which does not allow the main recursive hierarchy to be resumed from inside a List).

An Item has an optional Title.

It contains zero or more paragraphs (<Para>), which represent the body of a clause, followed by zero or more sub Items.

<Para> is made up of one or more TextBlocks, and other block-level content.

Block-level content would include block lists, tables, images, block quotations etc. With the exception of block lists this other block level content is a matter for the requirement 11 stage.

Each <TextBlock> is understood to be presented on a new line, and contains the actual inline content.

The two features:

1. <Para> being composed of a sequence of one or more block elements
2. an additional mixed content element representing the blocks of text themselves (<TextBlock>)

makes this a so-called 'True Paragraph' model: http://www.griffinbrown.co.uk/griffin-brown-paragraph-article.pdf

Similar to <Para>, the content model for <Title> is block level content, not just straight PCDATA, since this assists people who have multi-line headings and wish to control where the line breaks occur (ie using <TextBlock>. Obviously you ordinarily wouldn't put tables in a Title, but you could.

Examples

Example 1 – Simple Clause

1. Exercise of Discretions
Any discretions a party has under or in connection with this agreement must be exercised in a prompt and reasonable manner.

```
<Item ID="discretions">
<Num>1.</Num>
<Title><TextBlock>Exercise of Discretions</TextBlock></Title>
<Para>

        <TextBlock>Any discretions a party has under or in connection with
        this agreement must be exercised in a prompt and reasonable
        manner.</TextBlock>
</Para>
</Item>
```

The <Num> element is a placeholder element. It will be fully defined as part of Requirement 11.

Example 2 – Nested Clauses

2. Intellectual Property
2.1 **Ownership**
The Parties intend that all Intellectual Property Rights in New Contract Material (***Contract IPR***) will be owned by Customer.

2.2 **Assignment**
Subject to payment of the Contract Fees, Supplier assigns to Customer all of its existing and future Intellectual Property Rights in New Contract Material. Supplier must on request do all things necessary to give effect to this assignment.

```
<Item ID="ip">

        <Num>2.</Num><Title><TextBlock>Intellectual
        Property</TextBlock></Title>
<Item ID="ip-own">

        <Num>2.1</Num><Title><TextBlock>Ownership</TextBlock>
        </Title>

        <Para>
```

```
                <TextBlock>The Parties intend that all Intellectual
            Property Rights in New Contract Material (Contract IPR) will be
            owned by Customer.</TextBlock>

        </Para>
</Item>

<Item ID="ip-ass">
        <Num>2.2</Num><Title><TextBlock>Assignment</TextBlock>
        </Title>

        <Para>

            <TextBlock>Subject to payment of the Contract Fees, Supplier
        assigns to Customer all of its existing and future Intellectual Property
        Rights in New Contract Material.   Supplier must on request do all
        things necessary to give effect to this assignment. </TextBlock>
            </Para>
</Item>
</Item>
```

Example 3 – Lists

3. Termination

Without limiting any other rights it may have, a party may immediately terminate this agreement by giving notice to the other party if the other party:

(a) breaches a material provision of this agreement and fails to remedy the breach within 30 days after receiving a notice requiring it to remedy the breach;

(b) is unable to pay its debts as they become due; or

(c) stops carrying on business.

Note that this example does not contain sub-lists, but they are handled by the model (see DTD below).

```
<Item ID="termination">

        <Num>3.</Num>

        <Title><TextBlock>Termination </Title>

        <Para>
```

```
<TextBlock>Without limiting any other rights it may have, a party
may immediately terminate this agreement by giving notice to the
other party if the other party: </TextBlock>

<List>

        <ListItem ID="term-a">

                <Num>(a)</Num>

                <Para>

                        <TextBlock>breaches a material provision of this
agreement and fails to remedy the breach within 30 days after
receiving a notice requiring it to remedy the breach;</TextBlock>

                </Para>

        </ListItem>

        <ListItem ID="term-b">

                <Num>(b)</Num>

                <Para>

                        <TextBlock>is unable to pay its debts as they
become due; or

                        </TextBlock>

                </Para>

        </ListItem>

        <ListItem ID="term-c">

                <Num>(c)</Num>

                <Para>

                        <TextBlock>stops carrying on business.

                        </TextBlock>
```

```
                    </Para>

                </ListItem>

            </List>

            </Para>
</Item>
```

The List is contained in the relevant <Para>, at the block level.

I looked at using Item as the content model for a List. This was not possible, because of the problems set out in Attachment 2. It has been suggested that these problems could be overcome by having a special definition for Item when it is in the context of a List. However, that would contravene Architectural Guidelines 2, and if Recommendation 2 (support DTDs) is accepted, it would also contravene ArchitecturalGuideline 1.

There was also some discussion as to whether the List container could be dispensed with. I have kept it, for the reasons given in Attachment 3.

Recommendation 6: The concept of a List be included in the grammar.

DTD

The grammar I suggest is the following (expressed for convenience as a DTD):

```
    <!ELEMENT Item (Num?, Title?, Para*, Item*)>
    <!ATTLIST Item ID ID #REQUIRED>

    <!ELEMENT Num (#PCDATA)><!-- Place holder, pending requirement 11 -->

    <!ELEMENT Title (TextBlock)+>

    <!ENTITY % BlockLevelContent "TextBlock | List " >
    <!ELEMENT Para ( %BlockLevelContent; )+>

    <!ELEMENT List (ListItem)+>
    <!ELEMENT ListItem (Num?, Title?, Para+)>
    <!ATTLIST ListItem ID ID #REQUIRED>
```

```
<!ENTITY % InlineLevelContent "#PCDATA " >
<!ELEMENT TextBlock (%InlineLevelContent; )*>
```

Recommendation 7: The TC adopt the suggested grammar (using its preferred schema language).

Loose or Tight Model

It has been observed that the so-called "loose" model:

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

allows an Item to have both Para and Item children at the same time.

For example:

```
<Item>
    <Para>
        <TextBlock>Some text in this clause body</TextBlock>
    </Para>
    <Item>
        <Para>
            <TextBlock>Some text in a sub-clause.</TextBlock>
        </Para>
    </Item>
</Item>
```

It has been observed that this structure is rarely required in a contract, if at all.

For this reason, one might consider the so-called "tight" model:

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

My recommendation is to adopt the loose model.

The reasons are as follows:

1. However rare, people may wish to represent content which requires the flexibility of the loose model (particularly if the clause model is used outside the contract domain).
2. The loose model is useful in an XML editor which does continuous validation, since it allows you to re-structure content by temporarily having both a <Para> and a sub-<Item> (which would otherwise be invalid).

I have considered whether we should publish both models as part of our specification, recommending the tight model for use within an organisation, and the loose model for exchange purposes.

I recommend against this. Assuming use of the standard became widespread, an organisation would for all practical purposes have to support the loose model. This means that of the two, the loose model would become the defacto standard. Better to avoid the confusion, and make it the standard to start with. (Of course, particular organisations could still use the tight model internally. Put another way, if an organisation chooses to implement some subset of the standard internally, that is their business).

Recommendation 8: The TC should not officially sanction the so-called "tight" model as an alternative to the "loose" model recommended here.

Element Names

I have followed the naming conventions set out in Universal Business Language (UBL) Naming and Design Rules, Working Draft 22, of 18 February 2003: http://www.oasis-open.org/apps/org/workgroup/ubl/ubl-ndrsc/download.php/1380/wd-ublndrsc-ndrdoc-22.doc

Amongst other things, these call for:

> Names for XML constructs *must* use camel-case capitalization, such that each internal word in the name begins with an initial capital followed by lowercase letters (example: **AmountContentType**). All XML constructs other than attributes *must* use upper camel-case, with the first word initial-capitalized, while attributes *must* use lower camel-case, with the first word all in lowercase (example: **unitCode**), with the exception of attribute names consisting of one of the allowed acronyms, abbreviations, or truncations given in R3.

An exception is that I have truncated "Paragraph" to "Para". The intent is that it suggests a grammatical paragraph, but no more. It is not exactly a grammatical paragraph, since list items can contain Para.

Recommendation 9: Consistent with UBL, names for XML constructs *must* use camel-case capitalization, such that each internal word in the name begins with an initial capital followed by lowercase letters (example: **TextBlock**). All XML constructs other than attributes *must* use upper camel-case, with the first word initial-capitalized, while attributes *must* use lower camel-case, with the first word all in lowercase (example: **unitCode**), with the exception of ID.

I have called the main element "Item", not "Topic", because in some DTDs (eg DITA), Topic has a compulsory title. Since title is optional in the documents we are seeking to model, it was thought better to use something other than Topic.

Summary of Fit to Requirements

Requirement 1

1. The clause model adopted for the markup of contract terms must be able to markup the core structures found in contract and other legal & business documents similar to those described in Attachment 1.

Meets – see Attachment 1 to this document.

Requirement 2

2. The clause model markup must represent the structured hierarchy of the content so that it can be rendered in a form to accurately convey the structure and meaning of the text to readers. In the example in Attachment 1, object 1.1, the markup must ensure that the words "but excluding violet," are part of item (e). The words "from which all colours can be derived." are part of the introductory grammatical paragraph so that one element container must include everything beginning with "Here is .." to "derived".

Meets – see Attachment 1 to this document.

Requirement 3

3. The clause model must be able to represent the "benchmark" contracts which are submitted by members of the TC and accepted by the TC as in scope.

The TC is yet to specify its benchmark contracts.

Requirement 4

4. The clause model must define clause objects so they can be addressed and manipulated by software systems as self contained objects.

Meets – the "clause objects" are represented as <Item>, an <Item> encapsulates any sub-clauses (ie it is self-contained), and an <Item> must have an ID (ie it can be addressed).

Requirement 5

5. The clause model must provide for self contained markup of content so that if the parties desire to use the XML text file as the contract document, it is not necessary to use any software apart from that needed to display a text file to determine the terms of the contract. For example, other software should not be required to interpret numbering of objects or cross reference targets.

Meets (except for the numbering and cross reference aspects, which are currently out of scope and to be addressed as part of requirement 11).

Requirement 6

6. The terminology or element names must not imply any particular form of citation so that the terms can be included in different types of document by persons from jurisdictions with different citation traditions. Consequently, the DTD/Schema must not use the following terms in element markup: [list of terms omitted]

Meets.

Requirement 7

7. The clause model must permit the markup of contract terms without inclusion of any legal semantic markup or annotation. In other words, the XML markup must not be relevant to the interpretation of the document once it is rendered in a published form unless the parties specifically agree otherwise.

Meets.

Requirement 8

8. The clause model must be as simple as practicable to facilitate user training, support and application development.

Meets. The clause model is simple. As explained above, a key feature is its single main hierarchy.

The solution does contain some extra elements (eg <TextBlock>) in order to address some edge cases and in order to satisfy the architectural guidelines and thus simplify application development. It is not expected that these elements will prove burdensome for authors.

> To make it easy for users, semantic distinctions between generic objects will only be introduced where the benefits are clearly demonstrable.

Meets. The solution does retain the concept of a list, with "ListItem" as its content model. The reason for doing this is set out in attachment 2.

Requirement 9

9. The model must allow document authors to re-use content in different levels of the hierarchy, without having to change the names of the elements in order for the document to remain valid.

Meets. The model is recursive and its main element is called "Item". An item can be re-used at any level.

Note however that an item cannot be copied into a list, or vice versa. It is anticipated that if user demand justifies it, editing tools will contain macros or wizards to assist with this. See attachment 2 for further discussion.

Requirement 10

10. The model must allow clauses or other content to be incorporated into a document by reference.

Incorporation by reference is defered to requirement 11 stage, pending further discussion.

Recommendation 10: Incorporation of Items by reference be defered to requirement 11 stage, pending further discussion.

Dr Leff's suggestion

In an email message quoted in http://www.oasis-open.org/archives/legalxml-econtracts/200306/msg00014.html Dr Leff was concerned that:

```
     The clause model must permit the markup of contract terms without
inclusion of the XML tags whose purpose is the display of the document.
```

In approving the requirements document, the TC took this concern on board. I believe the clause model suggested here satisfies this concern.

Requirement 11 and other outstanding issues

This section captures preliminary discussions on these matters for the information of readers, since it informs some of the design.

(b)(i) quotations – block

Legal style is often to include brief quotations of say fewer than fifty words inline in the regular text (ie in the <TextBlock>). Longer quotations are generally presented as blocks; quotation marks may or may not be used.

Certain requirements relating to block quotations have been identified:

1. A citation to the source of a block quotation will often be placed in brackets immediately below the quotation: http://www.judiciary.state.nj.us/style.htm
2. Where the block quotation comes at the end of a sentence, the period marking the end of the sentence could appear on the same line as the last paragraph of the quotation. Where the sentence continues after the block quotation, it should continue on a new line.
3. The block quotation may be composed of multiple paragraphs
4. Generally, quotation marks aren't used for block quotes. However, we need to

document expectations: if the author wants them, should they insert them, or is it expected that presentation software would do this?

To meet these requirements, a content model similar to the following is suggested:

```
<!ELEMENT BlockQuotation (Para*, Citation?, TrailingText?)>
<!ELEMENT TrailingText (#PCDATA)>
```

The TrailingText element could be used to mark up a period (fullstop) when the sentence ends with the block quotation. If the sentence continues, it would continue in a sibling TextBlock.

(i) defined terms

There are three things one could potentially mark up:

- the text of the definition of a term (eg "Confidential Information")
- the meaning of a defined term
- the use of a defined term.

It is proposed that the clause model will mark up the text of the definition of a term, and any instance of the use of that term, but not the meaning of the defined term.

The reasons for this stance:

- where the definition begins and ends may be a legal issue
- the meaning of the defined term could cross block boundaries, and therefore complicate the content models unnecessarily if this were marked up.

For these reasons, it is proposed that the meaning of defined terms be captured in the non-structural layer, if anywhere. Non-structural markup is to be developed separately.

Inline lists

In addition to the issues already collected in requirement 11, we need to decide whether to support inline lists, and if so, how.

Peter Meyer has suggested (private correspondence) that the TC consider the following questions:

(a) Are inline lists common in modern documents or more of a legacy structure?

(b) Would authors sometimes want to swap between inline to new line rendering (either direction) of lists?

(c) Would authors reliably markup inline lists or would they sometimes just type in the text and forget about the markup?

(d) Would they ever include multiple levels of lists inline (list >sublist)?

(e) Would an author ever include both an inline list and a new line list in the same paragraph (Para element)?

(f) Is there any reason why a special markup structure is needed for inline lists? In other words, is there any material difference between the two list layouts?

(g) Is the use of inline lists better treated as a matter of house style or author preference? For example, is it necessary that an author can mix both layouts in one document?

Recommendation 11: Whether to markup inline lists, and how, to be considered as part of requirement 11.

Table of contents

To be considered as part of requirement 11.

Namespace handling and extension mechanism

To be considered as part of requirement 11.

Recommendation 12: Namespace handling and extension mechanism be considered at the requirement 11 stage.

Conclusion

The clause model recommended here is a solid foundation for a complete structural DTD:

- It is based on sound, articulated, architectural principles
- Reflects real document structure
- It handles the hard cases
- Neither too loose nor too tight
- Supportable in a wide range of popular editing environments and other existing tools
- Easy for authors
- Easy for developers, including to style and render

Attachment 1 – Clause Structure Example

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Item [

	<!ELEMENT Item (Num?, Title?, Para*, Item*)>
	<!ATTLIST Item ID ID #REQUIRED>

	<!ELEMENT Num (#PCDATA)><!-- Place holder, pending requirement 11 -->

	<!ELEMENT Title (TextBlock)+>

	<!ENTITY % BlockLevelContent "TextBlock | List " >
	<!ELEMENT Para ( %BlockLevelContent; )+>

	<!ELEMENT List (ListItem)+>
	<!ELEMENT ListItem (Num?, Title?, Para+)>
	<!ATTLIST ListItem ID ID #REQUIRED>

	<!ENTITY % InlineLevelContent "#PCDATA " >
	<!ELEMENT TextBlock (%InlineLevelContent; )*>
]>


<Item  ID="c1">
	<Num>1.</Num>
	<Title><TextBlock>Provisions about the specification of colours in contracts</TextBlock></Title>
	<Item  ID="c11">
		<Num>1.1</Num>
		<Title><TextBlock>Spectrum colours</TextBlock></Title>
		<Para>
			<TextBlock>Here is a contrived, complex List structure using the spectrum
colours and one or two others:</TextBlock>
			<List>
				<ListItem  ID="c111">
					<Num>(a)</Num>
					<Para>
						<TextBlock>red,</TextBlock>
					</Para>
				</ListItem>
				<ListItem  ID="c112">
					<Num>(b)</Num>
					<Para>
						<TextBlock>orange,</TextBlock>
					</Para>
				</ListItem>
				<ListItem  ID="c113">
					<Num>(c)</Num>
					<Para>
						<TextBlock>yellow,</TextBlock>
					</Para>
				</ListItem>
				<ListItem  ID="c114">
					<Num>(d)</Num>
					<Para>
						<TextBlock>green,</TextBlock>
					</Para>
				</ListItem>
				<ListItem  ID="c115">
					<Num>(e)</Num>
					<Para>
						<TextBlock>blue, including:</TextBlock><List>
						<ListItem   ID="c1151">
							<Num>(i)</Num>
							<Para>
```

```xml
                                <TextBlock>pale blue,</TextBlock>
                            </Para>
                        </ListItem>
                        <ListItem  ID="c1152">
                            <Num>(ii)</Num>
                            <Para>
                                <TextBlock>dark blue,</TextBlock>
                            </Para>
                        </ListItem></List>
                        <TextBlock>but excluding violet,</TextBlock>
                    </Para>
                </ListItem>
                <ListItem  ID="c116">
                    <Num>(f)</Num>
                    <Para>
                        <TextBlock>indigo, and</TextBlock>
                    </Para>
                </ListItem>
                <ListItem   ID="c117">
                    <Num>(g)</Num>
                    <Para>
                        <TextBlock>violet,</TextBlock>
                    </Para>
                </ListItem>
            </List>
            <TextBlock>from which all colours can be derived.</TextBlock>
            <TextBlock></TextBlock>
        </Para>
    </Item>
    <Item  ID="c12">
        <Num>1.2</Num>
        <Title><TextBlock>CMYK colours</TextBlock></Title>
        <Para>
            <TextBlock>CMYK colours (cyan, magenta, yellow and black) are normally specified for inputs to colour
printing processes.</TextBlock>
        </Para>
    </Item>
    <Item  ID="c13">
        <Num>1.3</Num>
        <Title><TextBlock>RGB colours</TextBlock></Title>
        <Item  ID="c131">
            <Num>1.3.1</Num>
            <Para>
                <TextBlock>RGB colour (red, green, blue) specifications are used for computer screen
displays.</TextBlock>
            </Para>
        </Item>
        <Item   ID="c132">
            <Num>1.3.2</Num>
            <Para>
                <TextBlock>Using only these 3 colours, you can specify any colour.</TextBlock>
            </Para>
        </Item>
        <Item  ID="c133">
            <Num>1.3.3</Num>
            <Para>
                <TextBlock>The number of colours you can specify
depends on the colour depth available. For example:</TextBlock>
                <List>
                    <ListItem  ID="c1331">
                        <Num>(a)</Num>
                        <Para>
                            <TextBlock>8 bit colour can render 256 colours;</TextBlock>
                        </Para>
                    </ListItem>
                    <ListItem  ID="c1332">
```

```
                        <Num>(b)</Num>
                        <Para>
                            <TextBlock>16 bit colour can render 65, 536 colours.</TextBlock>
                        </Para>
                    </ListItem>
                </List>
            </Para>
        </Item>
    </Item>
    <Item  ID="c14">
        <Num>1.4</Num>
        <Title><TextBlock>Using black and white</TextBlock></Title>
        <Item  ID="c141">
            <Num>1.4.1</Num>
            <Title><TextBlock>Greyscale</TextBlock></Title>
            <Para>
                <TextBlock>The number of greys depends on the available colour depth, as for other
colours.</TextBlock>
            </Para>
        </Item>
        <Item  ID="c142">
            <Num>1.4.2</Num>
            <Title><TextBlock>Black and white</TextBlock></Title>
            <Para>
                <TextBlock>This is really called monochrome. You can specify
either:</TextBlock>
                <List>
                    <ListItem  ID="c1421">
                        <Num>&#x2022;</Num>
                        <Para>
                            <TextBlock>black, or</TextBlock>
                        </Para>
                    </ListItem>
                    <ListItem  ID="c1422">
                        <Num>&#x2022;</Num>
                        <Para>
                            <TextBlock>white.</TextBlock>
                        </Para>
                    </ListItem>
                </List>
            </Para>
        </Item>
    </Item>
    <Item  ID="c2">
        <Num>2.</Num>
        <Title><TextBlock>Colour profiles</TextBlock></Title>
        <Para>
            <TextBlock>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.</TextBlock>

        </Para>
    </Item>
</Item>
```

Attachment 2 – Block List Model

We considered using Item:

<!ELEMENT Item (Num?, Title?, Para*, Item*)>

as the content model for a block <List>.

However, this introduced the problems identified below.

In view of these problems, Recommendation 7 includes the following content model:

<!ELEMENT List (ListItem)+>

<!ELEMENT ListItem (Num?, Title?, Para+)>

ie a distinct ListItem element.

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.

Problem 1 – nested lists can omit container

The first problem which would arise if Item+ were the content model for a <List> is that a list can be rendered as:

```
<Para><TextBlock>This is the introduction to my List:</TextBlock>
    <List>
    <Item><Num>(a)</Num><Para><TextBlock>First
Item;</TextBlock></Para></Item>
    <Item><Num>(b)</Num>
            <!-- No List element -->
          <Item><Num>(i)</Num><Para><TextBlock>First sub
Item;</TextBlock></Para></Item>
          <Item><Num>(ii)</Num><Para><TextBlock>Second sub
Item;</TextBlock></Para></Item>
    </Item>
    <Item><Num>(c)</Num><Para><TextBlock>Third Item with a sub
Item:</TextBlock>
             <List>
             <Item><Num>(i)</Num><Para><TextBlock>First sub
Item;</TextBlock></Para></Item>
             <Item><Num>(ii)</Num><Para><TextBlock>Second sub
Item;</TextBlock></Para></Item>
```

```
                </List>
               </Para>
    </Item>
    </List>
</Para>
```

In this example, the sublists are marked up in two ways, as sub-Items (ie like sub-clauses), then as a List, respectively. This is obviously undesirable.

The effect of doing them as sub-Items is that they becomes siblings, not children, of any Para in the top level List:

```
<Para><TextBlock>This is the introduction to my List:</TextBlock>
    <List>
    <Item><Num>(a)</Num><Para><TextBlock>First
Item;</TextBlock></Para></Item>
    <Item><Num>(b)</Num><Para><TextBlock>Second Item;</TextBlock></Para>
                <!-- Para precedes Items, no List element -->
           <Item><Num>(i)</Num><Para><TextBlock>First sub
Item;</TextBlock></Para></Item>
           <Item><Num>(ii)</Num><Para><TextBlock>Second sub
Item;</TextBlock></Para></Item>
    </Item>
    </List>
</Para>
```

Problem 2 - "tight model" doesn't help

We considered whether the so-called "tight model" :

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

would overcome these problems.

It doesn't, since the following alternatives are still possible:

```
<Para><TextBlock>This is the introduction to my List:</TextBlock>
    <List>
    <Item><Num>(a)</Num>
        <Para>
```

```
                <List>
                 <Item><Num>(i)</Num><Para><TextBlock>First sub
Item;</TextBlock></Para></Item>
                 <Item><Num>(ii)</Num><Para><TextBlock>Second sub
Item;</TextBlock></Para></Item>
                </List>
        </Para>
    </Item>
    <Item><Num>(b)</Num>
        <!-- Para List -->
                 <Item><Num>(i)</Num><Para><TextBlock>First sub
Item;</TextBlock></Para></Item>
                 <Item><Num>(ii)</Num><Para><TextBlock>Second sub
Item;</TextBlock></Para></Item>
    </Item>
    </List>
</Para>
```

Attachment 3 – The List element

The question arose as to whether the List container could be dispensed with, so that instead you'd just have list items instead of the container.

Here is a summary of the reasons for keeping the List container:

1. People are general familiar with the concept of a <List>; its only difficult to pin down in certain edge cases, and in those cases people make choices pragmatically:

- are these items in a paragraph?
- how do you want to format this thing?
- do you want following text?

2. The List container is convenient and natural for specifying numbering properties for list entries. To put them anywhere else, for example, on the parent container eg <Para>, contravenes Architectural Guideline 6, and doesn't work if you have two different lists in the <Para>. Putting it on the first list entry is even worse.)

3. The List container is convenient for manipulation using the structural view in an XML editor since you can select it and move it somewhere else, or delete the entire list. You could also view the entire list in a separate panel.

4. The List container will be useful for various vertical applications some of which we can't predict. Amongst these, is document automation. In document automation, you need a list container if you want to handle lists properly if some list entries are intended to possibly be omitted (eg1 correct placement of "; and" before the last list entry. eg2 behaviour if there is a single list entry, or none).

5. XSLT processing for whatever you want to do is assisted by (1) having a container you can match on and (2) pure content models, both of which <List> provide. For example, to check you are processing the first list entry, you can do position()=1 rather than something like count(preceding-sibling::ListItem)=0 which is necessary if you could have an <ListItem> straight after <TextBlock>.

Note that the <List> container is not essential to working out numbering levels:

```
   list level = count(ancestor::Para) -1
   article level = count(ancestor::ListItem) - list level
```

though the intent of count(ancestor::List) is much more self-evident than count (ancestor::Para). This is another illustration of point 5 above.