D. Highest Level Content Additions

1. Root element

2. top level contract structures

Peter Meyer and Jason Harrop propose "named containers", as follows:

```
instrument
(h*, extra?,( p* | (datearea, parties) ), background?,
        operative, (signatures | extras)* )
```

Notes:

1. h*, so that it is possible to have a heading and a sub-heading. Which is which, and we can't ensure that author will not, for instance split separate lines of a title into <h>s. My proposal (look in Dublin Core elements) completely avoids this problem.

2. The single <extra> at the beginning of the instrument content model is for the reference schedule sometimes found at the front of a contract. This element apparently may not contain "operative" contract language.. Does that mean that it's not "signed", ie accepted, by the offeree, that it has no force in the contract? Please provide a succint, functional definition for this element. I do provide a crisp definition for my use of a for non-clause material:

   "A encapsulates all flowed material within the contract that is not a clause. A clause is material that is primarily numbered, but may optionally be titled. Non-clause material is primarily titled, but may be optionally numbered."

3. The alternative [(p* | (datearea, parties))] is necessary to accommodate differences between the USA and UK approaches to date/parties. **US** users may simplify the structure to [instrument (h*, extra?, p*, background?, operative, (signatures | extras)*)] and put the content in a <p>, for example:

   > **<p>**This License Agreement is made this 12th day of January 2001, by and between the National Association of Realtors, an Illinois not-for-profit
   > corporation which has its principal place of business at 430 North Michigan Avenue, Chicago, Illinois 60611 ("NAR") and Larry Liquour ("Licensee")**.</p**> This example doesn't show the proposed "datearea" or "parties" elements both of which seem warranted (ie "12th…" and "NAR" and "L Liquor")

   UK/ANZ users may simplify it to:
   [instrument (h*, extra?, datearea, parties, background?, operative, (signatures | extras)*)>] with:
   datearea (h?, p*)
   parties (h?, (party | p)*)
   party (I | #PCDATA)*

   See the sample in appendix 3. Note that <p> is included in <parties> for the case where "BETWEEN" and "AND" appear on lines by themselves. What's the issue with an author using the <l> element in this situation? In my proposal, this "issue" of connector words is on a separate line or not is totally irrelevant…. Parties are found in the contract's Dublin Core metadata.

As shown in appendix 2, the presentation options vary greatly in the way that the connecting words "BETWEEN" and "AND" are handled. A **(**complete**)** structural markup model **(**might require**)** one or more specific elements for these words so that a rendering application could create all the possible layout arrangements. Alternatively, tables could be used. Under my proposal, authors can even use tables to layout their non-clause content – to achieve the "look" they want. There is no need to worry about "all possible layout arrangements".

This model is hung up on the desire to parse the names of parties out of the contract language, and therefore needs to worry about all the myriad ways that the contract language can be drafted, and formatted. Under my proposal, one simply looks in the contract's associated metadata description.

Jason Harrop and Peter Meyer consider that it is not desirable to try to define a generic element for these components because of their limited function. Rather, they propose that these connecting words can be generated in layouts by the rendering application, if so desired by the user. Under such an approach, the parties element might only contain a series of party elements. The words "BETWEEN" and "AND" or similar words would not be present in the markup (bending at least the rule the no contract language may be generated).

To illustrate how the complete model might look, possible inline markup for the party names and addresses is sketched out in that appendix. Party related markup is different from most other semantic markup, in that it is immediately and obviously useful in an authoring environment. There's no debate about the need to mark up party names, the question is where… please justify why party-related information should be marked up within the contract's language, whereas other 'semantic' information is not. Words like "obvious" are not useful in a technical specification.

4. <background> is for the recitals. Its content model is: [background (nr?, h?, section*)]. The recitals are made up of sections. There are typically no opening paragraphs between the recitals heading and the recitals. The implication is that this material is not "operative" for the contract, when it most surely is. Insofar as the element name, I have no idea what a "contract background" *is* – again, please give a crisp definition. Under my proposal, there is no need for such ambiguity – the element is well-defined.

5. The content model for the operative clauses allows for one or more opening paragraphs before the clauses proper: [operative (nr?,h?,p*,section*)]. There is no justification provided for this element, nor any definition. I don't see the need to specifically identify "operative clauses" – under my proposal, they are simply the <section> elements within the <instrument> element. Poor to have extraneous elements.

6. <extras> is for schedules/annexures etc.

   The content model is simply: [extras (h?, extra*)]. The extras container carries an attribute which says whether its <extra> elements are (schedules | attachments | annexures | appendices | exhibits**) .** "schedules" is proposed as the default.

   Presumably the justification for using the antiquated device of a container like this is that it makes "numbering appendices much easier"…. stylesheets can be easily written without reliance on a container element. And why is <extra> not a recursive element?

Under my proposal (use , which can be recursive) there is no need for any additional “container” element. Also, I have no clue what the functional differences are between the 5 types of “extras” so far identified.

The content model for extra is: [extra (nr?, h?, p*, (section* | instrument*))]. By means of <instrument>, a document can be attached in a schedule.

Using as I propose, a legal instrument can quite naturally be embedded as an attachment. I note that ALL the content of <extra> is optional – troubling me because (a) empty elements are being said to be OK in the document (b) there is no definition of “extra” that is being enforced by the content model….. under my proposal, the <h> element is mandatory for all elements… as my definition says.

7. The model allows the signatures to come before the schedules and annexures, after them, or between them. This is inadequate – signatures can appear any where in the document, and can appear multiple times. They can appear on the bottom of each page, and can appear adjacent to specific clauses. There are many examples of multiple signature areas and signature “crosses”.

Peter Meyer and Jason Harrop favour this "named container" approach since it makes it easy for authors to create contracts in XML using standard XML authoring tools.

How does it make it easier? Particularly when it is more likely ‘easiest’ to create an XHTML 2.0 document using the element with standard XHTML authoring tools, i.e., without elements like “extras” and “extra” getting in the way.

The alternative the sub-committee considered is a generic container:

[instrument (h*, struct*) struct (nr?, h?, p*, (section* | struct* | instrument*))].

The alternative (as I am recalling it) before the TC is much closer to:

```
[ instrument    (h*, div*, p*, section, div*, area*)
  div           (nr?, h+, p*, div*|section|instrument, area*)
]
```

We considered: instrument (h*, p*, section*, struct*), but decided it was better not to allow p* or section* directly within <instrument>. Unfortunately, this important conclusion is contradicted by your proposed <instrument> content model… The generic struct container would have to be recursive to represent both a schedules element and the schedule elements contained within it. The generic container would need to carry an attribute identifying whether it contains for example, operative clauses or a schedule or the recitals.

The critical problem with this approach is that it is not easy to stop someone doing this:

```
<struct class="schedules">
        <struct class="operativeclauses"/>
</struct>
```

(ie putting the main operative clauses inside the schedules container, which should only

contain schedule elements) or, for example, from putting schedules or recitals inside their operative clauses.

Noone is suggesting to put “operative clauses” inside of a “schedules” via an attribute value or an element. I have repeatedly said there’s no need for an <operative> element – and certainly no need for an “operativeclauses’ attribute either. Placing a <section> element within the <instrument> is quite adequate to serve the function of identifying what you claim are ‘operative’ clauses for the instrument.

A content model which permits this:

- makes life difficult for our TC, since we need to document the various possible forms of markup and how they are to be treated by processing tools. Of course the spec needs to document all attribute values,,,, I guess your concern is based on worrying about the layout considerations earlier alluded to, which I find easily avoidable.
- makes life difficult for tool providers, since ease-of-use demands that they program around the unintended possibilities. Same issue – your statement is based on the presumption that we care to extract semantic information from contract language – I think we shouldn’t even try, for reasons outlined elsewhere.
- makes it hard for authors to use generic tools, since their desired alternatives are mixed up with non-sensical ones. Same issue – you want to extract semantic info from the contract language – I believe it’s folly to even try to do so. But the impact on authors is that they will be encumbered with providing (to them) extraneous markup that is totally meaningless to their tasks at hand. My proposal avoids that.
- makes life difficult for stylesheet developers, since they need to handle the unintended combinations authors will create when they use a generic tool. Same issues… Just look at the content models you’ve drafted as a direct result…. With its multiple variants – that’s not going to cause heartburn for XSL writers?

For these reasons, the structural model should where practical restrict valid content to exclude non-sensical constructs. This gets into validation issues, a topic I posted a note about and will spare everyone repeating the arguments here.

<snip/>

In addition the named containers can readily be transformed to vanilla standards-compliant XHTML should one wish to do so for any reason. Sure, with XSLT almost anything can be transforned into anything else – this is an uncommunicative statement.

I suppose what gives me greatest pause with this proposal, is that NO function for the element is defined at all. It’s difficult to support a specification that seeks to eliminate one of the most useful and used elements in the HTML dialect. It has a specific default meaning – a block-level element that contains flowed content – which I am proposing have the additional default styling of an implied “page-break”, and to use either @class or @property for typing it further.

I am also unimpressed by markup for partynames which contain no universal identifier for the parties named. That, I consider, is the proposal’s most fatal flaw – it pretends that these text strings are useful in their own right, declaring them of “obvious” signifcance to document exchange. But to what end? I don’t see how this aspect of their schema will enable anything.

Lastly, I don’t detect much sense of a “technical architecture” driving the design of the proposed schema – meaning there is precious little foundation there for the TC to build upon over time.