Elkera Pty Limited

XML schema

Comparison of XML schema for narrative documents

3 August 2005

By Andrew Squire and Peter Meyer

Elkera Pty Limited ABN 68 092 447 428
Suite 701, 10 Help Street, Chatswood, NSW 2067, Australia
PO Box 5280, West Chatswood, NSW 1515
Telephone +61 2 8440 6999
Facsimile +61 2 8440 6988
www.elkera.com

Contents

Comparison of XML schema for narrative documents

1 Introduction

1.1 Purpose of this document

The purpose of this document is to identify and compare the key features of four XML DTDs or schema that may be considered as candidates to model a wide range of business, legal and technical documents commonly prepared by business and government enterprises (***narrative business documents***).

Enterprises may wish to manage narrative business documents using XML to support various requirements, including single source publishing, automated content re-use and long term data management for long life cycle documents.

These sorts of documents are often created by non-technical authors who are familiar with word processing software but have no experience with XML.

The schema to be compared in this document are:

- DocBook (www.oasis-open.org);
- DITA (www.oasis-open.org);
- XHTML 2.0 (www.w3.org); and
- Elkera BNML (www.elkera.com).

The first three schema are standards or proposed standards. The Elkera BNML schema is newly released by Elkera in July 2005. The Elkera BNML schema is to be freely available under an open source licence and is not a commercial product for Elkera.

While they overlap in many respects, each of the four schema is designed with a particular focus. They each have advantages for particular document types and application requirements.

The selection of a schema is possibly the most critical decision in the establishment of an XML based content management system. Every application must be built around the schema. Once applications are developed, it is very expensive to replace the schema or make major changes to it. The schema design has major impacts on the cost of application development and on the ease with which authors can create XML content.

The aim of this document is to assist technical architects to define their application requirements and identify those schema that are likely to be suitable for more detailed assessment. Due to the complexity of the schema and the wide range of potential user needs, this comparison is not a comprehensive review of each schema. It focuses on particular features relevant to narrative business documents.

1.2 Assumed requirements

To provide a basis for the comparison, certain high level requirements are assumed:

(a) Documents are predominately text with graphics and tables that should be rendered using standard layouts for all or most documents of a particular class.

(b) Content must be created by authors from scratch using an XML editor.

(c) The schema must define a common markup language that can be used for a range of document types to minimise application development and author training effort.

(d) Metadata may be required to be added to documents and content objects to facilitate content management, publishing requirements and semantic processing needs.

(e) Documents may be revised from time to time by different authors and re-published.

(f) Documents may be published in print, on the web and often in other formats.

(g) Document components may require automatic numbering and dynamic cross references to numbered components.

(h) Content may be shared in multiple document renditions and content may be re-used in multiple documents, either by manual or automated copying of content chunks.

(i) It may be necessary to provide for collaborative authoring where parts of a document are produced by different people and assembled into a whole before publication.

1.3 Approach taken in the comparison

This document provides a two level comparative analysis of the four schema covering technical issues and the business implications of using each schema.

The technical assessment and comparison considers these characteristics:

(a) the stated purpose or function of each schema by its developers;

(b) a description of the basic structural model defined by the schema and the way it may be applied to narrative business documents;

(c) the approach taken for component numbering and internal cross references;

(d) the schema's support for content re-use in authoring applications and in automated processing systems such as document assembly systems;

(e) particular specialized markup features provided with each schema;

(f) the support for conditional output; and

(g) the approach taken by each schema for customization and specialization.

The business implications are based on an assessment of:

(a) the likely application development and support effort imposed by the schema design; and

(b) the ease with which authors might be trained and supported to use the schema consistently, assuming they use an XML editor for document authoring.

2 Overview of the schema

2.1 DocBook

DocBook was released in 1991 by the Davenport Group. The original design goal of DocBook was to enable the interchange of computer documentation. It was not originally designed as an authoring DTD. DocBook is now an OASIS Standard (www.oasis-open.org).

As DocBook was originally developed for computer documentation and it has a heavy bias toward these types of documents.

DocBook users typically use either the book or article as the root element. The article element is the DocBook element may be used for a range of narrative business documents.

2.2 DITA

The Darwin Information Typing Architecture (DITA) was originally developed by IBM. It became an OASIS Standard (www.oasis-open.org) in June 2005.

DITA was designed to support topic based content. Topic based content is information that is chunked into discrete topics, usually found on web sites and computer help systems. It was not designed with narrative documents in mind though it could be used for this purpose if required. DITA offers a sophisticated customization and specialization model that allows customizations to be compatible with applications capable of processing the base DITA.

DITA allows documents to be created using either the topic element or the map element. The topic element is used to model discrete chunks of information but topics can be arranged hierarchically. The map element is used to define the relationship among a set of resources, such as DITA topics. It provides an information framework and navigation structure for on-line content. The map element could be used to represent a hierarchy for many narrative documents.

2.3 XHTML 2.0

Extensible Hypertext Markup Language (XHTML) 2.0 is currently in development by the World Wide Web Consortium. At July 2005, it is a Working Draft and has not reached the Recommendation stage.

XHTML is an XML version of Hypertext Markup Language (HTML). XHTML 2.0 builds on XHTML 1.0 but is primarily intended for publishing documents on the web. XHTML 2.0 adds the ability to create document sections and grammatical paragraphs.

XHTML only provides a single root element html. All possible document types must be modelled using this element.

2.4 Elkera BNML

Elkera Business Narrative Markup Language (BNML) was developed by Elkera Pty Limited and released as an open source schema in July 2005. This schema is the result of over 10 years experience developing DTDs for specific purposes.

Elkera BNML is designed specifically for the creation of narrative business documents, particularly legal and general business documents. It aims to provide a simple structural markup of documents that will support high quality print and web publications, facilitate content re-use and allow authors to be trained and supported with minimal effort.

Elkera BNML provides four standard document types, document for normal business documents and articles, contract, correspondence and item for discrete, reusable chunks.

The BNML schema provides a number of specialized elements that are specifically suited to these document types. The adjunct element provides for appendices and attachments to documents, contracts and correspondence. This element can contain normal narrative content or a complete document, contract or correspondence instance.

Elkera BNML also includes the party-signature element and its components to represent provisions in contracts and correspondence where written signatures and seals must be applied.

3 Basic structural markup

3.1 Purpose of this section

The basic structural markup of narrative business documents can be broken down into three distinct patterns:

(a) the hierarchical structure used to create divisions or sections within the document;

(b) the paragraph markup; and

(c) the list markup.

Each schema defines a basic document hierarchy or structure. The markup of that structure must be consistent and logical, allowing authors to easily grasp the basic markup principles.

This section will compare the approaches taken by each schema to model these basic patterns within a document. This will include an assessment of the ability of each schema to support the consistent application of markup to meet content re-use and collaborative authoring needs. This characteristic is also a significant factor in usability for authors and in reducing application complexity.

3.2 Hierarchical structures

3.2.1 Examples of common structures

Narrative business documents usually contain a hierarchical structure, as shown in the following example:

Example 1 Numbered document hierarchy

1 First level

1.1 Second level

1.1.1 Third level

This is the paragraph text of section 1.1.1, the third hierarchical level.

2 First level

This is the paragraph text of section 2, of the first hierarchical level.

The levels in a numbered hierarchy can include numbered hierarchical structures that may or may not have a heading or title, as in the following example:

Example 2 Numbered document hierarchy with and without headings

1 First level

1.1 Second level

1.1.1 Third level

This is the text of the paragraph under the third level heading.

1.2 Second level

1.2.1 This is paragraph text of a numbered structure at the third level. This level does not have a heading or title. This kind of structure is common in legal and specification documents.

1.2.2 This is another paragraph of a numbered structure at the third level.

The levels in a hierarchy can also be unnumbered, as in this example:

Example 3 Unnumbered document hierarchy

First level

Second level

Third level

This is paragraph text at the third hierarchical level.

First level

This is the paragraph text of the first hierarchical level.

3.2.2 Common hierarchical models

There are two common approaches to modelling the hierarchical structure of narrative documents:

(a) a recursive model where the same element is used at all levels of the hierarchy by nesting; and

(b) a named level model where different elements represent each level of the hierarchy.

This comparison will focus on the recursive model. In most narrative business documents there is no meaningful distinction between the content at the different levels. Content at level 1 in one document may be re-used at level 2 or level 3 in another document. The recursive model is seen as a superior model in terms of author usability, flexibility and convenience for these kinds of document. A recursive model allows the author to restructure a document by moving document sections around without having to rename the element.

3.2.3 The optional heading or title

Example 2 shows a structure in which some numbered components at the same hierarchical level have titles and others do not. It needs to be considered whether the absence of a title on clauses 1.2.1 or 1.2.2 in Example 2 make it semantically different to clause 1.1.1 in that example. The numbering sequence to be applied to the objects is the same. The only practical difference is that, usually, the structures without a title are not included in a contents listing. In another context, another author could easily add a title to the clause.

Consistently with the point made in the previous section, it is submitted that authors of the kinds of documents under consideration would commonly regard the two structures in the examples as the same. However, the optional title does have some implications, It may be necessary for the developer to deal with inconsistent title usage in contents generation. In addition, the layout properties may be different in each situation. When the structure has a title, the content following the title might be rendered on a new line. When the structure does not have a title, the paragraph content is normally rendered inline, after the number.

3.2.4 DocBook

DocBook provides:

(a) the recursive section element; and

(b) named levels using the elements sect1, sect2, sect3, sect4 and sect5.

DocBook also has higher level structural elements such as chapter and part.

The recursive section element has the following (simplified) model:

```
(sectioninfo?,
    (title,subtitle?,titleabbrev?),
    (((itemizedlist | orderedlist | para | ...)+, section*)
    |
    section+)
)
```

The key features of the DocBook recursive model are:

(a) The section must have a title.

(b) The number (or label) of the section is stored as the section *label* attribute or derived by processing applications (numbering is further discussed in section 4).

(c) paragraphs and other content are allowed before the lower level sections.

DocBook does not provide a practical way to markup all the structures in the examples in section 3.2.1. The section element cannot be used for the numbered structures in Example 2 that do not have a title. A list might be used to create the numbered component but this does not provide for the automatic number sequence because the list cannot inherit the parent section number. Additionally, it may be necessary to distinguish such structures from conventional lists for both numbering and rendering purposes.

3.2.5 DITA

DITA only provides the recursive topic element to model hierarchical structures. The topic element has the following content model:

```
(title,
    (titlealts)?,(shortdesc)?,(prolog)?,
    body,
    (related-links)?,
    (topic)*
)
```

The key features of the DITA model are:

(a) the topic must have a title;

(b) there is no provision for a topic number. If numbering is required, this must be generated by rendering applications and cannot be stored with the topic.

(c) the body is required, though this can be empty;

(d) the recursive topic element occurs outside the body of the topic.

DITA cannot markup the numbered examples in section 3.2.1. If the document requires use of container elements without a title that themselves contain elements without a title (paragraphs), DITA does not provide for such a structure. This is a reflection that DITA is strongly oriented to online content.

The DITA requirement that a topic and any specializations of it must have a title prevents it from representing the full range of structures required in narrative business documents.

3.2.6 XHTML 2.0

XHTML 2.0 also provides both approaches to model hierarchical structures within documents:

(a) the recursive section element; and

(b) named levels using elements h1, h2, h3, h4, h5 and h6, however these are not container elements.

Key features of the XHTML 2.0 model are:

(a) The title of section can occur in any order with other child elements and is not constrained to be the first child element of the section.

(b) The section title is not required.

(c) The section can directly contain #PCDATA or any of the XHTML heading, structural or text elements in any order.

(d) As with DITA, there is no provision to store a number with the section.

XHTML 2.0 cannot markup the numbered examples in section 3.2.1,

3.2.7 Elkera BNML

Elkera BNML provides one way to model hierarchical structures using the recursive item element. The item element has the following content model:

```
(metadata?, num?, title?,
    inclusion*, (block, inclusion*)*, (item, inclusion*)*
)
```

BNML provides three different patterns for the arrangement of item and block children of item or specialist containers. These patterns are a loose model, a standard model and a tight model. The loose model allows item and block elements to occur in any order. The standard model allows a block before the first item but not otherwise. The tight model prohibits item and block at the same level. Schema designers can choose the pattern that is desired for any container. The content model shown above is the standard model.

The key features of the BNML model are:

(a) The main hierarchy of the document is defined by a recursive item element with a block element for paragraphs of narrative.

(b) The item number is stored as element content in the num element (numbering is further discussed in section 4).

(c) Both num and title are optional so the same element can be used for all the hierarchical structures described in Example 1, Example 2 and Example 3.

(d) The inclusion element is used to represent all block level components such as quotations, examples and notes that need to be shielded from the normal narrative content for the application of distinct automatic numbering or layout properties. It is also used to provide for titles and numbers on graphic objects and tables.

The Elkera BNML Schema can model all the example structures in section 3.2.1 using the recursive item structure.

3.3 Paragraphs

3.3.1 Concepts

There are two common approaches to modelling the paragraph content in narrative business documents using XML markup:

(a) a grammatical or true paragraph, where all the content of a grammatical paragraph is contained within the paragraph element including #PCDATA and other block style elements such as lists and quotations; and

(b) a loose paragraph, where the paragraph element content is not contained within the paragraph element. The paragraph element usually sits at the same level as the block level elements such as lists and quotes. This is the way paragraphs are handled by word processor software.

These concepts are best shown by example.

Example 4 A grammatical HTML paragraph

```
<p>This is a list of primary colours:
<ol>
<li>red</li>
<li>blue</li>
<li>yellow</li>
</ol>
</p>
```

Example 5 A loose HTML paragraph

```
<p>This is a list of primary colours:</p>
```

```
<ol>
<li>red</li>
<li>blue</li>
<li>yellow</li>
</ol>
```

The grammatical paragraph encapsulation of all paragraph content within a container element. This facilitates more precise rendering layout control and easier manipulation of content during editing.

It may be argued that the grammatical paragraph model is more semantically correct. In the loose model, there is no relationship to the text that introduces the list and the list itself. The grammatical paragraph model maintains the relationship between all components of a paragraph. This is a logical concept for authors to grasp. The container based approach to the paragraph makes it easier for application developers to manipulate content where content reuse is a priority.

3.3.2 DocBook

DocBook offers both the grammatical paragraph and the loose paragraph models. Block level content and other paragraph components, such as lists, can occur directly inside the section element or it can occur within one of two paragraph elements:

(a) para – this is a standard grammatical paragraph; or

(b) formalpara – this is a standard grammatical paragraph with a heading.

Docbook also offers the simpara element as a simple paragraph that does not allow block style content to occur within it.

3.3.3 DITA

DITA offers both the grammatical paragraph and the loose paragraph model. The loose model allows block level content and other paragraph components to occur within the body element and the grammatical paragraph uses the p element.

3.3.4 XHTML 2.0

XHTML 2.0 works in the same way as DITA. Its loose model allows block level content and other paragraph content to occur within the body or section element. It also provides a grammatical paragraph model using the p element.

3.3.5 Elkera BNML

Elkera BNML provides a grammatical paragraph element called block. It does not permit the loose paragraph model.

3.4 Lists

3.4.1 Concepts

Broadly, a list is a catalogue of things belonging to a class. In narrative content it can be difficult to determine which parts of the narrative are lists and which are, say, clauses or subclauses. Very often, the only distinction is between the numbering scheme desired by the author. If a schema defines a list structure, it is necessary to apply that structure in a consistent, useful way to achieve rendering and processing objectives.

In narrative business documents, lists are commonly:

(a) numbered;

(b) unnumbered; or

(c) bulleted.

Some schema provide a number of other specialist lists for creating different types of lists such as definition lists and variables lists. These types of lists will not be considered in this document.

Some schema make a distinction between a numbered list (ordered list) and a bulleted list (unordered list). In narrative business documents, lists are always reproduced in the order that was intended by the document author, whether numbered or bulleted. The distinction between ordered and unordered lists is not a useful distinction in these types of documents. In practice, the only difference between the following two examples is the numbering style.

Example 6 A numbered list

The simple content of a list:

(a) text of a list item;

(b) text of another list item.

Example 7 A bulleted list

The simple content of a list:

- text of a list item;
- text of another list item.

Schema that require the numbered list to be an ordered list and the bulleted list to be an unordered list force the author to assign a vague semantic meaning to achieve a desired numbering style. This is also a problem for the author when they wish to change one list to another.

Some schema imply the type of numbers used for numbering a list. It may be important for a particular application that an author can control the types of numbers used for numbering a list. It may also be important that an author can control the index used for the first item in the list. This can be used to allow list numbering to continue from a previous list, if necessary, or to create list fragments.

3.4.2 DocBook

DocBook provides three different elements to markup lists:

(a) orderedlist – used for a numbered lists;

(b) simplelist – used for unnumbered lists; and

(c) itemizedlist – used for bulleted lists.

The orderedlist and simplelist can contain one or more listitem elements. The simplelist can contain one or more member elements.

The orderedlist provides the author with control over:

(a) the type of list item number that is assigned to list items;

(b) whether list item numbering should continue from the previous list;

(c) whether the list item number should inherit the parent list number (to achieve hierarchical style numbering of lists).

The "mark" or bullet for the items in an itemizedlist is normally determined by processing applications. The itemizedlist provides the *mark* attribute to control the type of bullet but specific values must be added by a customization layer to enable this functionality.

The listitem element allows the author override the number for the particular item.

3.4.3 DITA

DITA similarly provides three different elements for marking up these common lists:

(a) ol (ordered list) – used for numbered lists;

(b) sl (simple list) – used for unnumbered lists; and

(c) ul (unordered list) – used for bulleted lists.

Both ol and ul can contain one or more li (list item) elements. The sl can contain one or more sli (simple list item) elements. No control is given to the author to control list item numbering.

3.4.4 XHTML 2.0

XHTML provides the ol and ul lists. These have the same functionality as these elements in DITA.

3.4.5 Elkera BNML

Elkera BNML does not have a distinct list element. The item element is used as the list item element. This is the same element that is used for the document hierarchical structure, as described in section 3.2 (redefined to prevent directly recursive item elements). An item is considered to be a list item when it occurs within a block (grammatical paragraph).

Attributes on the block element give the author control over how the list items are numbered within a block. This can be either numbered, bulleted, manually numbered or unnumbered. The item element allows the author to specify a number restart index for the item so authors can control the first number in a list.

3.5 Do the schema provide for consistent markup?

3.5.1 Concepts

The consistency of markup within a set of documents has implications for both authors and developers. It is an issue that is often overlooked when evaluating a schema, particularly where it is necessary to capture legacy data with a poor structure.

A schema may provide many different ways to mark up the same content. Authors may use the markup inconsistently within a single document. Different users will apply different markup in similar situations. An author's view of the best way to mark something up may change over time.

Application developers may make assumptions about how the markup should work and base application behaviour on these assumptions. This may lead to unexpected or inconsistent behaviour by the application. Some rendering applications are style based applications, operating in a way that is consistent with word processor paragraph styles. Consistency of markup produces a better result in these applications because fewer contexts need to be considered for styling.

Schema that provide more than one way of marking something up also have implications for authors and author training.

This section considers how each schema promotes the consistent use of markup for the three basic patterns in structural markup discussed in previous sections.

3.5.2 DocBook

DocBook provides both the named level and recursive models for representing the document hierarchy. Ideally an organization would disable one of the competing hierarchical models and only provide one model, depending on the needs of the organization. If this is not done the same underlying content structure may be represented quite differently in different documents and even within the same document.

The DocBook schema allows a paragraph to have a heading using the formalpara element. This could be used instead of a section for leaf nodes in the narrative. It seems highly likely this will occur inconsistently and that, frequently, such differences will be of no useful semantic significance.

There are three different paragraph elements provided in DocBook, para, formalpara and simpara. The section element can contain most of the block style elements that are found inside the para and formalpara elements. This allows authors to create these structures in different ways by mixing grammatical and loose paragraph models. It would not be unexpected if different authors used the loose paragraph model or the grammatical paragraph model or even a combination of both models in the same document.

The markup of structures at the list level could also be inconsistent. Both the itemizedlist and the orderedlist allow a large number of block level elements to occur before the first list item, including para. This allows authors to place the introductory text and other objects within the list element as well as outside the list element.

Based on this partial analysis, it is clear that the markup of basic hierarchical structures using DocBook may be quite inconsistent between authors or between documents created by the same author. It might be possible that in some document types these differences could be meaningful. However, in common technical, legal and business documents, this is extremely unlikely. It will not be possible for processing applications or users to determine if these differences represent a subtle, yet real difference in structure or are merely arbitrary variations that, somehow, must be resolved into a common layout by processing applications.

3.5.3 DITA

DITA only provides the topic element for marking up the main hierarchical structures. The markup of those structures using DITA should be more consistent than DocBook.

The markup of paragraph and list content in DITA is likely to be very inconsistent. The body of DITA places very little constraint on how particular content is marked up and is very similar to XHTML 2.0. This would result in a mixture of grammatical and loose paragraphs, leading to poor layout consistency in rendered output, particularly in print where greater precision is usually required.

3.5.4 XHTML 2.0

The markup of hierarchical structures using XHTML may be very inconsistent because authors can use both the recursive section model and the named level model and mix these in a single document.

The markup of paragraph and list content may also be very inconsistent. The body of XHTML is very loose and places very little constraint on how particular content is marked up. This would result in a mixture of grammatical paragraphs, loose paragraphs and text data inside the section element.

3.5.5 Elkera BNML

Using BNML, the item element is the only element provided for hierarchical markup. It is possible that the inclusion element could be abused in some situations because it too is a recursive element. Its layout properties will usually be so distinct that such abuse will be very limited.

The markup of paragraph and list content ought to be highly consistent because there is only one way to markup this content.

4 Component numbering and cross references

4.1 Introduction

As discussed in section 3.2, most legal and many narrative business documents have component numbering. Component numbering is typically used for complex documents such as legal and technical documents that are to be printed. Component numbering, except for lists, is less common for web based documents. There are three related issues that will be addressed in this section:

(a) controlling automatic component numbering;

(b) implying or storing component numbers; and

(c) managing document cross references to numbered components.

This section will compare the basic approaches used in each schema.

4.2 Controlling automatic component numbering

4.2.1 Concepts

Component numbering is normally applied to the document hierarchy (outline), to lists, or to other structures within the document such as tables and figures.

Control of component numbering is important where documents use complex component numbering or require different parts of the document to be numbered in different ways. In these types of documents, the author may require explicit control over the numbering to activate different numbering styles or to restart numbering in certain contexts.

Control of component numbering may also be important in content reuse applications where existing numbers may be preserved (e.g. quoted content) or regenerated, depending on the reuse requirements.

4.2.2 DocBook

DocBook provides several elements that can be numbered. These include section, appendix, table and figure.

DocBook: The definitive guide states that a section "may" be numbered, an appendix is typically lettered. No instruction is given for table or figure.

DocBook provides no explicit way to control whether these elements are automatically numbered or how they are numbered by a processing application, except the ability to specify a manual number for a particular element. Numbering must be implied from the document type or the context of the component in the document.

4.2.3 DITA

The DITA specification makes no mention of topic numbering. Topic numbering could be implied from topic depth if numbering is required, but it is not part of standard DITA. There is no explicit storage mechanism for numbering.

4.2.4 XHTML 2.0

The XHTML 2.0 specification makes no mention of section numbering. Section numbering could be implied by processing applications from section depth if numbering is required, but it is not part of standard XHTML.

4.2.5 Elkera BNML

Elkera BNML provides three elements which can be numbered: item, inclusion and adjunct (the equivalent of DocBook appendix). Elkera BNML provides a number of ways to control numbering within the document. The main control is on the root level element of each of the Elkera BNML document types. These elements have two attributes that control numbering:

(a) *number-outline* – this attribute allows the author to control the style of numbering that is used to number the document outline. Commonly, this may be a named numbering style, manual numbering or no numbering; and

(b) *number-disable* – this attribute allows the author to disable all automatic numbering within the document, including list numbering.

Elkera BNML treats outline structure numbering and list numbering separately so that, for example, manual numbering could be applied to the outline structure but automatic numbering applied to lists.

The adjunct element also has the *number-outline* and *number-disable* attributes to control the numbering of content within the adjunct independently from the rest of the document. In addition to these attributes, the adjunct has attributes to control the numbering of the adjunct itself and an attribute to specify a number restart index for the adjunct.

The inclusion element can be used to number objects such as tables, figures and examples, according to the value of the *class* attribute on the inclusion element.

The number of an item element is controlled by its parent element or another ancestor element, including the root element. These are regarded as shield elements. An application may restart numbering sequences from 1 in each shield element.

4.3 Generation and storage of component numbers

4.3.1 Concepts

There are two common approaches to component numbering within XML documents:

(a) the component numbers are written into designated markup; or

(b) the component numbers are not explicitly stored but are calculated and implied when rendering the XML document.

This is an important issue for processing applications. When component numbers are implied by applications, numbering functionality may need to be reproduced in multiple applications. It would not be unusual for the numbering functionality to be required in three applications, including the XML editor display, the print output rendering and the web output rendering. If these applications cannot share the numbering code, developers may need to maintain separate numbering applications, leading to redundant development and maintenance plus inconsistency between applications. This problem increases when documents are exchanged with other organizations.

These problems are avoided when component numbers are written directly into the XML markup. Application developers should only need to create the numbering behaviour once.

The explicit numbering approach also facilitates cross references within documents, as described in section 4.4.

The explicit numbering method may be preferable when component numbering is significant to the application and numbers must be accurately reproduced from rendition to rendition.

One of the problems with component numbering is that there is no standard way to represent numbering styles for the outline or lists. Similarly, off the shelf tools to generate component numbers are not widespread.

This section looks at the approaches used by DocBook and Elkera BNML only. DITA and XHTML do not support explicit component numbering.

4.3.2 DocBook

The default position is that component numbers should be implied by the processing system and not included in the markup. Numbered elements such as section have a *label* attribute. This is used to store a number when processing systems are incapable of generating the number for a section. If the *label* attribute is specified, the content of the attribute must be used as the section number.

Listitem elements do not have a label attribute and cannot be numbered in the same way as the section element, if numbers are handled explicitly.

4.3.3 Elkera BNML

In Elkera BNML, numbers may be stored in the child num element of all numbered elements. Such functionality assumes the availability of an editing application that can generate automatic numbers and write these into the markup during the authoring process. If this is not available, an application can imply numbers with BNML just as with any of the other schema.

4.4 Component cross references

4.4.1 Concepts

Many narrative business documents include cross references to other components of the document, similar to "See section 4.4" or "in clause 3.1.1 (a)". Basic features required of applications in support of cross reference functionality in narrative business documents include:

- The reference should be able to include multiple components of a target number ("clause 3.1.1 (a)").
- The reference should permit the reference information, such as the target number or title, to be displayed in line for author convenience.
- It should be possible for an application to update the content of the reference when the target information changes as it is re-numbered or re-located within the document, so the information is accurate for the author and when the document is rendered.
- It should not be necessary for multiple rendering applications to have to calculate cross references.

As explained in connection with automatic numbering, the way in which component numbers are stored will affect the way in which cross reference functionality is enabled. If the editor calculates component numbers and writes them into the markup, they will be available to a cross reference utility. However, if the numbers are not written into the text, it will be necessary for a separate application to calculate them for display purposes.

4.4.2 DocBook

DocBook provides the xref element to enable cross reference functionality. The cross reference text can be generated in three different ways depending on how the xref attributes are used:

(a) the author can specify the identifier of a target element whose content is taken for the cross reference text;

(b) the author can specify the identifier of a target element with an *XRefLabel* attribute which provides the cross reference information; and

(c) the author can specify a style on the xref element for a custom cross reference processing approach.

In DocBook, the xref element is an empty element. It does not provide any facility to store the calculated value of the cross reference. The cross reference text must be generated by each application that renders the output.

If the target of the cross references uses implied numbering (see section 4.3), the author or the XML editor application must calculate the target number and write it into the *XRefLabel* attribute on the target element, if the cross reference is to be displayed in the text. This appears to be a cumbersome requirement that would not be easily implemented for dynamic display in an application.

DocBook permits compound references through the ulink element which may contain multiple xref elements.

4.4.3 DITA

DITA provides the xref element to enable cross reference functionality. The specification states that if the xref element is empty, the application "may" generate cross reference text from the destination object. Once the element is populated with data, it appears it cannot later be updated. No other mechanisms are provided to control this functionality.

The model provided by DITA does not appear to be suitable for the kinds of cross references found in narrative business documents.

4.4.4 XHTML 2.0

In XHTML 2.0, any element may reference another element. However, there is no defined mechanism to generate cross reference information from target elements for display as internal cross references. XHTML 2.0 does not appear to contemplate this requirement.

4.4.5 Elkera BNML

Elkera BNML provides the reference and autovalue elements to enable this functionality. The cross reference text can only be generated one way. The autovalue *href* attribute contains the id of the target element. The autovalue *display* attribute contains an XPath statement from the target element to the location of text to be used as the cross reference text.

For example, to provide the functionality described in section 4.4.2 (b), the autovalue would be:

```
<autovalue href="somesectionid" display="@XRefLabel"></autovalue>
```

The autovalue element can contain #PCDATA. The intention is that a processing application populates the content of the element with the cross reference display value so that rendering applications do not have to reproduce the cross reference behaviour.

A reference may contain multiple autovalue elements and text to provide for compound references with citation wording. The autovalue can access any information stored in a target attribute value or element.

5 Other markup issues

5.1 Specialized container elements

5.1.1 Concepts

All the schema provide other container elements in addition to the basic structural markup elements discussed in section 3.

Some schema come from a particular domain and may have a significant number of specialized elements to cater to that domain. When used in their original domain they may be an ideal choice for an application. When used in other contexts, domain specific elements need to be removed from the schema so the author is not presented with unnecessary element choices.

5.1.2 DocBook

DocBook provides a very large number of specialized container elements. The section element has 59 specialized containers and the para has 139. Many of the elements found in section are repeated within para. Because DocBook was originally designed for computer manuals, there is a wide variety of elements specifically intended for computer hardware and software documentation.

5.1.3 DITA

DITA provides a smaller number of specialized container elements. The body element has 22 specialized container elements and the p has 54. Once again, many of the elements found in body are repeated within p. Like DocBook, DITA has also come from the technical documentation domain and has a significant number of specialized container elements that are specific to that domain.

5.1.4 XHTML 2.0

XHTML provides a small number of specialized container elements. The section element has 32 specialized containers and the p has 20 (not including XForms controls). XHTML does not have the technical orientation of DocBook or DITA. Because XHTML has its basis in HTML, specialized container elements are normally applied to achieve particular formatting rather than to define a generic structure.

5.1.5 Elkera BNML

Elkera BNML provides a very small number of specialized container elements. The item element has 1 specialized container element (apart from block and the recursive item), block and text have a combined 21 elements. Like XHTML, BNML does not have the technical orientation of DocBook or DITA. This reduces the number of options significantly.

In Elkera BNML, the inclusion element is a generic container element for content that is distinct from the normal narrative. This element was designed to replace a variety of the specialized container elements that are found in the other schema. Specialization of the inclusion is achieved using a class attribute which allows different styles to be applied when rendering the element.

The BNML Schema also provides the adjunct element (similar to DocBook appendix) that is used for schedules, attachments and annexures to documents. The adjunct may include a complete document or item and block content.

5.2 Metadata

5.2.1 Concepts

The basic markup approach taken by each schema may be described as a generic structural markup. Such markup provides very limited semantic information about the content of a document. It may define clauses in a contract but it says nothing about the function of particular clauses. This definition of structural components serves common publishing needs. It also provides a skeleton that can be decorated with additional semantic information using metadata where required by an application. Such metadata may be required at the document or component level.

Metadata can be used by application developers for a variety of purposes such as searching, producing document cover pages, document management and version identification.

Metadata is also highly organizational and application specific. Some organizations may need to meet government or industry regulations for metadata. It is rare for a schema to provide all necessary metadata elements.

There are two common approaches to specification of metadata:

(a) use specific named elements to store metadata; or

(b) use generic elements to store metadata.

Some organizations may wish to use a particular metadata language such as RDF.

Specific named metadata elements allow the schema designer to better control the type of metadata that is captured for a particular document. This can be done by making certain metadata elements required and validating the content of metadata when using XML schema or similar schema languages.

Generic metadata allows the capture of arbitrary metadata and allows easy extension of metadata. However the schema designer cannot enforce its capture or validate it as easily as with named metadata.

Due to the wide range of possible approaches and application requirements, a schema should not be prescriptive about the way metadata is managed.

This section looks at the approaches used by the different schema to capture metadata.

5.2.2 DocBook

DocBook provides the articleinfo element for document level metadata and the sectioninfo for section level metadata. Both these elements contain a large number (64) of specific named metadata elements. This provides a "bucket" of options that can be used by organizations for metadata. There is no provision for generic metadata.

The schema must be customized to add organizational specific metadata.

5.2.3 DITA

DITA provides the prolog element to contain topic metadata. This element contains a small number of specific named elements:

author, source, publisher, copyright, critdates, permissions, metadata, resourceid

It also provides the metadata element which contains:

audience, category, keywords, prodinfo, othermeta

Additional generic metadata is provided by othermeta element. This approach only allows the creation of name, value pairs. Record based metadata structures cannot be created.

5.2.4 XHTML 2.0

XHTML 2.0 provides the generic meta element for representing all metadata. The meta element has a *property* attribute which contains the name of the metadata property and the content of the meta element is the value of the property. The content of meta can contain inline elements allowing metadata to be formatted. However, as with DITA, this approach only allows the creation of name, value pairs. Record based metadata structures cannot be created.

5.2.5 Elkera BNML

Elkera BNML provides a metadata element at the document level and for the item, inclusion and adjunct elements. Elkera BNML provides a subset of standard Dublin core metadata elements that can be used within the metadata element. However, it is expected that users of the BNML schema will define their own application specific metadata. The BNML Standard schema permits any content within metadata for interchange purposes.

6 Conditional output

6.1 Concepts

Conditional output is often required for single source publishing. Output that may be appropriate for print publishing may not be appropriate for web publishing. Another common use is for international publishing requirements to cater for variations in spelling or other phrases.

This section looks at the features provided by each schema for conditional output.

6.2 DocBook

DocBook provides the *condition* attribute on all elements to enable conditional processing and output. This is part of the common attributes so it occurs on all elements. The phrase element can be used for inline conditional text. However, the semantics of conditional processing are left to processing applications.

6.3 DITA

DITA provides the *audience* attribute on all elements to enable conditional processing and output. The ph (phrase) element is used for inline conditional text. Like DocBook, the semantics of conditional processing are left to processing applications.

6.4 XHTML 2.0

XHTML makes no explicit provision for conditional output.

6.5 Elkera BNML

Conditional output is an option that can be activated within Elkera BNML. When activated, BNML provides the *condition* attribute on the item and block. The conditional element is used for inline conditional text. Like DocBook, the semantics of conditional processing are left to processing applications.

7 Content re-use

7.1 Concepts

XML is ideally suited to applications that require content re-use. If content re-use is a requirement for a particular application, provision for re-use must be made in the schema.

At one level, content re-use only requires a mechanism to identify to a processing application an external component that must be incorporated into a particular place in the document. In practice, other aspects of the schema design are important with content such as narrative business documents. The schema must define content components that can be processed as discrete units of information. It has already been noted that in narrative business documents, content components may be re-used at different levels in the document hierarchy, depending on the overall context. Thus the ability to insert components at arbitrary levels may be important. Such components need to be marked up in a predictable and consistent way for this to work effectively.

This section looks at the specific features provided by each schema to facilitate content reuse.

7.2 DocBook

DocBook makes no explicit provision for content re-use. Content re-use must be added as a customization.

7.3 DITA

DITA provides two approaches to content re-use. Firstly, virtual documents can be defined using map with topicref elements that point to topic elements.

The second approach is that all DITA elements provide a *conref* attribute for content re-use applications. This attribute can contain the id of a target element. The content of the target element is rendered instead of the source element. The only qualifier for this functionality is that the source and target elements must be the same element. For example, only a topic element can point to a topic.

7.4 XHTML 2.0

All XHTML elements provide a *src* attribute for content re-use applications. This attribute can contain a URI. The content of the URI is rendered instead of the source element. The *srctype* indicates the MIME type of resource. This is the same method that is used for including graphics and scripts in to XHTML documents.

7.5 Elkera BNML

Elkera BNML provides an option to use XInclude (http://www.w3.org/TR/xinclude/) for content re-use. By default, content re-use is only provided for item and for block in contexts where the loose structure model is used. Customization is required to allow other elements to be re-used.

8 Customization and specialization

8.1 Concepts

Off-the-shelf schema will normally require some level of customization before they are used for an application. It is important to select a schema that is easily customized. Customization usually takes one of the following approaches:

(a) The schema functionality is built up from a core set of elements by adding elements, attributes and attribute value enumerations until the desired level of functionality is reached; or

(b) The schema functionality is reduced from a large pool of elements by removing elements and attributes that are not applicable to the application. Elements, attributes and attribute value enumerations may be added to cover functionality not provided by the base element pool.

Any schema can be modified to overcome perceived limitations. It is hoped this document will provide information that will enable persons evaluating schema to work out which of the four schema is the closest fit to their needs. When deciding whether to modify an existing schema, there are two important considerations. Firstly, will the result be confusing to people familiar with the original schema? They may think they understand it but find it is now something different. Secondly, will the changes retain the benefit of existing tool support? If the changes invalidate available processing tools, little advantage may be obtained.

This section explores the customization approaches used by each schema.

8.2 DocBook

DocBook is a very large schema with a large number of elements and loose content models. It is likely that most authors of narrative business documents would have difficulty using standard DocBook without extensive customization. Customization and specialization is first achieved by reducing the large pool of elements.

DocBook uses the traditional DTD method of customization and specialization which builds up the DTD using parameter entities. Parameter entities can be used to alter element content models and attributes. It has been layered in a way that minimises the impact of new versions on a customization.

There are three classes of customization:

(a) Subset - the customization is a strict subset of DocBook;

(b) Extension - the customization is not a strict subset of DocBook (elements, attributes, attribute values have been added)

(c) Variant - this is used when an organization doesn't want to use Subset or Extension.

When a customization is made the schema can no longer be called DocBook.

Specialization of the DocBook requires a detailed understanding of:

(a) the structure and layout of the DocBook;

(b) the DocBook elements; and

(c) the way the parameter entities are structured and used.

8.3 DITA

DITA has a smaller pool of elements and is generally customized by building up the base functionality. DITA allows for a form of specialization and customization that is different from DocBook. The topic element and other elements model a particular pattern in a document. These patterns can be reused by creating new elements from a source element. A new element must have the same pattern as the source DITA element (a content model that is the same or a more restrictive content model than the source). The model cannot be "loosened". For example, if a specialization was created from a topic, the specialization must have a required heading.

The effect of the DITA approach is to provide access to default processing rules for topic content, thereby minimising the amount of application development when new specializations are created.

There are two types of specialization in DITA:

(a) topic specialization - new topic types are created. This is achieved by renaming the topic element and may include renaming child elements to suit the new topic type. The topic ancestry is recorded in attributes to allow processing applications to fall back on generic processing models.

(b) domain specialization - a new language or vocabulary is added for a particular domain (eg. programming, user interface, hardware, etc). New elements are created that can occur within para, body and section, etc. These are made available to all topic and specialized topics within the DTD.

Specialization of DITA requires an understanding of the two types of specialization and how the specialization ancestry is represented in the DTD.

8.4 XHTML 2.0

XHTML 2.0 does not explicitly support customization. Users can customize the schema for specific uses but it is no longer XHTML.

8.5 Elkera BNML

Elkera BNML has a smaller pool of elements than DITA and is customized by building up the base functionality. The core elements provided by Elkera BNML model a small number of common patterns that can occur throughout narrative business documents. These patterns can be used to create most document content required by authors of narrative business documents. This approach is intended to permit re-use of processing applications based on the base patterns. In this respect it has a similar aim to DITA but lacks the flexibility of DITA to create new element names based on those patterns.

It is not expected that Elkera BNML (BNML-standard) can be used without customization. The schema provides several levels of customization. The minimum customization an application developer must undertake is to define organizational specific metadata and class attribute value enumerations for a number of generic element types.

The application developer can then decide whether to create a subset or a variant. When creating a subset the developer is only allowed to add metadata, add attribute value enumerations to existing attributes and remove optional elements or attributes. This allows BNML documents to be exchanged with other organizations using the BNML Standard schema.

If the application developer wants to make further changes, such as to add new elements or document types, a variant of BNML must be created. The only restriction on a variant is that the core patterns for item and block must be maintained. If these are changed, the application cannot be part of the BNML family of schema.

9 Application development effort

9.1 Concepts

Application development effort is an assessment of the time and effort required to develop applications using the schema. Other things being equal, schema that require less effort are likely to find a bigger market than those that require a lot of development effort.

An application developer should be able to find developed applications and support within a community of schema users. Established schema are likely to have built up a significant resource base over time. The value of this resource will depend on the applicability of the schema to the enterprise needs. The availability of already developed applications is not necessarily related to the design of the schema and is not further considered in this comparison.

9.2 DocBook

The design of DocBook impacts on application development effort in two ways. Firstly, it provides a very large number of elements for which it is difficult to define precise usage. Secondly, the content models are loose, allowing a very large number of element combinations. The effect is that developers may create applications on the assumption of particular usage of particular elements, only to find that unexpected results through inconsistent usage or the occurrence of unhandled contexts. In order to develop an application that will take account of all possibilities provided by the schema, a massive development effort is required. This makes DocBook applications brittle and unreliable. Except where appropriate developed applications are already available from the DocBook developer community, DocBook applications are very expensive to develop.

9.3 DITA

DITA has fewer elements and at the document structure level, it provides a slightly stricter model than DocBook. However, the content of a topic uses very loose content models that introduce application development complexity similar to DocBook in some areas.

9.4 XHTML 2.0

XHTML 2.0 has fewer elements than DocBook but the extreme looseness of the content models will make XHTML applications unreliable and expensive to maintain, particularly if high quality rendering is required.

9.5 Elkera BNML

Elkera BNML uses a very small set of elements and tight content models compared to the other schema. In this respect, BNML ought to permit lower cost and more reliable application development.

9.6 Conclusions

All the schema operate as recursive, generic structural schema. All such schema impose demands on application developers to process information based on its hierarchical context rather than its explicit naming.

DocBook and XHTML 2.0 are likely to be the most difficult schema with which to create low cost, reliable applications due to the large number of elements (particularly DocBook) or very loose content models. Elkera BNML ought to be the easiest schema with which to create low cost, reliable applications.

10 User training and support effort

10.1 Concepts

A key feature of narrative business documents is that they are created by humans and they are likely to require ongoing revision. It is assumed that the objective is for authors to create these documents using an XML editor application.

To date, XML authoring has not caught on widely outside specialized content creation units. Experience indicates that many organizations that have tried it, have difficulty training and supporting authors who have been raised to use common word processing software. The change to XML content authoring involves authors learning new concepts.

If authors are to take up XML authoring, the two critical objectives for the system are to:

- minimise the initial training effort so that a new system can be introduced with minimum disruption and cost; and
- obtain consistent markup that does not require costly data rectification in later stages of the publishing workflow.

The ease with which application developers can create an easy to use authoring interface will substantially depend on the design of the schema. Empirical evidence suggests that it is easier to train and support authors to use an XML editing application if:

- the new concepts that authors need to learn are clearly defined and few in number; and
- common authoring processes can be simplified so that authors do not have to choose from a list of elements and find the correct location to insert them while also trying to write the narrative.

Both objectives can be achieved if the schema avoids asking authors to make semantic distinctions that cannot be applied consistently and that serve no real purpose in the application.

10.2 DocBook

As discussed in earlier sections, DocBook provides a very large number of elements that can be arranged in a variety of patterns. To use DocBook effectively, authors may need to understand which of those elements are important to them, which are not and which of several approaches to paragraph and list markup are applicable to particular circumstances. It is difficult to reduce this to a few simple principles for authors. Similarly, it is difficult for an application developer to package this in a simple form for authors. To be effective, an authoring application may need to impose a very tightly restricted version of the DocBook schema.

10.3 DITA

DITA appears to suffer from many of the same problems as DocBook, particularly at the paragraph level. The topic structure may suit some kinds of documents but it is not a natural fit for many business documents.

10.4 XHTML 2.0

XHTML 2.0 suffers from many of the same problems as DocBook. While it has fewer elements and the basic principles are simple, it may be difficult for authors to understand how to consistently markup content within a section where almost anything is permissible.

10.5 Elkera BNML

The core of Elkera BNML involves just three important elements, item, block and text. The function of each can be quickly explained, along with the distinction between clause item structures and list item structures. Once these concepts are understood, an author can reliably create the basic patterns in narrative documents.

The limited element options and the strict content models should assist application developers to provide an interface that lets authors easily create a hierarchical structure from item and block elements.

10.6 Conclusions

It is possible to markup basic narrative structures using just a few elements from each of the four schema. DocBook, DITA and XHTML 2.0 each require authors to make element selections at the paragraph level that are unnecessary and confusing. Due to the large number of choices offered, those schema do not define simple patterns that can be easily explained to new authors and consistently applied by them.