OASIS LegalXML

eContracts Technical Committee

Structural markup – Basic clause model
Proposal and specification

Author: Peter Meyer, Elkera Pty Limited, pmeyer@elkera.com.au

Draft 1.0

11 November 2003

Contents

Clause model proposal and specification

Submitted by: Peter Meyer, Elkera Pty Limited, (pmeyer@elkera.com.au)

Draft 1.0
11 November, 2003.

1. Introduction

1.1 Purpose of this document

This document is a proposal to the Technical Committee for a "basic" clause model as the first part of its eContracts standards development.

It is provided as a comprehensive alternative to the report provided by Jason Harrop on 14 October 2003 and the alternative submitted by Peter Meyer on 15 October 2003.

The clause model requirements document set out a staged approach to development of a structural markup component of the proposed standard, 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.

1.2 Contributions to this proposal

Some ideas and content, particularly related to the alternatives presented under recommendations, were contributed by Jason Harrop of SpeedLegal Pty Limited, jharrop@speedlegal.com. However, this document does not necessarily reflect Jason Harrop's views and is not endorsed by him.

The document also includes contributions from Andrew Squire and Daniel Noll of Elkera Pty Limited.

1.3 Related documents

The following documents may be directly relevant to the interpretation of this document:

- Clause model requirements located at http://lists.oasis-open.org/archives/legalxml-econtracts/200305/msg00027.html
- TC member scenarios listed at http://www.oasis-open.org/apps/org/workgroup/legalxml-econtracts/documents.php.

1.4 Form of the document

The document is structured as a specification for the clause model with additional components to reflect its evolutionary nature during development. It will contain general explanatory content plus a description of each content model starting with the highest level element.

The document is written with the expectation that it will be read by a wide range of persons with varying levels of experience with XML applications. It assumes that the reader at least knows that XML is a standard that provides a syntax for defining document markup languages. In places it is necessary to understand DTD syntax and design concepts.

The description of each content model section may include:

(a) an explanation of its purpose;

(b) examples of its possible use;

(c) particular design issues that have been identified;

(d) major alternative options considered;

(e) reasons for the choice between available options;

(f) policy or other questions that require a decision from the Technical Committee or input from other sources;

(g) a record of Technical Committee decisions on questions raised.

During development, the updated versions of the specification will provide an ongoing record of the state of the proposed model. As further stages of the clause model requirements are addressed, it is proposed they should be included in this specification. At the end of the process, the document should easily be converted into a final specification for the model by removal of unwanted historical material.

2. Strategic background

2.1 Business problems addressed by the model

The business problems to be addressed were discussed in the clause model requirements document. They are summarised as follows:

(a) It is necessary to provide for the XML markup of contract documents to support the needs of users broadly described in the TC member scenarios. These needs cover a vast spectrum of possible requirements for XML markup of contract documents.

(b) The model should meet the needs of the widest range of users in a single standard.

(c) The model should facilitate the development by system vendors of innovative, low cost tools to service the identified needs. It is perceived that currently there are major impediments to the adoption of XML markup for contract documents by law firms and other enterprises. These include the high costs of development around proprietary markup models and the change management issues involved in moving from the format based approach to authoring of word processing software to a structured authoring environment using XML.

(d) The model should facilitate the adoption of the standards by user enterprises. In particular, it is recognised that law firms and other user enterprises create many kinds of documents in addition to contracts. Many of the needs identified in the TC member scenarios are applicable to non contract documents as they are to contract documents. Accordingly, the standard should not compel user enterprises to adopt multiple, XML markup standards to cover their business documentation needs where these overlap with their needs for contract documents.

Unless the standard addresses these business problems it is highly likely that the standard will either fail to gain any acceptance or, at best, it will be used in only a few niche markets. The aim is to develop a standard that benefits the legal community as a whole and those that depend on its services.

2.2 Document types

The clause model requirements address contracts and other legal and business documents. These documents mainly contain hierarchically structured, narrative text content and are prepared in virtually any legal and business context.

The clause model is not intended to be applicable to non narrative documents such as spread sheets, financial and database reports, or highly graphically oriented documents with complex page layout requirements.

2.3 The user community

A recurring issue during development of the clause model requirements is the extent to which XML markup will or may be used for contract documents in various parts of the legal and business community. This issue can be encapsulated in these questions:

(a) Is it envisaged that the XML markup will be used mainly by specialist content authors who prepare contract documents as standard precedents or templates in larger firms and by persons who maintain documents used for, say, online and electronic transactions?

(b) Is it envisaged that, in addition to group (a), lawyers in everyday practice will also use XML markup during the preparation of contract documents in place of their existing word processor based authoring environments?

Scenarios submitted by TC members envisage that benefits from the use of XML markup for contracts preparation could accrue to both groups.

Certainly the standard must address group (a). It is unclear whether and how quickly XML markup for document creation will be used by group (b) document authors. The capacity to create documents using XML markup is increasingly being built into mainstream products such as Microsoft Word in Office 11 so there is room for optimism.

There is no clear separation between groups (a) and (b). Rather, there is a continuum between two extremes. It is expected that if the standard meets the needs of authors in group (b) it will maximise its chance of gaining widespread developer support and use.

2.4 Contracts and other legal and business documents

The clause model requirements explain that the preparation of contract documents overlaps extensively with the preparation of many other kinds of legal and business documents in law firms and other enterprises.

It is also shown that the basic structural pattern of clause and content of most other legal and business documents is fundamentally the same as for contract documents.

It is proposed that the clause model must be designed for use with other legal and business documents described in the clause model requirements. This will allow enterprises and authors to use common tools across the widest range of documents and support the business objectives set out in topic 2.1.

2.5 XML related standards and applications

If the standard is successful and the expected business objectives are achieved, a very substantial infrastructure of supporting applications will be built around the standard. User organisations will build up large collections of documents that conform to the standard. Once a substantial user community is in place it will be extremely difficult to alter the standard in a way that may invalidate existing data sets or force changes to applications. It is likely that the standard will be displaced only by major technological change that displaces XML for document markup.

It is hoped and expected that the standard will endure for many years. A lifespan of at least 10 years is realistic and it is conceivable that the lifetime of the standard could extend to some multiples of that time. This really depends on how XML evolves and whether completely new approaches might emerge to replace XML. A great deal of change has occurred with XML related standards and the development of complimentary processing applications over the past 5 years. During that period there have been continuous releases of new and improved tools for processing XML data and building XML related applications.

It is assumed that this evolution will continue over the life of the standard. Many of the limitations of today's XML processing tools are likely to disappear in a relatively short period. The diversity of tools and the approaches they take to solving problems will increase.

Care is required before the design is specifically adapted to work with particular current XML processing technologies. If it does, there is a high risk that, in the future as XML processing applications evolve and become more powerful, particular design feature will be seen as inconvenient.

3. Structural markup

3.1 An ordered hierarchy

The content of contracts, along with most other narrative text documents, is based on the arrangement of concepts into an ordered hierarchy. For example, a journal article or report may contain an introduction and then multiple chapters or sections. Each chapter may contain sub chapters, then paragraphs and then lists. Contracts may contain similar hierarchical structures, although the common citation names given to some of those structures will vary between different user communities and jurisdictions with different traditions.

The structural XML markup describes this ordered hierarchy of document components. It defines the boundaries of each component, its hierarchical relationship to other components and provides information sufficient for human authors and computer systems to determine the generic (non legal) function of each component.

Structural markup seeks to avoid or minimise the capture of purely presentational information. Part of the benefit of structural markup is to allow documents to be displayed appropriately in different publishing environments and by different users. It also enables enterprises to globally update house publishing styles without the need to edit individual document instances.

3.2 The purpose of structural markup

Structural markup allows system designers to define a small, finite number of markup elements so that processing applications can work with a manageable and stable data set. For instance, if the base markup of a contract were to define its legal operation so that the element markup for a payment clause is different to the markup of a warranty clause, it would be impossible to write an application to process a document with more than a few standard clauses.

Structural markup can reduce or avoid dependencies between the document content and proprietary processing systems used to process and publish the documents. It minimises the risks associated with the commercial and technological obsolescence of software applications. Document owners may change applications as commercial and technological conditions evolve without having to undertake expensive or impracticable data conversion to new formats.

If structural markup is to be effective, it must capture the ordered hierarchy of document components in sufficient detail that a processing application has sufficient information to know how to process the component to meet the needs of the application users.

The benefits of structural markup are maximised if the schema is itself not a proprietary language, controlled by one system vendor. An open standard schema is accessible to anyone and does not require users to translate data to new schema languages when changing software.

3.3 The use of structural markup in the standard

The clause model requirements, were approved by the Technical Committee on 18 June 2003. Those requirements propose development of a structural model for the XML markup of contract documents. The clause model described in this document is the foundation of the proposed structural markup model.

The purpose of the clause model structural markup is to provide a platform for the XML markup of contract documents that will meet the widest range of needs of persons preparing and using contract documents.

In the context of the proposed eContracts standard, structural markup will enable XML processing systems to perform these sorts of operations:

(a) automatically translate documents to other XML dialects or to completely different markup languages;

(b) automatically translate documents into presentational formats such as RTF, HTML/XHTML, PDF, SVG etc;

(c) programmatically locate and manipulate discrete chunks of content in document automation systems for document assembly, content re-use and content sharing;

(d) identify document components that may be addressed in hypertext linking systems.

The structural markup alone will not meet all the needs described in the scenarios submitted by Technical Committee members as a precursor to requirements development.

The function of the basic clause model is to provide a common, generic framework on which additional information (metadata) may be attached. The metadata may, for example, include legal semantic information or transactional information relevant to contract documents and identifiable components. That additional information may assist with some of the operations described above as well as support automated transactions involving contracts.

This approach will allow additional information to be added to the structural markup at the option of those who require particular functionality, without burdening those who do not require it.

We can think of the structural clause model as the base of a pyramid. Users with the most common needs are at the base and users with more specialised needs are at higher levels. At the base level, users don't need to add much, if any, extra information to the markup to meet all their basic document production needs. As they move to functions at higher levels of the pyramid, they may need to add more and more information to the markup or associate information from other sources with the markup.

4. XML and the contract

An XML document is not a useful document for human readability. As an electronic text file, it lacks normal presentational information that is essential to ease of readability. The presence of XML markup with the linguistic content is highly inconvenient to readers.

For use by humans, XML documents must be transformed into a presentation format or directly rendered into a presentation display with the aid of a style sheet and conforming software.

It is assumed that the parties to contracts would rarely, if ever, signify an XML document alone as "the contract document". However, it may be possible in some electronic transactions that this could occur.

It is assumed that contract documents marked up using the clause model will be translated or rendered into a presentation format chosen by the parties. In the normal situation, that presentation document will be the document that is treated as the contract document for evidentiary purposes.

The basic clause model makes no assumptions about the presentation format that may be used by the parties or the tools that may be used for translation or display.

Requirement 5 of the clause model requirements demands that the XML document must be self contained so that if the parties do signify it as the document with evidentiary status, it does not necessarily require software, apart from a text editor, to determine the terms recorded in the XML document.

5. Requirements for the “basic” clause model

The basic clause model is intended to provide only the foundation for meeting the needs described in topic 3.2. The specific requirements defined in the clause model requirements document are set out in Attachment A. They are summarised as follows:

1. It must markup core structures similar to Attachment 1 to clause model requirements.
2. It must represent the structured hierarchy.
3. It must be able to markup benchmark contracts.
4. It must define clause objects for computer manipulation.
5. The XML file for a contract document must not require software to interpret.
6. Element names must not include listed citation terminology so that the clause model can be used for most kinds of legal or business document.
7. XML markup must not be legally significant to the contract unless the parties specifically agree otherwise.
8. It must be as simple as possible and it must avoid semantic distinctions between objects except where clear benefits are shown.
9. Content must be re-usable at different levels of the hierarchy.
10. The model must allow incorporation of content by reference.
11. Other specific requirements to be determined.

The basic clause model is intended to deal with requirements 1 – 10 inclusive. However, this proposal does not demonstrate compliance with requirement 3 (markup benchmark contracts) because none have been defined. Nor does it deal fully with requirement 10. It is proposed to deal with it at a later time because it raises substantially beyond the basic structural model. However, anticipated needs of requirement 10 are considered in the model.

6. Why develop a new schema?

6.1 Previous review

As discussed in the recommendation submitted by Jason Harrop on 14 October 2003, there are no existing, generally available schema that provide the desired combination of structural markup capability and ease of use for document authors.

6.2 W3C proposal for XHTML 2.0

It is worth making some observations about the proposed XHTML 2.0 draft prepared by the W3C (See http://www.w3.org/TR/2003/WD-xhtml2-20030506/). The proposed

XHTML 2.0 will include a recursive hierarchical model that is similar in some respects to this proposal. It provides for a section element than can be used recursively to perform a similar function to the proposed Item element when it is used to create the document outline, as discussed in topic 9.3.2.

However, the current draft (draft 6) of proposed XHTML 2.0 retains the current XHTML presentational model for paragraph content (narrative content, as discussed in topic 9.3.3. In this respect the proposed XHTML 2.0 is unsuitable to satisfy the clause model requirements.

6.3 Possible significance of XHTML 2.0

The proposed clause model could be described generically as a “structural HTML”. If future drafts of the proposal were to provide a true structural model for narrative content, XHTML 2.0 could well provide a similar function to the proposed clause model. Notwithstanding its current limitations, it is possible that XHTML 2.0 will find more widespread use as a general purpose structural markup format than its predecessors.

The potential widespread adoption of a more structurally oriented XHTML 2.0 may make it that much more difficult for developers to be satisfied that it is worth supporting an alternative model to service a smaller market segment that requires the extra functionality. There is a risk that the work undertaken on a dedicated structural clause model could be overtaken by other developments.

6.4 Reasons for continuing with the clause model

The Technical Committee has already discussed the possibility of supporting multiple schema for the markup of electronic contract documents. There appears to be a consensus in favour of separating the contracts specific markup from the generic structural markup. If so, the contracts specific markup can be designed for use with a range of schema. This will provide the widest possible market for the specific contracts schema work developed by the Technical Committee and avoid the risks that the market for structural markup schema moves in an alternative direction to that represented by the structural clause model.

Despite the risks identified earlier, it is proposed that the Technical Committee should develop a structural clause model and comprehensive contracts document schema:

(a) Proposed XHTML 2.0 is still not a satisfactory structural model for the widespread markup of contract documents. A complete, simple structural model is still required.

(b) Developments of XHTML 2.0 take it in a direction that would make it comparatively easy to translate from the clause model to XHTML 2.0.

(c) Proposed XHTML 2.0 is only a draft and it is not clear what direction it will take. The Technical Committee needs a schema for contract documents that will provide a stable platform for development of semantic markup layers and as a means to provide potential users with a comprehensive framework for the markup of contract documents.

(d) Ignoring its limitations, the proposed XHTML 2.0 provides little extra functionality than the basic clause model for contracts markup. Much of the work is already done. If the basic clause model is adopted by the Technical Committee, it is still necessary to undertake extensive schema development. It will be easier to develop this on a platform under the Technical Committee's control, particularly in the first instance.

6.5 Specific recommendation

1. It is recommended that the Technical Committee proceed with development of a structural clause model proposed in this document.

7. Approach to design of the clause model

7.1 Serving the needs of two user groups

Two user groups directly interact with XML schema for documents:

- document authors (see topic 2.3); and
- application developers.

The perceptions and needs of these two groups are quite different. Where the different interests may affect the design, the competing interests will have to be resolved.

7.2 Ease of use for document authors

The clause model must be as simple as practicable for users (Requirement 8).

As stated in topic 2.3, it is assumed that lawyers may use an XML editor to draft contract documents. It may take quite a long time before this happens on a large scale because of the substantial change it will involve to conventional document authoring processes. As applications improve and use of the standard increases, more and more lawyers will adopt XML editing tools.

Much can be done in the design and customisation of many XML editing applications to enable authors to create content without having to be particularly aware of the markup. However, when an author wishes to undertake less common operations or they wish to modify a draft document by rearranging content into a different sequence or to different levels of the hierarchy, it is more difficult to shield the author from the markup.

The clause model design assumes that authors of contract documents who use an XML editor will be aware of the XML markup of their documents and that they will need to acquire a basic understanding of the DTD or Schema, particularly its clause model. A core aim of the design is to make the clause model simple for authors.

7.3 Ease of use for application developers

The clause model must be as simple as practicable for developers (Requirement 8).

As far as practicable, the design will not attempt to suit particular XML processing models. Developers should be free to adopt the tools that suit their needs. For the reasons explored in topic 2.5, it would be a mistake to align the model with any particular standard that may deprive the standard of the flexibility needed for its possible lifespan. The model has to be accessible to current processing tools so processing issues associated with particular common tools are not to be ignored.

7.4 Resolving the tension

To meet the requirement for ease of use for application developers and to balance those requirements with the needs of users, these principles are applied during design:

(a) Ease of use for authors will be favoured over ease of use for application developers.

(b) The design aims to provide only the minimum markup that will permit desired structural information to be logically inferred from the markup for processing purposes. Put another way, the clause model will contain only the minimum markup necessary for authors and developers to do their work. New elements must have a clearly demonstrated function. This approach aims to minimise the prospect that the resulting schema will grow into a complex and unwieldy model that will frustrate attainment of its objectives.

7.5 Selection of element names

The need for ease of use for authors takes precedence over ease of use for application developers. Consequently, elements are named for author perspective and convenience.

Element names must be consistent with requirement 6 which precludes the use of terminology that does not easily cross document types or jurisdictional boundaries.

It is proposed that the following principles will enhance author convenience:

(a) Names should consist of a single, short word. The objective is to make names distinctive, easy to read and to not take up more space on screen than necessary.

(b) Names should be chosen for their likely appeal to authors rather than to application developers. Persons introducing the schema to new users should not have to apologise for technical terminology.

(c) It is desirable that element and attribute names conform to a consistent orthography. As far as practicable, names should not consist of a mixture of abbreviated and unabbreviated words or mixtures of technical and non technical words.

7.6 Ongoing review of the clause model

During development a wide range of additional requirements, will be defined. Many of these are listed in requirement 11. From time to time it may be necessary to review aspects of the basic clause model in the light of those further requirements.

The basic clause model cannot be regarded as settled until the entire clause model is substantially complete.

8. Use of DTDs or Schema

8.1 Content models in this document

Content models in this specification use DTD syntax because this is more concise and easier to read than a XML schema.

8.2 Schema language options

It is necessary to determine the reference schema language for the standard and the level of support provided for other schema languages.

The options are:

- DTDs
- W3C Schema (XML Schema)
- RELAX NG Schema
- Schematron Schema.

The following table lists the support for each by major XML authoring applications.

Application	DTD	XML Schema	RELAX NG	Schematron
Microsoft Office 2003	No	Yes	No	No
Altova Authentic 2004	Yes	Yes	No	No
Corel XMetaL 4	Yes	Yes	No	No
Arbortext Epic	Yes	Yes	No	No
Adobe FrameMaker 7	Yes	No	No	No
Corel WordPerfect 11	Yes	No	No	No
XOpus	No	Yes	No	No
Topologi (not really an authoring tool)	Yes	Yes	Yes	Yes
Oxygen (not really an authoring tool)	Yes	Yes	Limited	No

8.3 Reference schema language

It will be necessary to determine:

(a) What is the reference schema?

(b) What other schema languages does the standard intend to support?

(c) Is it necessary to constrain the application so its capable of complete, consistent expression in multiple schema languages and, if so, which ones?

8.4 XML Schema (XSD)

8.4.1 Advantages

XML Schema are a W3C standard.

XML Schema are already widely supported by applications. This support appears to be growing. MS Word in Office 2003 supports only XML Schema.

The significant technical merits of XML Schema include:

- data typing;
- re-define element content models according to context;
- more easily mixed with namespaces.

8.4.2 Disadvantages

XML Schema are very difficult for humans to read.

Not all applications support XInclude which is commonly used by Schema developers.

8.5 DTDs

8.5.1 Advantages

DTDs are concise and comparatively easy for humans to read.

8.5.2 Disadvantages

DTDs lack some of the technical advances of Schema.

Support in new XML applications is not guaranteed (eg. MS Word).

8.6 RELAX NG Schema

8.6.1 Advantages

It is an OASIS standard.

It allows greater control and stricter conditions than XML Schema. See, for example Option 3 in topic 10.6.6.

It is easier to read than XML Schema.

8.6.2 Disadvantages

There is a lack of vendor support at this stage. However, it is noteworthy that the W3C is developing proposed XHTML 2.0 using RELAX NG Schema.

8.7 Choice of reference schema

The TC will likely need to select a reference schema in which to publish its standard and to define the limits of the functionality it offers.

The principal candidates today are DTDs and XML Schema. However, even RELAX NG Schema should not be ruled out.

There is the fundamental problem with DTD use that Microsoft Word and the XOpus editor do not support DTDs.

It is also highly likely that the standard will require Name Space support and that this is more conveniently implemented using XML Schema than DTDs.

While there are arguments in favour of proposing XML Schema as the reference schema for the standard, strictly, it is premature to make this recommendation until the Technical Committee's requirements are fully developed and the overall application architecture is defined.

8.8 Support for multiple schema

It seems inevitable that XML Schema will gain increased acceptance and that may applications will adopt and likely extend the standard using XML Schema, regardless of any decision to promote the standard using DTDs.

RELAX NG Schema may also gain support. There does not seem to be any reason why the Technical Committee would wish to preclude use of a particular schema. In a fluid technical and commercial environment there is no reason to attempt to pick winners.

Consequently, the Technical Committee should adopt a flexible approach to schema support.

A question has been raised as to whether it is necessary for the application can be fully implement ed in some or all alternative schema in addition to the reference schema.

Again, until the Technical Committees requirements and a technical architecture are developed it is not necessary to make a final determination of this issue. It is noted that as the specific features of XML or RELAX NG Schema are incorporated into the standard, it will become increasingly difficult to ensure that document prepared under those Schema will validate to a DTD.

8.9 Specific recommendation

2. It is recommended that the Technical Committee defer a decision on the selection of a reference schema language until its application requirements are fully developed.

9. Overview of the proposed model

9.1 Basic content model

For the purpose of introducing the basic pattern of a clause and the elements required to represent it, the proposed clause model is:

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

This is not the complete model tha t is required in practice. It is a simplification of the basic clause model proposal. By design, it also lacks elements for objects such as graphics, tables, annotations etc and inline objects that are essential to fully describe clause like content . These will be dealt with under requirement 11 and following once a basic clause model is accepted.

Simple examples of use of the proposed model to mark up clause structures are shown in these examples.

Example 1 Three clauses

1. Very simple clause

   Text of very simple example clause.

2. Nested objects without titles

2.1 Text of first nested clause or subclause.

2.2 Text of second nested clause or subclause.

3. Nested objects with titles

3.1 First nested object

Text of first nested object with a title.

3.2 Second nested object with a list

Text of second nested object with a list of the three main clause model elements:

(i) Item;

(ii) Block; and

(iii) Text.

These clauses would be marked up as follows:

```
<Item>
<Num>1.</Num><Title>Very simple clause</Title>
<Block><Text> Text of very simple example clause.</Text></ Block >
</Item>

<Item>
<Num>2.</Num><Title> Nested clauses or subclauses without titles </Title>
<Item>
<Num>2.1</Num>< Block ><Text> Text of first nested clause or
subclause.</Text></Block >
</Item>

<Item>
<Num>2.2</Num>< Block ><Text> Text of second nested clause or
subclause.</Text></Block >
</Item>

<Item>
<Num>3.</Num><Title> Nested clauses with titles </Title>
```

```
<Item>
<Num>3.1</Num><Title> First nested clause or subclause </Title>
< Block ><Text> Text of first nested object with a title.</Text></Block>
</Item>

<Item>
<Num>3.2</Num><Title> Second nested object with a list </Title>
< Block ><Text> Text of second nested object with a list of the three main clause model
elements:</Text>

        <Item>
        <Num>(i)</Num>< Block ><Text> Item;</Text></Block>
        </Item>

        <Item>
        <Num>(ii)</Num>< Block ><Text> Block; and</Text></Block>
        </Item>

        <Item>
        <Num>(iii)</Num>< Block ><Text> Text.</Text></Block>
        </Item>

</Block>
</Item>
```

9.2 Introduction to the elements

The proposed clause model is built around `Item`, `Block` and `Text` elements. These terms are chosen because they are considered to be document neutral and jurisdictionally neutral. They can fit in a contract, a letter, a pleading, an advice or virtually any other legal or business document in any English language jurisdiction.

A brief overview of the elements is as follows:

(a) An `Item` is the main structural element in the model. It occurs in two contexts. It is used recursively to create the basic document outline. In so doing it acts as the main structural division in documents (eg, chapters, parts etc) and as the container for the major text objects (eg, clauses, sections, articles, etc). Secondly, the `Item` occurs within the `Block` element. In this context it is used to create list structures. These distinctions are explained in detail in topic 9.3.

(b) 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.

(c) The `Text` element is the PCDATA element for text data content. Generally, it can exist only inside a `Block`.

(d) The `Num` element holds the numbers assigned to document components. It is shown only as a placeholder at this time. Component numbering is to be dealt with in a later stage of the development.

(e) The `Title` element is the heading, title or caption of an object.

These terms are further described in more detail in later sections.

9.3 The clause model pattern

9.3.1 Two overlapping hierarchies

The proposed model identifies two distinct, but overlapping hierarchies in structured narrative documents contracts and other legal and business documents.

9.3.2 Document outline

The first hierarchy consists of information objects (`Item` elements) that are arranged in a directly recursive hierarchy to form the document outline. These Items usually have titles and may be numbered. The document outline is usually revealed by the contents listing for the document, although this is not always so. In some cases, particularly those with many levels in the hierarchy, the contents listing shows only a part of the document outline.

At any level in the document outline, the author can decide to add narrative content (by inserting `Block` elements) or add another outline level (by inserting `Item` elements). In this way, the author organises the narrative content in a hierarchical structure. Titles or headings at each level assist readers to follow the author's structure.

Clauses 2 and 3 in Example 1 show a two level document outline. In a large document there could easily be several levels of structure above those Items.

9.3.3 Narrative content

Once the author enters narrative content (the Block element), the document outline terminates in that branch of the hierarchy. This is shown by clauses 2.1 and 3.1 in Example 1.

The narrative content can have its own hierarchy. Clause 3.2 in Example 1 shows a list within the content of that clause. Items in that list could contain further list levels.

Objects that are hierarchically inside the narrative content would not be expected to appear in a contents listing. If, for example, the numbered items in the list in clause 3.2 in Example 1 were to have titles, those titles would not appear in the contents listing for the document. Such content is not part of the document outline. In this document it is treated as part of the narrative content.

9.4 The overlap between the two hierarchies

9.4.1 Nomenclature – clauses and lists

In topic 9.2 it is explained that element names are to use terminology that can fit into a wide range of document types in many different jurisdictions.

This document defines a "clause model". Throughout the document it is necessary to refer to "lists" as part of the discussion. To avoid confusion and to set a firm foundation for the selection and use of elements in the model, it is necessary to explain these concepts and establish a context for their use when discussing the clause model. To do so, a series of examples of simple structures will be used. These are set out in Example 2.

Example 2 Simple cases

Case 1

1. Very simple clause

   Text of very simple clause.

2. Very simple clause

   Text of another very simple clause.

3. Very simple clause

   Text of a third very simple clause.

Case 2

4. More complex clause

(1) Text of first nested object.

(2) Text of second nested object.

(3) Text of third nested object.

Case 3

4. Complex clause

(1) Title of first object

Text of first nested object.

(2) Title of second object

Text of second nested object.

(3) Title of third object

Text of third nested object.

Case 4

(1) Very simple clause

Text of very simple example clause.

(2) Very simple clause

Text of another very simple example clause.

(3) Very simple clause

Text of a third very simple example clause.

Case 5

(a) Text of a simple object.

(b) Text of another simple object.

(c) Text of a third simple object.

Case 6

- Text of a simple object.

- Text of another simple object.
- Text of a third simple object.

Case 7

This paragraph contains the following items:

(a) text of a simple object;

(b) text of another simple object; and

(c) text of a third simple object.

Case 8

This paragraph contains the following items:

(a) **Title to the first object**

Text of a more complex object;

(b) **Title to the second object**

Text of another object; and

(c) **Title to the third object**

Text of a third simple object.

9.4.2 What is a clause?

Most would agree that Case 1 contains 3 clauses. Which of the other cases is or contains a clause? Most would probably agree that Case 2 is a clause and that Case 4 also contains 3 clauses because it conforms to the same pattern as Case 1. Only the numbering style is different.

In Case 3, is the object numbered 4 a clause? Whatever it is called, does it contain 3 clauses or are they subclauses?

Presumably Case 2 is a clause but does it contain 3 clauses or 3 subclauses? Are these objects different to those in Case 3 only because they lack titles?

Finally, what are the numbered or bulleted objects in Cases 5 and 6? Are they the same? Are they clauses or something else?

At the simplest level, we can probably say that a free standing text object that is numbered and has a title is a clause. Beyond this simple case, the categorization of objects as clauses or as something else becomes murky. What happens when the number is removed, is it still a clause? Is it necessary to have a title for it to be a clause? If so, why are the objects in Case 8 not clauses? Is it just that they are part of a single narrative?

The conclusion from this is that the true definition of a clause is illusory. Fortunately, this is of no importance. For the purpose of this document, the term “clause” is used in a general sense that could refer to any of the objects in Cases 1 to 6 in Example 2. The term “clause” is not used in a sense that requires it to have a precise meaning.

In the proposed clause model, clauses are created by allowing `Item` elements to occur recursively to any depth. It is up to applications and users to determine the form of citation they wish to apply to that structure.

9.4.3 What is a list?

Considering the Cases in Example 2. How many contain lists?

Most people would say that Cases 5 and 6 each contain a list but if Case 5 is a list, why are the objects (1) to (3) in Case 2 not a list?

The listed objects in Cases 5 and 6 are not preceded by introductory text. Contrast this with Cases 7 and 8. These two pairs reveal not only issues about what is a list but also how it is unclear whether lists can exist in the document outline as well as in the narrative content.

Cases 5 and 6 could be marked up in one of two ways under the proposed clause model. In Example 3, Case 5 is marked up one way and Case 6 the other.

Example 3 Markup of Cases 5 and 6

Case 5 markup

```
<Item>
<Num>(a)</Num><Block><Text>Text of a simple
object.</Text></Block>
</Item>
<Item>
<Num>(b)</Num><Block><Text>Text of another simple
object.</Text></Block>
</Item>
<Item>
<Num>(c)</Num><Block><Text>Text of a third simple
object.</Text></Block>
</Item>
```

Case 6 alternative markup

```
<Block>
<Item>
<Num>•</Num><Block><Text>Text of a simple
object.</Text></Block>
</Item>
<Item>
<Num>•</Num><Block><Text>Text of another simple
object.</Text></Block>
</Item>
<Item>
<Num>•</Num><Block><Text>Text of a third simple
object.</Text></Block>
</Item>
</Block>
```

The difference between the two cases is that the Items in Case 6 are contained by a `Block` element, while those in Case 5 are not. In other words, the Items in Case 5 are treated as part of the document outline, even though the author may conceive of them as a list. On the other hand, the Items in Case 6 are treated as part of the narrative content.

Which is correct and why would an author choose one over the other? There is no correct approach. In practice, it is expected that the author’s choice will depend upon the way in which document numbering is applied by the author’s applications and the degree of control given to the author to specify numbering patterns at particular parts of the document.

Example 4 shows how an author may need to markup the Items in both Cases 5 and 6 inside a `Block` element if the application adopts decimal numbering for the document outline, as in this document.

Example 4 The significance of numbering

Assume that the document outline numbering is as follows:

1.	Item at first level in document outline
1.1	Item at second level in document outline
1.1.1	Item at third level in document outline
1.1.1.1	Item at fourth level in document outline

If the author is at the second level and wishes to create a sequence of items with the numbering scheme in Cases 4 or 5, the author may need to enclose those Items with a `Block` to achieve the desired result. Otherwise, the Items would be numbered as 1.1.1, 1.1.2 and 1.1.3.

If the document outline is not numbered, only items in the narrative content would be numbered.

The conclusion to be drawn from these examples is that, ignoring markup, any series of numbered or bulleted items may be a regarded as a list. However, when we try to represent this in a universal way using markup, the position is not so clear.

There appears to be a continuum of structures that range from clauses to lists. At one end (Case 1), we are fairly certain that there is a clause. At the other end, (Case 7), we are certain we have a list. In between, there are shades of grey.

The previous topic demonstrates how `Item` elements are used in the document outline and also when contained by the `Block` element as part of the narrative content.

More commonly, lists of that kind will be preceded by introductory text, as in Example 5 (using simplified markup).

Example 5 Hierarchically correct list

```
<Block><Text>This is a list of primary colours:</Text>
<Item>(a) red</Item>
<Item>(b) blue</Item>
<Item>(c) yellow</Item>
</Block>
```

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

The content model of `Item` shown in topic 9.1 could be modified as follows:

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

This is described as the “loose model” in topic 10.4.

Under the loose model, Item elements can follow `Block` elements at the same level of the hierarchy, as shown in Example 6 (using simplified markup).

Example 6 Another approach to lists

```
<Block><Text>This is a list of primary
colours:</Text></Block>
<Item>(a) red</Item>
<Item>(b) blue</Item>
<Item>(c) yellow</Item>
```

In this example, the listed items are outside the introductory `Block` element. Particularly if numbered as shown in the example, many people would still regard this as a list.

The Item elements in Example 6 are structurally part of the document outline. Whether the automatic numbering to be applied in this context would match that shown in the Example will depend on the approach taken by the application developer to automatic numbering.

9.4.4 The document outline and narrative content boundary

Cases 5 to 8 in Example 2, Example 5 and Example 6 show that structures can exist in the document outline or in the narrative content and be regarded as lists by many authors.

In the majority of situations, authors will not need to consider how to create the "right" structure. The document outline will be made up of those `Item` elements that precede the narrative content. Once narrative content is created, further lists will be contained by the `Block` element because that is the only way to create a list from that context provided that the issues discussed in topic 11.5 are resolved.

In marginal cases, the examples show that authors may need to deliberately markup a list as either within the document hierarchy or within the narrative content to achieve a particular automatic numbering outcome where the application treats numbering differently in the document outline to numbering in the narrative content.

The approach taken by the clause model is that authors should not have to care as to whether something is correctly characterised as a clause or a list. This is achievable when there are only 2 ways in which the hierarchy can be represented. An Item is either part of the document outline (Items occur within a parent `Item` element) or it is part of the narrative content (Items occur within a parent `Block` element).

Based on this distinction, applications can satisfy users' complete numbering and presentation requirements.

The use of a single structure ensures that:

(a) authors can reorganise content during drafting just by moving objects into different contexts with minimal or no element transformation;

(b) regardless of who created a content object and their conception of its hierarchical context, it can be re-used easily (in a markup sense) by different authors in different documents; and

(c) applications and authors can employ their own conceptions of citation and structural naming without limiting those benefits.

9.5 Why a recursive model?

9.5.1 Basis for the recursive model

It was decided to adopt a recursive model under which the `Item` element can occur inside itself and inside element `Block`.

This decision is based on three key factors:

(a) The clause model is to be used for a wide range of legal and business documents in addition to contracts. There is no common, named hierarchical structure (e.g. Chapter, Division, clause, subclause) that will suit such a range of document types within even a single jurisdiction. It is considered that only a generic, recursive model or a generic, named hierarchy model could satisfy the standard's objectives in this respect.

(b) A recursive model provides all necessary information about the generic clause structure to human users and processing applications with the fewest number of elements. A generic, named hierarchy model might use terms such as "Level1", "Level2" or "Item1" and "Item2". These add no information to the markup that cannot be inferred from the hierarchical relationship of the `Item` elements. They frustrate re-usability and are inconsistent with requirements 8 and 9.

(c) Objects in a generic, recursive model can be relocated anywhere in the document hierarchy without re-tagging. This provides convenience to authors and facilitates element re-usability to satisfy requirements 8 and 9.

9.5.2 Limitations with a recursive model

When authors of draft documents wish to re-order or relocate `Item` objects within the document, they may need to work directly in a tags on view of the document, depending on the functionality of the XML editing application they are using. Particularly if the target location is between the end tags of a series of nested `Item` elements, it can be a little confusing for authors to work out where to insert the moved `Item`.

A recursive model does add some complexity to document processing applications. However, XML processing tools are improving rapidly. This problem will become less significant over time.

Another limitation with recursive and generic, named hierarchy models is that applications may lack explicit information necessary to determine which levels of Items from the document outline are to be included in contents listings for print and online publications. This is not normally a problem with those named hierarchical models that are specific to particular document types.

The first issue is considered to be minor compared to the problems of imposing on authors and applications the need to constantly re-tag objects as they are re-located in different parts of the hierarchy.

The second issue is unavoidable with a recursive model that uses only a single element to construct the document outline. Resolution of this issue is discussed in topic 10.6.

9.6 User customisations and data exchange

In practice, it is anticipated that many enterprise users of the standard will need to modify the schema in various ways, including by adding their own elements. This is common with almost all schema and there is no reason to expect it will not occur with the eContracts standard.

If users do customise the schema for their own use, the standard must address how users will exchange data with others who may use the standard schema or a different set of modifications. Issues that may need to be considered include:

(a) Is there one exchange model or will there be different models for different types of users who need different levels of information in the markup?

(b) How do parties include information about non standard extensions that may have been used in the source data and which may be of interest to the recipient?

It is not the purpose of the basic clause model proposal to resolve these issues.

9.7 Key points about the model

The proposed model provides a very simple way to markup any narrative text structure found in contracts and other legal and business documents.

The proposed model provides very limited options for authors to represent the same hierarchical structure in multiple ways. This strictness is intended to simplify application development, simplify training for authors and facilitate content re-usability.

The model provides a strict hierarchical representation of all content objects to that they can be conveniently and accurately referenced and processed by applications.

The proposed model is capable of representing all generic structural relationships that are needed for accurate document publishing and processing.

10. Item element

10.1 Content model

The simplified content model for purposes of the basic clause model is as follows:

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

Element `Block` is described in topic 11.

Element `Title` is described in topic 13.1.

Element `Num` is described in topic 13.2.

10.2 Purpose of the Item element

The `Item` element performs a range of functions:

(a) As a recursive element it is used to create the basic document outline, as explained in topic 9.3.

(b) It models the basic clause structure, i.e. narrative content that, in a contract is likely to be numbered and to have a title or caption.

(c) Within the narrative content, it serves as the list item.

These uses are all shown in Example 1 in topic 9.1.

Used recursively with a small number of other elements, the `Item` is able to perform these functions and provide a concise description of the generic document structure and maximise author convenience and content re-usability.

The `Item` element does not have required content. This allows authors to create an outline structure of `Item` elements. Until a `Block` is inserted in the leaf nodes for narrative content, it is likely to have only a `Title`.

10.3 Name selection

10.3.1 Options considered

Requirement 6 lists common names that are excluded from consideration.

Two candidate names were considered:

- Topic
- Item.

The Topic element is used in the proposed clause model submitted by Elkera Pty Limited (Peter Meyer), in the IBM DITA DTD. In each case it has a required Title element and is not also used in list item contexts. For these reasons, Topic was excluded as the name of an element that can exist at any level of the hierarchy. The topic element also occurs in the Topic Maps DTD.

In topic 10.7 the discussion considers whether it is desirable to create a distinct element for use as list items in narrative content. If the conclusion from that discussion is that a new element is required, the main constraint on the use of “Topic” in the document outline would disappear.

The term “Item” denotes a separate, identifiable piece of information. This reflects its intended use in the clause model.

10.3.2 Specific recommendations

3. The name “Item” is adopted as the primary clause equivalent element if that element name is allowed to exist at all levels of the hierarchy. [*Resolve after consideration of Recommendations 11 and 12*]
4. The name “Topic” should be adopted if a distinct element is created for list items in narrative content. [*Resolve after consideration of Recommendations 11 and 12*]

10.4 Content model options

The content model for Item set out in topic 9.1 represents the simplest pattern for the proposed clause model.

From the discussions in topic 9 several issues were identified that may affect the content model for the `Item` element:

(a) Should the content model for `Item` in the document outline be:

```
<!ELEMENT Item (Num?, Title?, (Block* | Item*))> (tight
model)
```

or

```
<!ELEMENT Item (Num?, Title?, Block*, Item*)> (loose model)
```

(b) Is it necessary to provide a facility to require the use of Titles on Items in the document outline and, if so, how should this be achieved (See topic 10.6)?

(c) How do we prevent authors from re-starting the document outline from within narrative content, i.e. creating lists by using `Block` > `Item` > `Item` rather than the desired `Block` > `Item` > `Block` > `Item`.

Each of these issues is considered in the following topics. In each case a recommendation is made or issues are identified for decision by the Technical Committee.

10.5 Use of the loose and tight models

10.5.1 The problem

In Example 6 it was shown how the loose content model for `Item` would allow `Block` and `Item` elements to exist at the same level to form an irregular document outline. The same, irregular hierarchy can occur when Items have Titles, as in Example 7.

Example 7 Mixed Paras and Items

1. A complex clause

Here is a Block before some Items. 🡐 Block before Items

1.1 Title of Item

Text of second nested clause or subclause.

1.2 Title of second Item

Text of third nested clause or subclause.

When this occurs in the document outline, the following problems can arise:

(a) The content of a `Block` element that precedes the Items, as in the above example does not directly appear in a contents listing. Contents listings become misleading.

(b) It may be confusing to authors as to how they should markup lists in the narrative content.

(c) 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. The extent of these problems may vary according to the size and number of documents and the publishing needs of the user enterprise.

10.5.2 Is there a need for the loose model?

The structure shown in Example 7 is rather uncommon in contract documents, particularly where clauses are numbered. However, this structure does occur from time to time in other documents. There are two key reasons for this:

(a) Authors omit the title before the first Block because it is usually some form of introduction. Use of a heading "Introduction" may seem redundant.

(b) Many authors simply do not share the view that such structures are anomalous.

Due to the flexibility of the proposed clause model, it is only necessary to use the loose model in the exact situation described in Example 7, i.e. where the Items in the document outline are numbered and it is desired to insert a paragraph of text before the first numbered Item. This is the most uncommon scenario.

In cases where the Items in the document outline are not numbered, the author can use `Item` elements everywhere in the sequence and omit the `Title` from the first or any other Item, if desired.

The conclusion is that almost all users should be able to work with the tight model. That model should not be constraining to authors in all but the most unusual circumstances.

It is expected that most organisations should implement the tight model for the creation of new documents such as contracts and other regularly structured documents.

In practice, the loose model may be required to markup legacy data that contains an irregular document outline. It may also be required by authors who wish to create an irregular document outline, as explained earlier.

It is proposed in topic 9.6 that the clause model should be capable of allowing users to exchange eContracts data while allowing them to maintain their own customisations. As the model capable of representing the widest range of documents, the loose model may provide a platform for that purpose.

For these reasons use of the loose model cannot be excluded. Provided that the issues are explained, it will be up to users to determine the model that best suits their needs.

10.5.3 Specific recommendations

5. The standard should allow use of both the loose and tight models.
6. The loose model should be designated as the standard for exchange of eContracts data between user enterprises.
7. The tight model should be recommended for general use.
8. The exchange of data using the tight model should be permitted by agreement between the parties. [The standard cannot prevent this.]

10.6 The use of Titles on Items in the document outline

10.6.1 The problem

Based on the content model for `Item` in topic 9.1, `Item` elements in the document outline may have a `Title`. A `Title` is not required and cannot be required on an element that serves the range of functions proposed for the `Item` element.

An important characteristic of the document outline to human readers of documents and to publishing and processing applications is whether objects in the outline have titles.

Example 8 shows how an author might create a document outline using the proposed clause model.

Example 8 Inconsistent use of titles

1. Title of first item

1.1 Title of first nested item

Text content of first nested item.

1.2 Text content of second nested item 🡐 No Title

1.3 Title of third nested item

Text content of third nested item.

2. Text content of second item 🡐 No Title

3. Title of third item

Text content of third item.

In Example 8, the objects numbered 1.2 and 2. do not have Titles, unlike all other objects at the same levels in the hierarchy. The author is able to add or omit titles at will, resulting in inconsistent usage.

The inconsistent use of Title elements may cause significant problems for applications that build contents listings for documents that are published in print or online. In particular:

(a) Omission of titles causes contents listings to be inaccurate in print documents.

(b) Omission of titles causes content to be almost completely invisible in web publications. There may be no hypertext link to the relevant content or else it will lack any descriptive information.

(c) If headings can be added or omitted at the will of the author, how does an application know how many levels of the hierarchy at particular nodes should be included in the contents listing?

(d) An application may specify that, say, the first 3 levels of items with titles will be included in the contents listing. What does the application do if none or only some Items at that level have titles? Does it have to attempt to analyse the data and take action based on rules provided by the application developer? If such processing is required, this will add considerable complexity to applications.

The extent of these problems may vary with different document types. Authors of contracts may be expected to use titles fairly consistently. Authors of other documents such as advices, reports and manuals may be more likely to be less strict in the use of Titles.

A classic problem occurs at the cross over point in the document hierarchy where authors cease to use titles on the structural objects. This is shown in Case 2 in Example 2. Using clause based terminology, this is where the author switches from clause to subclause. Unfortunately, the recursive model does not allow us to distinguish these cases. Applications would need to query the data to determine where the author has stopped using Titles. This is unsatisfactory at the best of times but produces unpredictable results when titles are used inconsistently at particular levels.

In very many cases there will be a boundary in the document outline at which Items are included in the contents listing and below which they are not. The problem is to easily and reliably determine that boundary.

10.6.2 Development issues

It is not possible to adopt a content model for Item that requires a `Title`. The author must be able to choose whether titles are required at any particular level of the hierarchy.

These problems encapsulate two important issues in development of the schema:

(a) Is it desirable to include a mechanism in the schema to allow applications to know which levels of the document outline can be included reliably in print and online contents listings?

(b) Should authors be able to signify their intentions about the items they want included in the contents listing? If so, how should this be done?

10.6.3 Desired functionality

The conclusion reached is that it would be highly desirable if the design could:

(a) require authors to consistently create Titles for Items at levels of the hierarchy that are expected to be included in contents listings;

(b) allow authors to easily signify the particular nodes that are to be included in or excluded from contents listings; and

(c) allow applications to process documents reliably without having to analyse the content to determine whether the use of Titles is sufficiently consistent to justify inclusion in the contents listing.

10.6.4 Option 1 – Use metadata to determine the contents boundary

Proposed solution

It is considered highly desirable that authors should not have to add metadata to each `Item` element as it is inserted. This would become extremely burdensome. It is also highly desirable that metadata should not be added to every `Item` element. This would cause authors to have to frequently change values as content is relocated in draft documents. This places unwanted burdens on authors and creates situations where they may not realise that particular settings are in place.

Under this option, the Item element would have, say, an optional attribute with a single value:

```
    <!ATTLIST Item  StopContentsBelow   (Yes)  #IMPLIED>
```

This markup would be applied under the following processing principles:

(a) Particular document types that do not require contents listings would be processed on that basis without any need to set this attribute value.

(b) Document types that require a contents listing would assume that all Items in the document outline are included in the contents listing unless a stop value is encountered.

(c) Once a stop value is encountered, all Items below that level would be excluded from the contents listing.

Advantages

(a) It provides a convenient mechanism for authors to signify their view of the content.

(b) It provides certainty to processing applications, subject to the problem that expected Titles may not be present.

(c) It avoids the need to translate element markup as content is re-used in different contexts.

(d) It provides a means for authoring applications to provide a markup checking utility to warn authors of any Items that are within the scope of contents generation but lack the expected Title.

(e) It does not prevent applications from applying other rules to the data if they are desired for particular document types or for particular publications.

(f) Users and developers can elect to support or ignore this feature without affecting data exchange.

Disadvantages

(a) It does not enforce the use of Titles where they are expected for contents generation.

(b) It may be burdensome for some processing applications to exclude Items from contents generation based on the value of an attribute on a parent or ancestor element. However, XSLT manages this quite easily.

10.6.5 Option 2 – Use an alternative element with a required Title

Proposed solution

The clause model could include another container element called `Topic`. 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 `Block`. The content model for the `Topic` element would be as follows:

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

Under this content model, there would need to be a general principle understood that applications should use only Topic elements to build contents listings. Authors would insert `Item` elements when they no longer wished to create consistent Titles or they wanted the Items excluded from possible contents listings.

This approach could be an optional extension to the standard for use by those enterprises and developers who find it useful.

Advantages

(a) 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.

(b) It simultaneously solves both problems: certainty about the presence of Titles when they are expected and definition of the boundary for contents generation.

(c) Application developers have complete certainty about the content that is available for inclusion in contents listings and for possible page chunking in web publications.

(d) Authors have a convenient mechanism to decide how to structure their data, subject to the possible need to translate markup as content is moved from one context to the other.

(e) Users and application developers do not have to adopt this option if they do not require it.

Disadvantages

(a) All applications must provide for two elements that perform similar functions in the document outline. This increases application complexity.

(b) Authoring and some processing applications would have to provide facilities to translate between Topic and `Item` element markup as content is moved to the other context during document drafting and document assembly type operations.

10.6.6 Option 3 – Apply a context based content model on Item

Proposed solution

This option cannot be implemented under current XML Schema standards but is likely to be possible under a future version of the standard (see the Requirements for proposed version 1.1 of the XML Schema standard at http://www.w3.org/TR/xmlschema-11-req/#N400120.

This option requires the use of attribute values to determine the context for an element declaration. This option could be implemented under RELAX NG Schema. However, the most widely used XML document authoring applications do not support RELAX NG Schema at this time.

The model is very similar to Option 1, except that the use of Titles would then be controlled by the Schema. If the initial assumption is that they are required, they would remain required until the author applies a not required value.

Advantages

(a) This option has all the advantages of Option 1, plus it enforces the use of Titles where they are expected. It would effectively provide the functionality of Option 2.

(b) It is consistent with Option 1. If Option 1 is implemented, it is likely to be simple to implement the Schema based solution.

Disadvantages

(a) It cannot be implemented at this time.

(b) Authors or applications moving from a non required `Title` context to a required Title context would have to ensure that a `Title` is added, if it is not already present.

10.6.7 Specific recommendations

9. It is recommended that the TC adopt Option 1 as the preferred option, in principle, at this time. It should not be formally included in the model but should be coordinated and implemented with other metadata requirements in a later phase of the development.

10. It is recommended that the discussion on Options 2 and 3 should be left in the draft specification and that feedback should be sought from interested parties during a review of a draft standard. A final decision should be made at that time to either exclude the options or recognise them as optional extensions for non exchange purposes. [Note: Option 2 can be added at any time by anyone who wants to use and support that option, without impact on the rest of the clause model. Option 3 can be implemented at any time by anyone using RELAX NG Schema and supporting applications.]

10.7 Should Item be used for lists in narrative content?

10.7.1 The problem

The loose and tight models allow an `Item` to contain another `Item`. This is desired behaviour in the document outline but it is not desired in the narrative content. As discussed in topic 9.4, list structures in the narrative content are created by containing items inside the `Block` element. If an author can also insert an `Item` inside an `Item` in that context, authors will be apparently able to revert to the document outline from in the narrative hierarchy. This will not add any flexibility to the markup but create difficulties for authors and developers.

There is also the problem described in Example 6. This allows authors who use the loose clause model to create lists in the narrative content which contain a sequence of `Block` and `Item` elements, thus breaking the strict hierarchical relationship and providing another way to represent lists in the narrative content. This problem arises only under the loose clause model.

In topic 9.4.3 it is shown that lists in the narrative content should be created only within a containing `Block` element. Items that contain content conforming only to the loose clause model do not satisfy this requirements and ought not be inserted into a list context in the narrative content. It is not clear how much of a problem this will present in practice. Normally, an author would expect to re-locate a single item node from the document outline into a list context in the narrative content. It is difficult to see why an author would expect to be able to move an `Item` with contains Items into a list in the narrative content.

10.7.2 Option 1 – Redefine Item according to context using XML Schema

The proposal

In topic 9.4.3 it is shown that it may be necessary to characterise a list as either part of the document outline or as part of the narrative content to achieve particular numbering preferences. If the Item element is available in all contexts, authors can easily switch from one to the other by moving Items in or outside a containing `Block` element.

It also occurs that, while drafting, authors wish to re-draft and convert lists in the narrative content into Items in the document outline. The reverse may also occur.

To facilitate these changes and to allow complete flexibility for content re-use, it is desirable that content can be moved between the document outline and the narrative content without element transformation.

To achieve this objective, the content model for `Item` in the narrative content should be:

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

Using DTDs it is not possible to modify the content model of `Item` according to context. It is possible to do so using XML Schema and Relax NG Schema.

This proposal should be adopted if XML Schema are adopted as the reference schema language for the standard.

If the reference schema language is DTDs, it is still open to users to use this model by implementing XML Schema or RELAX NG Schema for their applications. All data created under this model will be valid under a DTD based standard.

Advantages

(a) It avoids creation of a new element and the disadvantages of Option 3.

(b) It maintains complete content re-usability when the “tight clause model” is used, as recommended in topic 10.5.3. `Item` elements can always be moved from the narrative content to the document outline. `Item` elements that do not themselves contain direct `Item` elements can be moved from the document outline to the narrative content.

Disadvantages

(a) Users who rely only on DTDs will have to deal with the possible disadvantages of there being two ways to insert Items in lists in the narrative content.

(b) DTD users may create data that will not validate under a Schema.

10.7.3 Option 2 – Ignore the content model problem

The proposal

The problem caused by allowing recursive `Item` elements in the narrative content is not particularly serious. In most XML authoring applications the problem will be avoided because `Item` elements inserted within the `Block` context should also include the contained `Block` element, thus preventing the author from making the mistake.

It is also highly unlikely that authors will allow the mistake to stand since automatic list numbering is unlikely to produce the expected results, assuming that numbering is displayed by the authoring application.

Advantages

(a) It preserves complete element re-usability.

(b) Most users should be able to adopt XML Schema and avoid the problem.

(c) On the assumption that XML Schema are likely to become the de facto standard in the future, if not the de jure standard, it avoids creation of an element that will become redundant at that time.

Disadvantages

(a) The content model problem is unresolved. Some authors may create lists in the narrative content in unintended ways.

10.7.4 Option 3 – Create a new element

The proposal

It is proposed that a new element should be created to replace `Item` in the narrative content (i.e. inside the `Block` element). Currently, the element is called "ListItem".

In support of this option, it is argued that a ListItem element will overcome issues with the content model for all schema users, albeit at the expense of limiting re-usability of content in some cases. It is argued that movement of list items between the narrative content and the document hierarchy is not common and the benefits of content model certainty outweigh the benefits of content reusability.

It is proposed that if this option is adopted, element names should be reviewed, as discussed in topic 10.3.

Advantages

(a) It allows only one way to create lists in the narrative content, thus avoiding possible, undesirable markup.

(b) DTD users have the same content model as Schema users.

Disadvantages

(a) It prevents direct re-usability of content between the document outline and the narrative content. List examples in Cases 5 and 6 must use different element markup. If authors wish to swap from one context to the other, it would be necessary to convert element markup. This would create the only context in which re-usability is inhibited in this way. It violates requirement 9.

(b) It creates a redundant markup distinction between elements that are otherwise adequately differentiated by their context as either within or outside a `Block` element.

10.7.5 Specific recommendations

11. It is recommended that the TC make a provisional choice between Options 1, 2 and 3 so that a working clause model is established for further development.

12. It is recommended that the TC review Options 1 to 3 once a decision is made on the questions set out in topic 8.3 and the requirements for inline lists (discussed in topic 11.7) are known.

10.8 Outstanding issues

This initial proposal does not seek to address any issues other than those necessary to the basic structural model. Outstanding issues include:

(a) in line lists;

(b) element identifiers and linking;

(c) metadata;

(d) higher level structural components of contract and other documents.

11. Block element

11.1 Content model

The simplified content model, for purposes of the basic clause model is:

```
<!ELEMENT Block (Text | Item)+>
```

Element `Text` is described in topic 11.7.

Element `Item` is described in topic 10.

Note: In a complete implementation, this content model will include other equivalent objects such as tables, block graphics, block quotations etc.

11.2 Purpose of the Block element

In simple cases the `Block` element broadly corresponds to a grammatical paragraph. However, because it is a container for a wide range of objects and because of its use at all levels of narrative content, the correlation does not extend beyond the simplest paragraphs.

The `Block` and `Text` elements are the basic content 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 separates narrative content from the document outline, as described in topic 9.3.

The `Block` element cannot be directly numbered. If a numbered structure is desired it must be enclosed by an `Item` element.

The `Block` element may never directly contain text data. The `Block` element must contain at least a `Text` element which contains the text data. The `Block` element will also contain tables, block graphics and other similar components. These other components are not included in the basic clause model proposal but will be considered in a later phase of the development. The `Block` element provides a container for all grammatically nested components so they may be manipulated as a group.

The following example shows how the `Block` element contains all components of a complex grammatical paragraph. This ensures that inside a clause structure (`Item` element), grammatical paragraphs can be distinguished from components such as the continuation of the paragraph after a list.

Example 9 Operation of the Block container

```
<Item>
<Num>1.</Num><Title>Foxes and lazy animals</Title>
<Block>
<Text>The quick brown fox jumps over the lazy:</Text>
<Item><Num>(a)</Num><Block><Text>dog;</Text></Block></Item>
<Item><Num>(b)</Num><Block><Text>cat;</Text></Block></Item>
<Item><Num>(c)</Num><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>
</Item>
```

The operation of the `Text` element is considered in a later topic.

Lists in the narrative content are represented by `Item` elements contained by the `Block` element, as discussed in topic 10.2. It is proposed that a single element can be used for all lists and that is not necessary to define ordered lists and unordered lists, as exists in HTML and some other DTDs such as DocBook. The distinction between these is based solely on the author’s selection of numbering options. Frequently, authors wish to switch from one option to the other. This is very common as content is re-used by different authors and in different documents. The use of distinct elements for each numbering type would unnecessarily burden authors and frustrate content re-use.

11.3 Name selection

11.3.1 Options considered

The following options were considered:

- Block
- Para
- Clause.

11.3.2 Arguments relating to “Block”

The proposed element does not have any real equivalent in a word processing model. A word such as “Para” implies a correspondence with word processing paragraphs that does not exist. Word processing documents do not explicitly recognise hierarchically based containers.

The term “Para” is likely to create confusion when contract drafters refer to a “paragraph”. This will normally signify a list item but could easily be confused with the Block element.

“Para” is an abbreviation of “paragraph”. This is in conflict with the names expressly excluded by requirement 6. Abbreviations are generally avoided in the model and are discouraged, as proposed in topic 7.5. The word is aesthetically inconsistent with other terms used in the clause model.

It is argued that a more neutral term is required and that “Block” serves this need. This term avoids association with word processing concepts or with particular document types.

In proposing “Block” it is argued that any technical associations for developers are irrelevant. They do not need to rely on the element name to tell them its rendering function. They will determine from the element structure how it should be processed. In any event, the element names should be chosen for convenience of authors over developers.

11.3.3 Arguments relating to “Para”

It is argued that “Block” is a technical term for developers who will assume that it is to be rendered as a block (new line) element in a CSS sense when in fact it is the Text element that would normally be treated as the new line element. It is argued that developers will be confused.

It is then argued that the term “Para” recognises that in many cases the element is representing a grammatical paragraph. On this basis, “Para” is a better candidate than “Block”.

11.3.4 Arguments relating to “Clause”

The clause model is a recursive model, as discussed in topic 9.5. It seeks to apply to a wide range of legal and business documents in many jurisdictions. The term “clause” is inappropriate for the basic recursive container object in the clause model for these reasons:

(a) It is contrary to common terminology to call all the hierarchical levels before the “real clause” (the one that starts the narrative content) clauses.

(b) When people think of clause, they then think of subclause. Use of a subclause element conflicts with the use of a recursive model.

(c) The model is applicable to documents with numbered and unnumbered components. The use of the term “clause” is uncommon for unnumbered objects such as a chapter in an article or book.

(d) The term “clause” may be in common use for contracts by some users in some jurisdictions but this is not universal. The term will be out of place for many users who will have already associate a particular meaning to the term.

(e) The term “clause” does not even suit numbered objects in some legal and business documents. In many jurisdictions the term is not commonly used in litigation pleadings, for example. Most people, particularly outside the legal industry, would not use it at all for non contract documents.

The net result is that the term “clause” is a poor choice for a recursive element in the proposed clause model. The clause model aims to avoid this by creating a new, neutral language that does not denote particular citation patterns that suit some users but not others or some documents but not others.

11.3.5 Specific recommendation

13. The Technical Committee is invited to choose between “Block” or “Para” or propose an alternative name that satisfies the objectives set out in topic 7.5.

11.4 Content model options

Two variations on the proposed content model have been considered:

(a) Create a distinct element for list items within the narrative content;

(b) Create a list container element for lists within the narrative content.

Each of these proposals is discussed in the following topics.

11.5 Is a distinct list item element required?

This issue is fully considered at topic 10.7.

11.6 Is a list container required?

11.6.1 The problem

It has been proposed that a list element is required for the markup of lists in the narrative content. This would produce the following content models, ignoring the possibility of a list item element:

```
<!ELEMENT Block (Text | List)+>
<!ELEMENT List (Item)+>
```

There is nothing per se in the basic clause model that requires use of a list element to determine the generic structure of the narrative content. Authors and applications can determine that an `Item` is functioning as a list item in the narrative content simply because it is contained by a `Block` element. No further structural markup is required.

It is necessary to deal with this issue as part of the basic clause model development. Inclusion of a List element will affect other aspects of the proposed clause model.

Four arguments have been offered in support of the List element.

Firstly, it is proposed that the List element will be required to hold attribute values to control numbering options for list items in the narrative content. Most commonly, an author may wish to specify that a list is to be bulleted, rather than numbered using a standard numbering sequence such as (a), (b) etc. How is this choice to be stored for ongoing use by the applications?

Secondly, it is also argued that the problem in Example 10 cannot be dealt with except by creation of a List element.

Example 10 Two lists in a Block

Simplified markup example:

<Block>
<Text>Here is a list of the primary colours:</Text>

(a) red

(b) blue

(c) yellow,

<Text>followed by the rest of the spectrum colours: </Text>

(c) orange

(d) green

(e) indigo

(f) violet.

</Block>

If the author wished to make the second list bulleted, how would this be signified? It is proposed that a List container could hold an attribute value to reflect this choice.

Thirdly, it is argued that the List container is required for possible use by processing applications involving document assembly processes. In particular, applications that assemble lists from precedent Items, may need to determine the penultimate list item so that the correct wording can be inserted, depending on whether the list is disjunctive or conjunctive. It is argued that the List container would make this process easier in some processing applications.

Fourthly, it is argued that lists structures are commonly understood by authors and that the use of a list element provides explicit recognition of that understanding.

11.6.2 Option 1 – Omit the List element

The proposal

It is argued that the List element is unnecessary and should not be created.

The first argument in support of the List container does not justify its use. Attribute values to control automatic or manual numbering options can be added to the containing `Block` element. There is no logical reason why that element is less suitable than a List element for this purpose.

It is acknowledged that the functionality proposed for the second list (second argument) could not be sensibly achieved if numbering options are set on the containing Block.

The second argument in support of the List element is not based on practical requirements. The suggested problem can be managed in these ways:

(a) Multiple lists in a `Block`, as shown in Example 10 are very rare.

(b) An application can and should define a standard numbering sequence for such structures, if they are likely to occur.

(c) If the author wishes to apply a custom numbering sequence to the second list, they could set the whole list sequence to manual numbering and handle the list numbering manually to achieve a desired result.

(d) More likely, if the author considers that the standard numbering scheme is inadequate and an alternative numbering scheme should be applied to the second list, the author can re-write the structure and separate it into two Paras.

The third argument is rather speculative. All the processing problems can be solved without the List element. Even if it is more convenient for some processing applications to have the List element, this is insufficient to justify creation of a new element. Processing applications are constantly evolving and improving, as discussed in topic 2.5.

The fourth argument is in direct conflict with the analysis in topic 9.4. From that analysis it is clear that authors would have a broader conception of what is a list than a series of numbered objects contained by a Block element.

It is also clear that there are many circumstances in which authors will need to move list structures between the document outline and the narrative content. Why have a list container in one but not the other?

In summary, it is proposed that the basic model will provide fully adequate facilities for authors without creating a new element that will be pervasive in the markup.

Advantages

(a) The clause model is simpler, yet no less effective in describing the generic structure of narrative content. A redundant element is avoided.

(b) All practicable functionality that is offered to justify the List element can be provided without that element.

(c) Explicit recognition of a “list” structure is avoided, as promoted in the clause model requirements and in topic 9.4 and following.

Disadvantages

(a) Very occasionally, an author might feel they need to re-write a double list structure.

(b) Some application developers might find it less convenient to undertake document assembly or similar functions.

11.6.3 Option 2 – Create a List element

The proposal

It would be necessary to create new content models as set out in topic 11.6.1 or as follows:

```
    <!ELEMENT Block (Text | List)+>
    <!ELEMENT List (ListItem)+>
    <!ELEMENT ListItem (Num? Title?, Block+)>
```

Advantages

(a) It provides a container element that can hold the attributes proposed and it might assist some processing.

Disadvantages

(a) The element is structurally redundant.

(b) The element adds unnecessary complexity to the process of changing lists from the document outline to the narrative content.

11.6.4 Specific recommendations

14. The arguments in favour of the List element mainly revolve around specification of numbering options. It is also possible that the issue will be affected by the requirements for inline list, discussed in topic 11.7. It is recommended that:

    (i) the TC defer a final decision on the List element issue until the requirements for managing component numbering and inline lists are determined; and

    (ii) pending a resolution of those requirements, the `List` element should be provisionally excluded from the clause model.

11.7 In line lists

11.7.1 The problem

In some United States jurisdictions and possibly elsewhere, list structures are rendered in line so that each numbered item does not start a new line when rendered in print or online.

At this point, the clause model does not address this need. Before adapting the clause model it is necessary to determine the requirements for support of in line list structures. The following questions are proposed:

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

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

(c) If markup is provided for inline lists, would authors reliably mark them up or would they sometimes just type in the text and forget about the markup?

(d) Is it necessary to create 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 (Block element)?

(f) 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?

(g) Apart from inline or new line rendering, are there any other material differences between in line lists and conventional list?

Until these issues are resolved it is not possible to assess how possible support for inline lists might affect the basic clause model proposal.

11.7.2 Specific recommendation

15. It is recommended that the TC provide guidance on the 7 questions regarding the requirements for support of inline lists.

12. Text element

12.1 Content model

The simplified content model is:

```
<!ELEMENT Text (#PCDATA)>
```

Note: In a complete implementation, this content model will include elements for inline markup of objects such as defined terms, references and citations etc.

12.2 Purpose of the Text element

12.2.1 Overview

The `Text` element is intended to hold all character data content inside the `Block` element and, possibly other similar contexts. It functions as a new line element, not as a new grammatical paragraph. It is not the word processor equivalent of pressing the Enter key. Rather it is the equivalent of Shift + Enter in Microsoft Word.

A `Block` may contain multiple `Text` elements interspersed with Items or other objects.

The `Text` element could be dispensed with as redundant in most situations. The clause in Example 9 could be marked up as follows:

Example 11 Alternative clause markup without the Text element

```
<Item>
<Num>1.</Num><Title>Foxes and lazy animals</Title>
<Block>The quick brown fox jumps over the lazy:
<Item><Num>(a)</Num><Block>dog;</Block></Item>
<Item><Num>(b)</Num><Block>cat;</Block></Item>
<Item><Num>(c)</Num><Block>rabbit,</Block></Item>
and any other animal that may be sleeping.</Block>
<Block>More alert animals won’t be caught so easily.</Block>
</Item>
```

The `Text` element is added to meet a range of author and processing needs, including:

(a) It provides “new line” functionality to add greater flexibility for authors, as discussed in topic 12.2.2.

(b) It makes content creation more consistent for authors because the `Text` element provides a placeholder and slot for data entry.

(c) It assists some processing by consistently separating new line content from inline content.

(d) Subject to the outcome from recommendations in topic 12.4, it provides a mechanism to control content that is normally rendered on a new line but sometimes rendered inline, as discussed in that topic.

The principal issue to overcome with the `Text` element is that to meet these needs, the `Text` element becomes pervasive. Every `Block` must contain a `Text` element. In practice this does not present any problems for authors. They will readily understand its benefits as a new line control feature and, depending on the functionality provided by their editing application, should rarely have to directly insert the element during content authoring. It is expected that applications will insert the `Text` element by default each time a `Block` element is inserted. Authors can remove it on those occasions when it is not desired.

It is considered that the balance of convenience is strongly in favour of use of the `Text` element and the flexibility it provides.

12.2.2 New line functionality

New line functionality is useful in the markup of content in tables. Table markup is strongly presentation oriented. From time to time an author needs to control where a break will fall for arbitrary heading like content in tables or for other content.

New line functionality is also useful in narrative content as an alternative to list markup from time to time. 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 where the author wants them to align to the left margin for the containing `Block`.

12.2.3 Hard cases

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 may allow the data to specify whether this continues in line or starts a new line.

A further issues occurs with quotations. Occasionally, the quotation may be followed by data that is sometimes rendered on a new line and at other times rendered in line. This is shown in Example 12.

Example 12 Continuing text after quotations

Consider the following two quotation examples (using simplified markup):

Case 1 – Text after quotation is desired on a new line.

```
<Block>
<Text>In Doe v Bloggs, it was stated:</Text>
 <Quote>
   "some great point of law, blahh, blahh.
   Blahh, blahh."</Quote>
<Text>See Denning LJ at p 400.</Text></Block>
```

Case 2 – Text after quotation is desired in line.

```
<Block>
<Text>In Doe v Bloggs, it was stated: </Text>
 <Quote>
   "some great point of law, blahh, blahh.
   Blahh, blahh."</Quote><Text>, per Denning LJ at p 400.
</Text></Block>
```

See also http://www.judiciary.state.nj.us/style.htm (at point 3A) for another example similar to Case 1.

An application would not be able to render the following Text in Case 2 inline unless it can be distinguished from the Text in Case 1. Some form of element markup is required to deal with this issue.

This particular problem is not common in contracts but it does arise in other legal documents. Sometimes it is just a question of adding the period after the quotation. Authors should not be forced to include it inside the quotation markup.

In the case of more complex content, it is not an option to include it inside the quotation.

It is necessary to provide authors with a convenient facility to deal with these situations. The options for this are considered in topic 12.4.

12.3 Name selection

12.3.1 Options considered

The following options were considered:

- Text
- TextBlock

“Text” is a simple, short word that conveys the desired meaning to authors.

“TextBlock” incorporates the word “Block” to signify to developers that the element is a new line or block element. This is redundant and contrary to the principles discussed in topic 7.5.

12.3.2 Specific recommendation

16. It is recommended that the TC adopt the name “Text” for this element.

12.4 Content model options

12.4.1 The problem

In Example 12 two quotation examples show the need for a mechanism to control whether content within the same grammatical paragraph that follows a block object should continue in line or start on a new line.

The `Text` element has been introduced as the new line element. It would be used to markup the concluding text in Case 1 in Example 12. How should Case 2 be marked up?

The clause model requirements treat the issue of quotations and the markup of similar objects as part of requirement 11 which is to be dealt with at a later stage. However, the operation of the basic clause model, particularly the role of the `Text` element, may be affected by the way in which those issues are handled. It is not the purpose of this discussion to settle a way to handle quotations but to clarify the role of the `Text` element as part of the structural model.

12.4.2 Option 1 – Create a special element for the inline text.

The proposal

It is proposed that a content model similar to the following may be needed for the markup of quotations:

```
    <!ELEMENT Block (Text | Item | BlockQuotation)+>
    <!ELEMENT BlockQuotation (Block+, TrailingText?)>
    <!ELEMENT TrailingText (#PCDATA)>
```

The TrailingText element could be used to mark up a period (full stop) or text such as that shown in Case 1 in Example 12.

This proposal is promoted on the basis that it ensures that the `Text` element can never be treated as an inline element and that this certainty is necessary for document processing. It is argued that the proposed override behaviour on the `Text` element (Option 2) creates particular difficulty for XSLT processing.

It is argued that it is inherently undesirable and bad design to have an element that can behave as either an inline or new line element, regardless of how that behaviour is specified.

Advantages

(a) It may facilitate easier programming.

(b) There are no exceptions to the new line behaviour of the `Text` element.

Disadvantages

(a) The model is confusing about the content of the quotation. The TrailingText is not part of the BlockQuotation but is included inside that element. This is semantically incorrect. To solve this problem, it would be necessary to add a further container around the BlockQuotation element, adding even more redundancy to the markup.

(b) It introduces a new element to markup narrative text. Everywhere else, narrative text is contained in a `Text` element. This exception is confusing to authors and application developers.

(c) It adds unnecessary complexity to style sheets by creating another element that duplicates much of the functionality of another element.

(d) It adds markup that is driven by perceived processing convenience today, not user convenience for all time.

(e) It is more difficult for authors to switch from one layout to another.

12.4.3 Option 2 – Create an exception to the new line behaviour of Text

The proposal

It is proposed that a content model similar to the following may be needed for the markup of quotations:

```
<!ELEMENT Block (Text | Item | Quotation)+>
<!ELEMENT Quotation (Block+)>
<!ELEMENT Text (#PCDATA)>
<!ATTLIST Text text.flow (runon) #IMPLIED)>
OR
<!ATTLIST Quotation text.flow.after (runon) #IMPLIED)>
```

Note: The use of the Quotation element is illustrative only. It is not proposed that an element called Quotation should be created. Rather, an element should be created to meet a general need for inclusion of content that is not part of the main narrative flow. This is to be addressed under requirement 11.

The effect of this proposal is that the author can override the normal new line behaviour of the `Text` element by setting an attribute on it. This would cause processing systems to render it in line with the preceding element.

Alternatively, it would be possible to add the override value to the Quotation element. Either solution would be acceptable under this proposal. The net effect is similar: there would be an exception to the principle that `Text` element is always a new line element.

However, setting the attribute on the Quotation element would ensure that the processing application only has to act on the first `Text` element after the Quotation. If the attribute is set on the `Text` element, it may be necessary to evaluate multiple successive `Text` elements to determine their behaviour.

It is argued that any processing inconvenience is minor. Processing can be handled and we should not make decisions such as this based on particular processing issues. One of the aims of the proposed standard structural model is to make it worthwhile for developers to do the work necessary to fully implement an XML markup model. Once the programming is done, the problem is gone. Redundant element markup is with us forever.

It is argued that there are no inherent principles that prevent a general new line element from having exceptional behaviour to allow it to become a new line element, provided the reasons for so doing are clear.

Advantages

(a) It literally and accurately models the data in the simplest way.

(b) It avoids creation of a special purpose element with extremely limited use that is included only to deal with a perceived processing difficulty with some of today’s tools.

(c) It allows authors to switch easily from one layout to the other.

Disadvantages

(a) Use of the attribute on the `Text` element may make XSLT processing more inconvenient. This limitation may be mitigated by moving the attribute to the Quotation element so it relates to the `Text` after the quotation.

12.4.4 Specific recommendations

17. It is recommended that the TC determine whether it prefers simplicity of the element markup or a rigorous separation of new line and in line elements and XSLT processing convenience as the primary design consideration. If it prefers simplicity of markup, it is recommended that it adopt Option 2. Otherwise it may adopt Option 1.

13. Title and Num elements

13.1 Title

13.1.1 Content model

```
<!ELEMENT Title (#PCDATA)>
```

Note: This is a simplified content model. Other content may be required. It is possible even that the Text element could be added to allow new line functionality within Titles. This can be considered at any time during development.

13.1.2 Purpose

The `Title` element provides a descriptive label to the content within the container to which it is attached.

The `Title` element is used by authors to assist readers to follow the structure of the narrative.

In the document outline, the `Title` element is used to populate the contents listing and to form the basis for hypertext linking in online publications.

13.2 Name selection

13.2.1 Options considered

The following options were considered:

- Title
- Heading
- Caption

The choice of name from this list is largely driven by personal familiarity.

“Title” is currently favoured but any of the other options would be acceptable.

13.2.2 Specific recommendation

18. It is recommended that the TC adopt the name “Title” for this element, unless there is a general desire to adopt another name.

13.3 Num

13.3.1 Content model

```
<!ELEMENT Num (#PCDATA)>
```

Note: This element is a placeholder only. The clause model requirements propose that component numbers should be explicitly added in the markup to facilitate transportability of documents and to support cross references within the document. The overall requirements for numbering markup are yet to be defined.

13.3.2 Purpose

The `Num` element contains citation numbers for `Item` elements. Usually, these numbers will be generated automatically but manually numbered content can be created. This is often needed to correctly represent quoted material or where the parties wish to freeze numbering of a particular document version.

It is also proposed that citation numbers are explicitly included in the markup to facilitate cross referencing within the document and transportability of documents to other applications. At present, CSS technology does not allow an application to create cross references to a number generated by CSS. It is likely this will not be available until CSS 3.0 is published. Even it if is available, there is no reason to expect that CSS will be the only rendering application used. If numbers are not written into the markup, every other application that processes the data has to correctly interpret the automatic numbering scheme applied to the document. This imposes unnecessary costs and the potential for inconsistency.

The proposed basic clause model makes no assumptions about the numbering schemes to be used or how they will be implemented. This is to be provided by the user’s supporting applications. Clearly additional attributes would be required on the `Item` element to provide numbering scheme control and to allow use of automatic or fixed numbering.

A common area where authors will require numbering control is to specify the numbering scheme for lists in the narrative content.

The way in which numbering is handled will greatly affect the exchange of XML documents between users.

Numbering requirements are to be dealt with in a subsequent stage of the development.

13.4 Name selection

13.4.1 Options considered

The following options were considered:

- Number
- Num

- No

The current selection of “Num” is chosen because “Number” is inconveniently long for screen display in a tags on view. It is not consistent with the principles set out in topic 7.5. Abbreviations are avoided elsewhere. It is possible another name should be chosen.

13.4.2 Specific recommendation

19. It is recommended that the TC invite suggestions from committee members for an alternative name once the requirements for specification of numbering are finalised and it is finally determined that numbers should be retained in element markup. Pending that decision, the element “Num” should be provisionally retained.

14. Intellectual property rights

Elkera Pty Limited, ACN 092 447 428, an Australian corporation, claims to be the owner of intellectual property rights, including copyright, in a work known as the “Elkera Topic DTD”. Parts of that work are included in this proposal, viz:

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

Peter Meyer, as managing director on behalf of Elkera Pty Limited acknowledges the terms of the OASIS Policy on Intellectual Property Rights, OASIS.IPR.3.1 and the licence thereby granted to OASIS. Subject to that licence, Elkera retains its intellectual property rights in the “Elkera Topic DTD”.

Upon request by the Technical Committee, Elkera Pty Limited will consider release of its intellectual property rights in relevant parts of the Elkera Topic DTD to the extent considered necessary by the Technical Committee to facilitate development of its schema.

15. Summary of fit to requirements

1. Markup the core structures found in documents like Attachment 1 to the Requirements

This is satisfied, as shown in Attachment 1.

2. Represent the structured hierarchy of the content

This is satisfied, as shown in Attachment 1.

3. Represent terms of benchmark contracts

To be demonstrated when the TC's benchmark contracts are collated.

4. Define clause objects as self contained objects

This is satisfied, all clause like objects are self contained.

5. Self contained markup so that a text file could provide complete contract terms

All terms of the contract recorded in the XML document can be read with a text editor, if necessary. No part of those terms is implied by other software.

6. Must avoid listed terms for element names

This is satisfied.

7. Must permit markup of contract terms without inclusion of any legal semantic markup or annotation

This is satisfied.

8. Must be as simple as practicable to facilitate user training, support and application development

The proposed model provides extreme simplicity for authors. It does not require them to know the difference between a clause, subclause or list item. They can re-organise draft content with almost no re-tagging.

The proposed model is as simple as practicable for application developers. The minimal model proposed favours simplicity for authors over simplicity for developers and resists the creation of new elements that some developers may find convenient.

The final judgment on this balance does not have to be made now. That judgment can be made after feedback is received from a wider community of potential users.

9. Must be able to re-use content in different levels of the hierarchy, without having to change names of elements

This requirement is fully satisfied. Content everywhere is tagged in the same way. Components can be readily moved to other locations in the hierarchy without the need for transformation. This provides a very high level of flexibility for automated content management and for authors.

10. Must allow clauses or other content to be incorporated into a document by reference

This requirement is not specifically addressed in the basic clause model proposal. It is likely to be facilitated by the use of only 3 elements (Item, Block and Text) for most markup.

A number of possible techniques can be employed to provide for content incorporation by reference. These can be implemented independently of the basic clause model.

11. Other listed features to be determined

These are to be addressed in later stages of the development.

16. Summary of issues & recommendations

Summary of options		PM recommendation
Recommendation 1 (topic 6.5)		
1.	Develop a structural clause model in terms of the proposal	Yes
Recommendation 2 (topic 8.9)		
2.	Defer selection of reference schema until requirements are developed	Yes
Recommendations 3 & 4(topic 10.3.2)		
3.	Adopt "Item" as the primary container name, subject to recommendations 11 & 12	Yes
4.	Adopt "Topic" as the primary container name if a separate list item element is adopted under recommendations 11 & 12	Yes
Recommendation 5, 6, 7 & 8 (topic 10.5.3)		
5.	Allow use of both the loose and tight models	Yes
6.	Designate the loose model as the exchange standard	Yes
7.	Recommend the tight model for general use	Yes
8.	Recognise exchange using the tight model by agreement of the parties.	Yes
Recommendations 9 & 10 (topic 10.6.7)		
9.	Adopt Option 1 in principle (use metadata to determine contents boundary)	Yes
10.	Leave Options 2 & 3 in the draft specification for comment by persons reviewing the model.	Yes
Recommendations 11 & 12 (topic 10.7.5)		
11.	Make a provisional choice between:	
	Option 1 – redefine Item in narrative context	Yes

Summary of options		PM recommendation
	Option 2 – ignore content model problem	Acceptable
	Option 3 – create ListItem element	No
12.	Review the Options after decision on the reference schema and related issues & inline list requirements are known.	Yes
Recommendation 13 (topic 11.3.5)		
13.	Choose element name:	
	"Block"	Yes
	"Para"	No
Recommendation 14 (topic 11.6.4)		
Consider:		
	Option 1 – Omit List element	Yes
	Option 2 – Create List element	No
14(i)	Defer final decision until requirements for managing component numbering are determined.	Yes
14(ii)	Provisionally exclude List element pending a final decision.	Yes
Recommendation 15 (topic 11.7.2)		
15.	TC to provide guidance on the 7 questions regarding inline lists.	
Recommendation 16 (topic 12.3.2)		
16.	Choose the element name from:	
	"Text"	Yes
	"TextBlock"	No
Recommendation 17 (topic 12.4.4)		
Consider:		
	Option 1 – Create a special element for trailing, inline text after quotations etc.	No
	Option 2 – Create an exception to new line behaviour for element Text.	Yes
17.	TC to select based on preference for perceived	Option 2

Summary of options		PM recommendation
	markup simplicity against rigorous separation of newline from inline elements and possible XSLT processing simplicity.	
Recommendation 18 (topic 13.2.2)		
18.	Adopt the element name “Title” or suggest an alternative.	
Recommendation 19 (topic 13.4.2)		
19.	Provisionally retain “Num” and consider alternative names for this element once numbering requirements are known.	Yes

Attachment 1 Clause model schema and XML markup example

Using DTD syntax, the basic clause model is as follows:

```
<?xml version="1.0" encoding="us-ascii"?>
<!DOCTYPE Item [
<!ELEMENT Item (Num?, Title?, (Block* | Item *))>
<!ELEMENT Block (Text | Item)+ >
<!ELEMENT Text (#PCDATA) >
<!ELEMENT Num (#PCDATA) >
<!ELEMENT Title (#PCDATA) >
]>
```

Notes:

1. If the loose or exchange model is desired, Item is defined as:

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

2. Using DTD syntax, Item cannot be re-defined in the Block context, as is proposed in the XML Schema version below.

The proposed XML Schema is set out below.

```
<?xml version="1.0"?>
<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="Block" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="Item" minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:element name="Block">
    <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="Block" 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</Title>

<Item>
<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>
     <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>
          <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>
</Item>

<Item>
<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>
</Item>

<Item>
<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>
     <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></Item>

<Item>
<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>
     <Num>&#x2022;</Num><Block><Text>black, or</Text></Block></Item>
     <Item>
     <Num>&#x2022;</Num><Block><Text>white.</Text></Block></Item>
</Block>
</Item></Item>

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