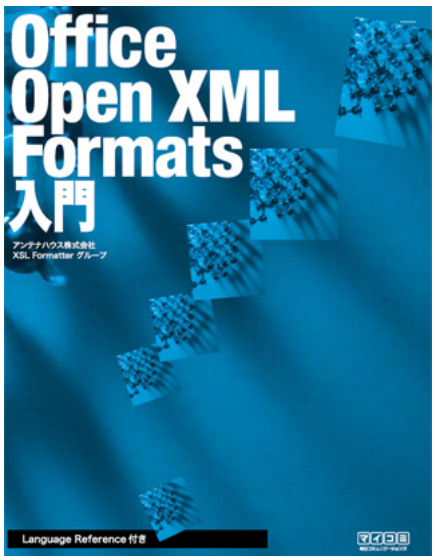
XSL-FO による書籍の自動組版

2007年11月 アンテナハウス（株） 石野恵一郎

目次

はじめに



アンテナハウスでは、2007年9月25日に「Open Office XML Formats 入門」を毎日コミュニケーションズ（マイコミ）から出版いたしました。この書籍は、表紙以外全てを XSL Formatter を使って自動組版し、PDF 化し、マイコミは PDF を印刷会社に渡すという方法で制作しました。商業出版における書籍の制作は DTP による方法が主流です。しかし、自動組版により、制作期間やコストを削減することが可能な分野も多いと思います。そのためには、商業出版用の書籍を、XSL-FO で制作する経験を重ねて、ツール、スタイルシート、および仕事の進め方を改善していくことが必要でしょう。

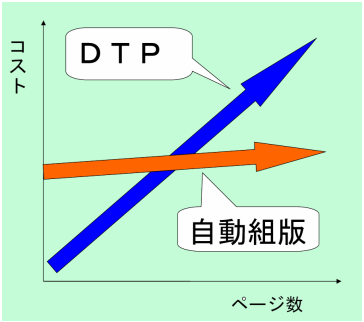
ここでは、XSL-FO による書籍の製作におけるその過程や、問題点などについて、かいつまんで紹介します。

DTP による方法との違い

DTP では、原稿を元に 1 ページずつページアップしていく作業を繰り返して書籍を完成させます。そのため、きめ細かな微調整や好き勝手なレイアウトが随所で可能ですが、同じ体裁の書籍を作る場合も、同じ手間がかかります。つまり、DTP による組版では、ページ数に比例してコストがかかることになります。

自動組版では、書籍の体裁に従ったスタイルシート（変換規則）を作成し、それに従ってすべてのページを機械的に組版します。同じ体裁の書籍ならば、同じスタイルシートを適用できるため、組版のコストはほぼゼロとなります。コストはページ数とは無関係で、最初のスタイルシート開発とその保守にコストがかかることになります。また、目次や索引などのページ付けも自動的に行なわれるため、改訂によるページずれなどを意識する必要がありません。また、スタイルシートを取り替えるだけで、まったく異なる体裁の書籍を組むことも容易です。

次の図は、毎度弊社のセミナなどで出てくるグラフです。



必要な知識

典型的な XSL-FO による自動組版を行なうためには、次のような知識が最低限必要です。これらは、原稿執筆者に求められる知識ではありません。自動組版システムを作ろうとする者に必要な知識です。

- XML
  原稿は、XML の形式を仮定します。どのような XML を用いるのかは、きちんと決めておく必要があります。元原稿が XML でない場合は、それを XML 化する必要があります。XML 1.0 は、JIS 化されています（JIS X 4159:2005）。
- XSLT
  原稿 XML を XSL-FO に変換するためのスタイルシートを記述します。このとき、付随して XPath などの知識も要求されるでしょう。ここでは、XSLT 1.0、XPath 1.0 を仮定します。これらは、JIS 化されています（JIS X 4169:2007、JIS X 4160:2007）。
- XSL-FO
  ここでの XSL-FO は、W3C による XSL 1.1 を指します。規格は XSL という名称です。XSLT と区別しやすくするために、XSL-FO と呼ばれることが多いです。組版自体の処理は、この XSL-FO を用いて行なわれます。XSL 1.1 は、JIS 化作業中です。
- 組版の知識
  当然ですが、正しい組版の知識がなければ、よい書籍は作れません。XSL-FO は、それを具現するための手段です。JIS X 4051:2004 は、ひとつの基準となります。

XML、XSLT、XSL-FO は、いずれも W3C による国際的な標準規格です。原勧告は、誰でも参照することができます。

製作過程

実際の「Open Office XML Formats 入門」の製作を振り返ってみます。

XML の決定

自動組版による製作過程において、もっとも重要なのは XML の決定です。これはきわめて重要です。いわば、原稿を日本語で書くのかアラビア語で書くのか、といったようなことを決めるわけです。(意味的にはちょっと違いますが)

XML は、原稿の文書構造を規定します。これは、自前で定義することもできますし、XHTML や DITA などの既存の XML を利用することもできます。XML には、その構造や意味を定義するために、XML Schema、RELAX NG や DTD（Document Type Definition）といった情報（スキーマ）が付いています（そんなものが存在しなくても、開発できなくはないですが)。例えば、XHTML における <h1> は最上位の見出しで、<p> は段落を意味する、とかそういったことです。

> XHTML はご存知でしょう。たぶん、もっとも身近な XML です。DITA（Darwin Information Typing Architecture）は、XML ベースの技術文書の生成、管理仕様として 2005年に XML 標準化団体 OASIS の標準として承認された仕様です。既に欧米の技術マニュアル等の制作において、実用として使われています。日本でもようやく着目され始めています。非常に大きな仕様なので、単行本にはオーバースペックでしょう。この他にも、DocBook や JapaX など多くのスキーマが公開されています。
>
> XML Schema は非常に複雑で使いにくく、普及しているとはいえません。RELAX は、それを打破する目的で作られたもので、RELAX NG はその後継にあたります。DTD は非常に簡単なことしか記述できませんが、書籍の組版程度では DTD でほぼ用が足ります。

採用しようとする XML は、これから作ろうとする書籍の内容を十分表現できるスキーマを持っている必要があります。文庫本のような単純な書籍の場合、大雑把に言えば、見出しと本文が表現できればよいということになります。これは、XHTML でも十分で、その中の <h1> と <p> だけのサブセットでも十分かも知れません。

原稿 XML は、このスキーマにのっとって記述されることになります。未知の XML のスキーマの習熟が面倒なら、XHTML などよく知った XML を採用するのも手です。

「Open Office XML Formats 入門」は、何段階かの見出しと本文の他に、表、画像、リスト構造といった、ごく一般的な文書構造が表現できれば組版できます。今回は、弊社でサンプルとして公開している SimpleDoc という XML を採用しました。しかし、この XML は仕様が古く、不足している機能を補ったり、だいぶ手を入れました。

> SimpleDoc は、ごく簡単な文書を記述できるようにした仕様です。非常に簡単ですが、その反面、気の利いた指定は何もできません。弊社では、現在見直し作業を行なっております。

原稿の XML 化

全体の製作過程の中で、原稿を XML に仕立てるのは、割りと面倒な作業かも知れません（しかもつまらない）。

原稿執筆者が始めから XML で原稿を書くのが手間がなくていいのですが、そんなことのできる執筆者はなかなかいないでしょう。たいていは、XML 化を担当する者が、何らかのオーサリングツールを用いて原稿を XML 化するのだと思われます。よく知られたツールには、

- xfy
- oXygen
- XML Spy
- XML Notepad

などが挙げられますが、テキストエディタのように選り取り見取りというわけにはいかないようです。これらは、いずれもスキーマを解釈して不適合な XML を生成しないようにすることができます。書籍のような単純なスキーマであるならば、テキストエディタで直接タグ付けすることも容易かも知れませんが、それは XML を知っている者にとってであって、何も知らない人にとっては、何らかのオーサリングツールが必要となるでしょう。

規模の大きい技術文書や、製造マニュアルなどは、その内容トピックを XML データベースとして管理することも多いでしょう。そして、あちこちから必要なトピックや数値データをかき集めて、ひとつの文書に仕立てるわけです。ここでは、そのような大規模なシステムには触れません。

「Open Office XML Formats 入門」は、初期の原稿は MS Word で記述していました。そして、ほぼ本文内容を書き終えた後 XML Spy で XML 化しました。その後は、その XML を手作業で調整しています。MS Word には戻っていません。これは、きわめて原始的で、いかにもやっつけ仕事という感が否めません。

印刷物が目的なので、使用している画像には高い解像度のものが要求されます。SVG で記述できる画像は、すべて SVG で記述し、印刷時の品質を維持するようにも心掛けました。MathML もそうですが、いずれも XML 技術の応用であるため、XSL-FO 中にそれらを記述することが自然に行なえます。

「Open Office XML Formats 入門」では、トビラ背景と柱部分に画像を指定されていましたので、それらを SVG で記述しました。SVG や MathML のオーサリングツールは、（私自身はよく知りませんが）あまり豊富ではないと思います。例えば、Adobe Illustrator で SVG が生成できます。今回作成した SVG は、トビラ背景、柱中の画像と本文中のフロー図です。単純なものは直に SVG のタグを書いて生成し、それ以外は Adobe Illustrator を利用して生成しました。トビラ背景の SVG は、次のように簡単です。

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20010904//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg width="188mm" height="240mm" viewBox="0 0 188 240"
```

```
  xmlns="http://www.w3.org/2000/svg">
  <desc>Background of Naka-Tobira with Gradient : bleed +3mm</desc>
  <defs>
    <linearGradient id="Gradient1" >
      <stop offset="0%"  stop-color="#666"/>
      <stop offset="60%" stop-color="#FFF"/>
    </linearGradient>
    <linearGradient id="Gradient2" >
      <stop offset="0%"  stop-color="#444"/>
      <stop offset="80%" stop-color="#FFF"/>
    </linearGradient>
    <linearGradient id="Gradient3" >
      <stop offset="20%"  stop-color="#FFF"/>
      <stop offset="100%" stop-color="#000"/>
    </linearGradient>
  </defs>
  <g stroke="none">
    <rect width="188" height="240" fill="url(#Gradient2)"/>
    <rect x="18" y="22" width="142" height="0.2" fill="url(#Gradient3)"/>
    <rect x="18" y="216" width="142" height="0.2" fill="url(#Gradient3)"/>
    <rect x="18" y="22" width="0.2" height="194" fill="#FFF"/>
    <rect x="160" y="22" width="0.2" height="194" fill="#00"/>
    <circle cx="120" cy="18" r="14" fill="#FFF"/>
    <g transform="translate(132,35)">
      <circle r="14" fill="url(#Gradient1)" transform="rotate(135)"/>
    </g>
  </g>
</svg>
```

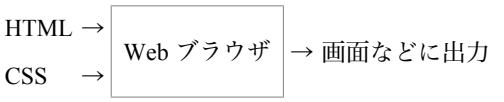
この背景画像は、裁ち落としを考慮してページサイズより上下左右に 3mm 大きく作られています。

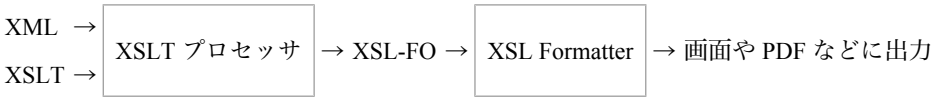
また、「Open Office XML Formats 入門」はモノクロ書籍なので、画面のキャプチャなどの含まれている画像はすべてグレイスケールに変換しました。

XSLT の作成

XSLT は、原稿 XML を XSL-FO に変換するために利用します。これは、ただの構造としての XML に、ページレイアウトのための情報を与えることを行ないます。XSLT はスタイルシートと呼ばれます。これは、HTML に対する CSS と同じです。

HTML+CSS と、XML+XSLT の処理の流れは次のようになります。





この図のように、HTML+CSS では、Web ブラウザが出力までを行なってしまいますが、XML+XSLT の場合は、一旦 XSLT プロセッサによって XSL-FO を生成し、その後 XSL Formatter のような XSL プロセッサがそれを組版して出力することになります。実際には、その流れは中間ファイルなどを作ることなく、シームレスに行なわれます。

XSLT スタイルシートの開発は、書籍のできを左右します。XSLT スタイルシートは、ページの版面などを記述し、採用している XML スキーマに従って、各要素属性から、どのような XSL-FO を生成すべきかを記述します。XSLT では、目次を生成したり、索引語を収集して索引を作成したりもします。また、柱やノンブルなどの指定を XSL-FO 中に埋めることも行ないます。

XSLT プロセッサには、Windows では標準的に組み込まれている MSXML がありますが、Xalan や Saxon など、UNIX でも利用できるものがいくつか存在しています。Java には、API が用意されています。これらは、XSLT 仕様に従っ

て作られているので、どれを使っても結果に違いはありませんが、それぞれ便利な拡張があったり、インタフェースが違っていたりします。

「Open Office XML Formats 入門」で作成した XSLT スタイルシートは、SimpleDoc のものを利用しましたが、索引関係が要求を満たさないため、そこだけ別途作成しました。あまりにも締め切りに間に合わすための作業だったため、一般に披露するような代物になっていないのが残念です。

> その代わりに、この XHTML を処理する XSLT を用意したので、参照してください。

書籍用 XSL-FO で、柱やノンブル、目次、索引などはどのように指定するのか、それを XSLT で生成するにはどうするのか、といったことを簡単に紹介します。（以下の例は簡略化されています）

柱、ノンブルの生成

柱やノンブルは、XSL-FO では <fo:static-content> という要素中に記述します。これは、本文の外に書かれています。固定的な柱ならば、そこに文字列を指定しておくだけです。XSLT は、そういう要素を出力すれば済みます。

```
<fo:static-content flow-name="page-header">
  <fo:block font-size="0.8em" text-align="center" space-before="0.5in">
    XSL-FOによる書籍の自動組版
  </fo:block>
</fo:static-content>
<fo:static-content flow-name="page-footer">
  <fo:block font-size="0.8em" text-align="center" space-after="0.5in">
    - <fo:page-number/> -
  </fo:block>
</fo:static-content>
```

本文内容の一部や見出しを柱に含めることは普通に行なわれます。「Open Office XML Formats 入門」では、左ページ（偶数ページ）には、書籍名と章の見出しとノンブル、右ページ（奇数ページ）には、章の見出しとノンブルを出力しています。また、各章先頭ページは別の形式の柱を出力しています。XSL-FO では偶数ページや奇数ページ、先頭ページといったそれぞれのページごとに、体裁を指定することができます。XSLT は、それら必要な体裁の指定をすべて出力しなければなりません。

見出しの内容などを柱に含めるには、XSL-FO では次のように <fo:retrieve-marker> という要素を使います。

```
<fo:static-content flow-name="page-header">
  <fo:block font-size="0.8em" text-align="center" space-before="0.5in">
    <fo:page-number/> |
    Office Open XML Formats入門
    <fo:retrieve-marker retrieve-class-name="part-title"/>
  </fo:block>
</fo:static-content>
```

本文中には、

```
<fo:marker marker-class-name="part-title">WordprocessingML</fo:marker>
```

というように <fo:marker> が、柱を変更すべき場所に埋められています。<fo:marker> 自体は、本文内容には影響しません。組版をしていて、<fo:marker> が出現すると、<fo:retrieve-marker> の対応する class-name の内容が <fo:marker> のものに置き換わって柱が出力されます。

XML 中では、

```
<part>
  <title>WordprocessingML</title>
    <chapter>
      <title>基本構造</title>
        <p>パッケージ構造で説明したように、Wordドキュメントのデータは…
```

としか書かれていないので、XSLT の処理で part/title を見つけたら <fo:marker> を出力するようにします。

```
<xsl:template match="part">
  …
```

```
  <fo:marker marker-class-name="part-title">
    <xsl:value-of select="title"/>
  </fo:marker>
  …
</xsl:template>
```

そのページが偶数ページか奇数ページか等は自動的に判定され、対応する柱に適用されます。

目次の生成

目次の生成は、XSLT の役目です。目次の生成では、XML 中のどの要素を目次項目にするのかをまず決めます。「Open Office XML Formats 入門」では、<part>、<chapter>、<section> といった要素のタイトルを目次として生成します。目次は、たいていは本文組版前に、本文を走査して組んでしまいます。ページ番号はこのときは決まりません。ページ番号は、本文組版中に次々と決まっていきます。

```
<xsl:for-each select="//part | //chapter | //section">
  <xsl:variable name="level" select="count(ancestor-or-self::part |
                                           ancestor-or-self::chapter |
                                           ancestor-or-self::section)"/>
  <fo:block text-align-last="justify">
    <xsl:attribute name="start-indent">
      <xsl:value-of select="($level - 1) * 2"/>
      <xsl:text>em</xsl:text>
    </xsl:attribute>
    <xsl:attribute name="font-size">
      <xsl:choose>
        <xsl:when test="$level=1">10pt</xsl:when>
        <xsl:otherwise>9pt</xsl:otherwise>
      </xsl:choose>
    </xsl:attribute>
    <xsl:attribute name="font-weight">
      <xsl:choose>
        <xsl:when test="$level=1">bold</xsl:when>
        <xsl:otherwise>normal</xsl:otherwise>
      </xsl:choose>
    </xsl:attribute>
    <xsl:if test="$level=1">
      <xsl:number/>
      <xsl:text> </xsl:text>
    </xsl:if>
    <xsl:value-of select="title"/>
    <fo:inline font-weight="normal" font-size="9pt" font-family="'Times New Roman'">
      <xsl:choose>
        <xsl:when test="$level=1"><fo:leader leader-pattern="space"/></xsl:when>
        <xsl:otherwise><fo:leader leader-pattern="dots"/></xsl:otherwise>
      </xsl:choose>
      <fo:page-number-citation ref-id="{generate-id()}"/>
    </fo:inline>
  </fo:block>
</xsl:for-each>
```

このような XSLT で、「Open Office XML Formats 入門」のような目次が生成されます。ここにある generate-id() というのは、ユニークな id を生成する関数で、目次付けされる対応項目は、次のように XSLT 処理されています。

```
<xsl:template match="section">
  <fo:block id="{generate-id()}" …>
    …
```

索引の生成

索引の生成も、XSLT の役目です。索引を生成するには、本文中にあるどの語を索引項目とするかを指定しなければなりません。「Open Office XML Formats 入門」では、<index> というマーク付けで索引語を指定しています。XSLT では <index> に対しては、次のように id を付ける処理を行ないます。

```
<xsl:template match="index">
  <fo:inline id="{generate-id()}">
    <xsl:apply-templates/>
```

```
    </fo:inline>
  </xsl:template>
```

ただ単に索引語を収集して索引を作成するだけならば、目次と同じようにすればできます。

```
  <xsl:for-each select="//index">
    <fo:block text-align-last="justify">
      <xsl:value-of select="."/>
      <fo:inline font-weight="normal" font-size="9pt" font-family="'Times New Roman'">
        <fo:leader leader-pattern="dots"/>
        <fo:page-number-citation ref-id="{generate-id()}"/>
      </fo:inline>
    </fo:block>
  </xsl:for-each>
```

しかし、これではよみ順に整列もされていませんし、同じ索引語をまとめることもできません。「Open Office XML Formats 入門」では、索引語に次のようなマーク付けがされています。

```
<p>…<index key="すたいるじょうほう">スタイル情報</index>を持つ<index>webSettings.xml</index>があります。</p>
```

索引語によみ情報を与え、索引の整列に利用します。

```
  <xsl:for-each select="//index">
    <xsl:sort select="normalize-space(concat(@key,' ',.))"/>
    ...
```

key 属性が指定してあればそれで、なければ <index> の内容で整列されます。XPath には条件式はないので、key 属性の値と、<index> の内容を連結してキーとしています。正確ではありませんが、だいたいうまくいきます。もしうまくいかないときは、key 属性の与え方を工夫すれば回避できます。

> よみの整列では、小書きの仮名や正濁音などをうまく処理する必要があります。実際の XSLT では、これらの処理も行なっていますが、ここでは省略します。

重複キーの削除は、（同じ索引語は同じにマーク付けされていると仮定して）次のようにします。

```
  <xsl:variable name="items" select="//index"/>
  <xsl:for-each select="$items[not(.=preceding::index)]">
    <xsl:sort select="normalize-space(concat(@key,' ',.))"/>
    ...
```

しかし、これだけでは参照ページがひとつしか出力されません。次のようにして、基本的な索引の出力は完成します。

```
  <xsl:variable name="items" select="//index"/>
  <xsl:for-each select="$items[not(.=preceding::index)]">
    <xsl:sort select="normalize-space(concat(@key,' ',.))"/>
    <xsl:variable name="item" select="."/>
    <fo:block text-align-last="justify">
      <xsl:value-of select="$item"/>
      <fo:inline font-weight="normal" font-size="9pt" font-family="'Times New Roman'">
        <fo:leader leader-pattern="dots"/>
        <xsl:for-each select="$items[.=$item]">
          <fo:page-number-citation ref-id="{generate-id()}"/>
          <xsl:if test="position()!=last()">, </xsl:if>
        </xsl:for-each>
      </fo:inline>
    </fo:block>
  </xsl:for-each>
```

「Open Office XML Formats 入門」では、索引を頭文字で区切って、見出しを付けて出力するようにしています。これは、上のループ中で、直前の見出し項目と異なる頭文字が現れたら見出しを出力する、という方法でも実現できます。ここでは、キー集合を定義する方法を紹介します。

まず、索引キーの頭文字の集合を <xsl:key> で定義します。

```
  <xsl:key name="index-key" match="index"
           use="substring(normalize-space(concat(@key,.)),1,1)"/>
```

これで、すべての <index> に現れる索引キーが、頭文字ごとに分類登録されます。そして、索引の出力を、頭文字ごとに行なうようにします。

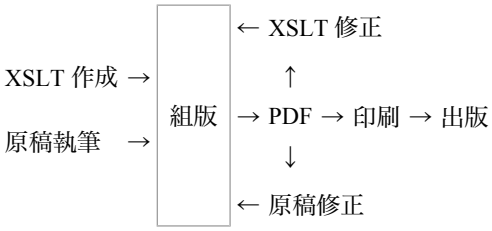
```
<xsl:template name="make-index">
  <xsl:call-template name="process-index">
    <xsl:with-param name="initial-str">ABCDEFG…XYZあいうえお…わ</xsl:with-param>
  </xsl:call-template>
</xsl:template>

<xsl:template name="process-index">
  <xsl:param name="initial-str"/>
  <xsl:variable name="initial">
    <xsl:value-of select="substring($initial-str,1,1)"/>
  </xsl:variable>
  <xsl:if test="$initial!=''">
    <xsl:variable name="items" select="key('index-key',$initial)"/>
    <xsl:if test="$items">
      <fo:block xsl:use-attribute-sets="見出し用">
        <xsl:value-of select="$initial"/>
      </fo:block>
      <xsl:for-each select="$items[not(.=preceding::index)]">
        このループは同じ
      </xsl:for-each>
    </xsl:if>
    <xsl:call-template name="process-index">
      <xsl:with-param name="initial-str" select="substring($initial-str,2)"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

XSL 1.1 にある、より高度な索引機能は使用していません。同一ページ番号の除去についても省略しています。

組版

XSL-FO さえできてしまえば、組版して結果を出力するのは機械作業です。それは XSL Formatter が行ないます。自動組版の典型的なワークフローは、次のようになります。



XSL-FO による自動組版での留意点

XSLT

XSL-FO による自動組版を行なうには、XSLT と XSL-FO という別々の技術を必要とします。ここでは、XSLT を開発する上での留意点については、あまり触れません。原稿記述に XHTML を利用するときに、XSLT で気をつけるべき事柄をひとつだけ述べておきます。

XHTML で、きめ細かな制御を行なおうと思ったら、class 属性を多用せざるを得ません（別の名前空間の XML を利用する方法もありますが)。class 属性は、ひとつの属性値に複数を指定できるので、

```
<p class="first keepnext">
```

などと記述できます。これを次のように処理したとします。

```
<xsl:if test="contains(@class,'first')">
 ...
</xsl:if>
<xsl:if test="contains(@class,'keepnext')">
 ...
</xsl:if>
```

これは、単純に class 属性値中に first という文字列が含まれているか否かという判定でしかありません。例えば、

```
<p class="foo bar">…
<p class="foobar">…
```

のような場合、

```
<xsl:if test="contains(@class,'foo')">
```

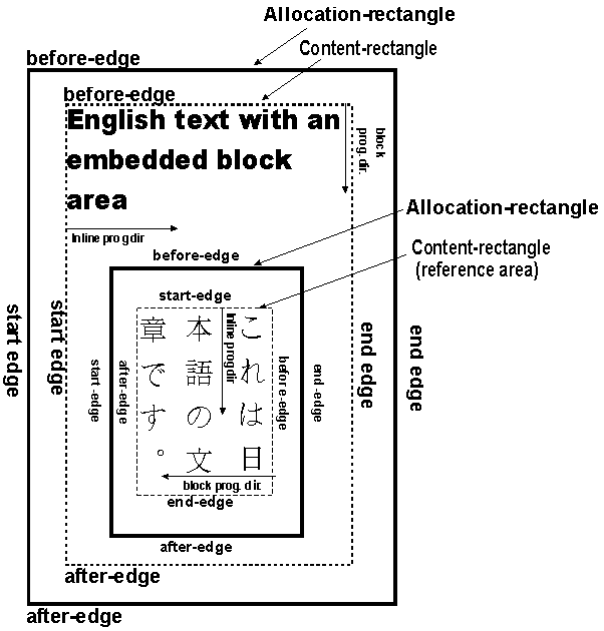
ではうまくいきません。XSLT 1.0（というか XPath 1.0）の文字列処理は非常に貧弱なため、このような文字列の正確な分割処理（字句解析）はほぼできないので、class 名には、部分一致しないものを付けておくのが無難です。

XPath 2.0 では、正規表現が使えたり字句解析ができたりするので、正確に判定できますが、XSLT はもう少し複雑になります。上のように単純にできるなら、その方がよいでしょう。

```
<xsl:for-each select="tokenize(normalize-space(@class), ' ')">
 ...
</xsl:for-each>
```

XSL-FO

XSL-FO の組版のモデルは、ページの中に長方形の領域を作り、その中に内容を流し込むというものです（領域モデル）。次の図は、XSL 1.1 からの引用です。このように、いろいろな領域が入れ子になって配置されている、と考えればよいです。



テキストは、領域をあふれたところで折り返され、その行の高さ分送られて次の行の端から埋められていきます。

XSL-FO は、レイアウトは規定するけれども、禁則処理はどのようにすべきかとか、どこでハイフネーションすべきかとかいうようなことまでは規定していません。これらは、XSL Formatter のような XSL プロセッサが都合のよいように実装することになります。XSL Formatter は、文字の性質に関する部分を Unicode 仕様に従っています。禁則処理、行分割処理については Unicode Standard Annex #14: Line Breaking Properties（UAX#14）にほぼ従っています。

XSL-FO のレイアウト指定能力は、（すったもんだしている CSS3 はともかく）現在の CSS2 よりも優れています。だからといって、XSL-FO は万能ではありません。日本語組版に限らず、不足している機能がいくらかあります。代表的なものは、段抜きと脚注でしょう。XSL-FO では、段抜きは全段抜きしかできません。脚注は、全段抜き脚注しかありません。各段末に脚注を配置することはできません。頭注や傍注といった構造もありません。これらは他の構造で代用することになります。

> XSL Formatter は、XSL-FO に不足する多くの機能を独自に拡張し、実装しています。それだけではただのローカル仕様になってしまうため、W3C へ提案し、これらの拡張が標準仕様となるように働きかけています。いくつかは、XSL 1.0 から XSL 1.1 への段階で取り入れられています。
> 段配置の脚注については、XSL Formatter では拡張済みです。傍注といった概念も導入済みです。部分段抜きは、XSL Formatter でもできません。拡張予定にはなっています。

これらを踏まえた上で、XSL-FO による組版を行なうと、それなりの体裁に組み上がります。

XSL-FO の多くのプロパティ値のデフォルトは、「何もしない」とか「ゼロ」とかです。つまり、何かしたければほとんどすべてのプロパティを明示的に指定してやる必要があります。また、どのプロパティがどのように継承されていくのかはきちんと理解しておく必要があります。例えば、

```
<fo:block font-size="12pt">
 <fo:block>
  いろはにほへと
 </fo:block>
 <fo:block>
  ちりぬるを
 </fo:block>
</fo:block>
```

というような場合、内側のブロックにも外側のフォントサイズが適用されるということです。

継承でもっとも陥りやすいのが、マージンやインデントなどの継承の問題です。

```
<fo:block start-indent="1em">
 Hello World!
 <fo:table>
  <fo:table-body>
   <fo:table-row>
    <fo:table-cell>
     <fo:block>
      CELL-1
     </fo:block>
    </fo:table-cell>
    <fo:table-cell>
     <fo:block>
      CELL-2
     </fo:block>
    </fo:table-cell>
   </fo:table-row>
  </fo:table-body>
 </fo:table>
</fo:block>
```

外側のブロックにあるインデントの指定は、テーブルセルの内側のブロックにまで継承されます。そのため、各セルにインデントが取られることになります。(下線はアキを表わしています)

```
__Hello World!
____CELL-1__CELL-2
```

多くの場合、これは意図した結果ではありません。これを回避するためには、

```
<fo:block start-indent="1em">
 Hello World!
 <fo:table>
  <fo:table-body start-indent="0pt">
   <fo:table-row>
   ...
```

のようにして継承を断つ必要があります。これによって、次のような結果となります。

```
__Hello World!
__CELL-1CELL-2
```

ちょっと考えればわかることですが、ここに、`<fo:table>` に指定しているのではないことに注意してください。`<fo:table start-indent="0pt">` としてしまうと、テーブル自身への指定となるので、次のようになります。

```
__Hello World!
CELL-1CELL-2
```

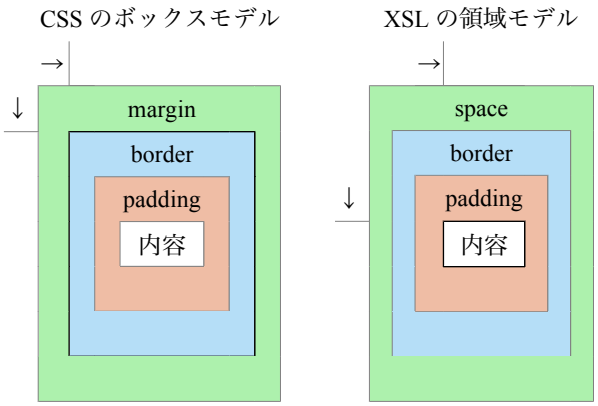
さらに面倒なのが、margin-* というプロパティです。margin-left や margin-top などのプロパティは、CSS から持ち込まれたプロパティで、XSL-FO では、他のプロパティに変換されます。例えば、margin-left からは start-indent が計算されて、それが保持されます。

```
<fo:block margin-left="1em">
 Hello World!
 <fo:table>
  <fo:table-body margin-left="0pt">
   <fo:table-row>
   ...
```

このような指定では、セルのインデントは打ち消すことができません。この例では border や padding が指定されていないので、ごく単純に、margin-left="1em" は、start-indent="1em" に変換されます。`<fo:table-body margin-left="0pt">` では、継承された start-indent が考慮されて、結果としてそこでの start-indent は 1em のままとなります（詳細は XSL 1.1 の 5.3.2 を参照してください）。このような小難しいことを避けるために、margin-* は使わずに、start-indent や padding-* などのプロパティを利用することを勧めます。

> XSL-FO の継承では、参照領域の考えをよく理解しておく必要があります。また、空白の扱いもよく理解しておかなければなりません。

CSS に慣れた人は、そのモデルの違い（CSS のボックスモデル、XSL の領域モデル）にも注意する必要があります。CSS では、border や padding は指定されているボックスの内側に取られますが、XSL では領域の外側に取られます。



「Open Office XML Formats 入門」で最後まで調整していたのは、次のような部分です。

…なんとかかんとかです。
次の図を参照してください。 ← ここで改ページが発生

何かの図

widow、orphan は、（わざわざ無効にしていなければ）普通に行なわれるので問題とはなりません。ここでは、ただ１行の段落がページ末に位置し、それが次の図や表を指す場合です。この例のような、互いに関連しているブロックには、分離を制限するプロパティを付けておく必要があります。例えば、原稿 XML では、次のように何か印を付けることを行ないます。

```
<p>
 …なんとかかんとかです。
</p>
<p class="keepnext">
 次の図を参照してください。
</p>
<p>
 <img .../>
</p>
```

これを XSLT で処理して、次のような XSL-FO に変換します。

```
<fo:block>
 …なんとかかんとかです。
</fo:block>
<fo:block keep-with-next.within-page="always">
 次の図を参照してください。
</fo:block>
<fo:block>
 <fo:external-graphic .../>
</fo:block>
```

あるいは、次のようにもできます。

```
<fo:block>
 …なんとかかんとかです。
</fo:block>
<fo:block keep-together.within-page="always">
 <fo:block>
  次の図を参照してください。
 </fo:block>
 <fo:block>
  <fo:external-graphic .../>
 </fo:block>
</fo:block>
```

XSL-FO による日本語組版の問題点

XSL-FO は、世界中の言語を組版することを目指しているので、日本語を組むこともできますし、縦書きにすることもできます。しかし、日本語組版（例えば JIS X 4051:2004）としての機能が少々不足しています。いくつかは代替手段がありますが、いくつかは W3C の XSL-FO の仕様では表現不可能です。例えば、次のような事柄が問題となるでしょう。しかし、XSL Formatter では、これらの多くを拡張機能として実装していますので、多くは問題とはなりません。

- 版面の指定が、文字数×行数でないため、換算しなければならない。（XSLT でうまく指定することができます）
- 強制的に行をグリッドに沿わせることができないため、途中に図版などが入るとずれてしまう。
- 約物に対するきめ細かな詰めの指定ができない。（XSL Formatter は、約物詰めのきめ細かな制御ができます）
- 起こし食い込みやぶら下げができない。（XSL Formatter では、ぶら下げができます。起こし食い込みは近い将来実装されます）
- 日本語に対する禁則処理は、UAX#14 ではうまくいかないものがある。（XSL Formatter は、日本語の禁則処理を正しく行ないます）
- 和欧文間空白をとれない。（XSL Formatter は、和欧文間空白を制御できます）

- ルビ、圏点、割注、縦中横、振分けなどの、日本語組版で必要とされる構造がない。（いくつかは XSL-FO で擬似的に実現できます）
- タブ処理の機能がない。
- JIS X 4051:2004 に規定されているような図版の配置調整ができない。

> 漢文などは割愛しました。

版面の指定

日本語組版では、組体裁をデザインするとき、基本となる版面の大きさを、文字数、行数、行間から決め、それを紙面のどこに配置するかを決めます。欧文組版では、上下左右のマージンから二次的に版面の大きさが決まります。XSL-FO も、このような指定を行なって版面の大きさや位置を決めます。したがって、日本語組版での基本版面の指定方法は、天地ノド小口からの距離に換算する必要があります。XSL では次のように指定することができます。

```
<xsl:param name="body-font-size">9pt</xsl:param>
<xsl:param name="body-line-height">15pt</xsl:param>
<xsl:param name="page-width">182mm</xsl:param>
<xsl:param name="page-height">257mm</xsl:param>
<xsl:param name="page-margin-right">
  ( <xsl:value-of select="$page-width"/>
  - 46 * <xsl:value-of select="$body-font-size"/> ) div 2
</xsl:param>
<xsl:param name="page-margin-left">
  <xsl:value-of select="$page-margin-right"/>
</xsl:param>
<xsl:param name="page-margin-bottom">
  ( <xsl:value-of select="$page-height"/>
  - 39 * <xsl:value-of select="$body-line-height"/>
  - <xsl:value-of select="$body-font-size"/> ) div 2
</xsl:param>
<xsl:param name="page-margin-top">
  <xsl:value-of select="$page-margin-bottom"/>
</xsl:param>
```

これは、用紙サイズ B5 縦、文字サイズ 9pt、行間 6pt、46字×40行で、版面をページ中央に配置するときの例です。

XSL-FO で採用されている単位 pt（ポイント）は、1pt = 1/72in = 0.3528mm です。日本語組版では JIS Z 9305 による jpt、1jpt = 0.3514mm が利用されています。XSL-FO では jpt は利用できません。正確に行なうならば、この換算も必要となるでしょう。また、級（1q = 0.25mm）も XSL-FO では利用できないので、必要ならば換算しなければなりません。

行グリッド

日本語組版では、基本版面でデザインした行位置に各行を揃えるのが原則です。小さい文字が含まれれば行間は広くなり、大きい文字が含まれれば狭くなります。XSL-FO のデフォルトは、行間を均一にするのが基本なので、何も指定しなければ、途中に大きい文字が入ったりするだけで、行の位置は不揃いとなります。このため、日本語組版では line-stacking-strategy="line-height" の指定が不可欠となります。次の例は、文字サイズ 9pt、行間 6pt の指定です。

```
font-size="9pt"
line-height="15pt"
line-stacking-strategy="line-height"
```

しかし、これはブロック内での制御の話なので、ブロックに space、border、padding が付いている場合は、その分ずれてしまいます。また、行取り見出しや途中に図版が入る場合なども、よほどうまく調整しないと、それ以降の行はグリッドに沿いません。その他にも、行がずれる要因はいろいろあります。XSL-FO には、自動的に行グリッドに沿わせる機能はありません。

> CSS3 の glid-height は、この目的に適います。

数式や図版、表、プログラムの引用などの多い技術文書では、あまり行グリッドにこだわる必要はないのでしょう。

行方向で、各文字を無理にグリッドに合わせる必要はありませんが、日本語組版では、折り返し行末を揃えて組むのが基本です。したがって、XSL-FO では、

```
text-align="justify"
```

の指定は必須です。

約物の詰め処理

日本語の文字（フォント）は、全角幅の正方形の仮想ボディを持っているのが一般で、漢字仮名などはすべて同じ大きさです。句読点や括弧などの約物は、字幅を半角として扱うとされています（JIS X 4051:2004）。そして、組版時に「アキ」を入れて組むことになっています。しかし、実際のフォントでは、これら約物も漢字などと同じ全角幅を持っています。つまり、初めから半角幅の（二分の）アキが含まれています。このため、実際には、「アキ」を入れるのではなく、「詰め」の調整をすることになります。

> 全角幅を持っていない、プロポーショナルなフォントには、「詰め」処理が適用できないことに注意してください。プロポーショナルフォントは、横書き用のフォントであり、正方形の仮想ボディを前提とする日本語組版の規則は適用できないと考えます。

XSL-FO には、このような「詰め」の調整機能はありません。XSL Formatter では、次のような調整を行なえるように拡張しています。

- 行頭での全角開き括弧
- 行末での全角閉じ括弧
- 全角閉じ括弧と全角開き括弧の間
- 全角閉じ括弧と全角閉じ括弧の間
- 全角閉じ括弧と全角中点類の間
- 全角閉じ括弧と非約物の間
- 全角開き括弧と全角開き括弧の間
- 全角中点類と全角開き括弧の間
- 非約物と全角開き括弧の間

ここでは、句点と読点は、閉じ括弧と同じに扱っていますが、これらは括弧類とは区別すべきものとされています。例えば、行末での句点では、二分アキを詰めてはならない、ということになっていますが、このような指定はできません。

区切り約物（疑問符、感嘆符）前後のアキを調整することは、XSL Formatter ではできません。テキストとして空白を挿入しておくなどの対処が必要です。

日本語組版の行全体の詰め処理では、約物前後の調整対象のアキを優先的に調整する処理が行なわれます。XML Formatter ではこのような詰め方法は行なっていません。

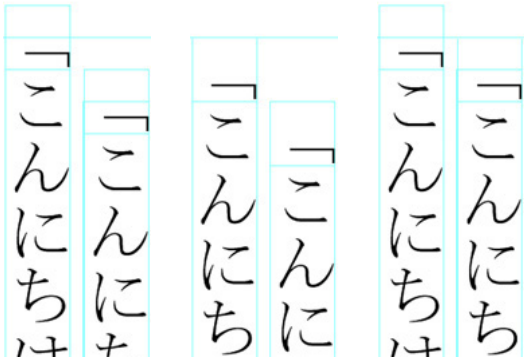
起こし食い込みとぶら下げ

日本語の段落での全角下げ（改行行頭での字下げ）は、text-indent="1em" として指定できます。日本語組版では、行頭に配置する開き括弧は、改行行頭の字下げと、折り返し行頭の字下げとで次の３通りの方法があります。

	1	2	3
改行行頭の字下げ	全角	全角半	二分
折り返し行頭の字下げ	天ツキ	二分	天ツキ

XSL Formatter では、3 を指定することはできません。行頭での全角開き括弧を詰める指定をすれば 1 となり、指定をしなければ 2 となります。

1 2 3



XSL Formatter で、句読点をぶら下げ組することができます。そのとき、ぶら下げは強制されません。なりゆきで、版面をはみ出すのを許容するという制御を行なっています。

禁則処理

日本語組版での禁則処理や、行分割処理は、UAX#14 でほぼうまくいきます。しかし、うまくいかないものがいくつかあることがわかっています。XSL Formatter では、次のような UAX#14 とは異なる処理をしています。

- JIS X 4051:2004 での行頭禁則和字は、UAX#14 では NS（Nonstarter）の行分割クラスに分類されます。日本語組版において、これらを行頭禁則しないで組むことが多いため、これらを NS から除外するオプションを設けています。
- 次の引用符の行分割クラスは QU（Ambiguous Quatation）です。
  - U+2018（LEFT SINGLE QUOTATION MARK）
  - U+2019（RIGHT SINGLE QUOTATION MARK）
  - U+201C（LEFT DOUBLE QUOTATION MARK）
  - U+201D（RIGHT DOUBLE QUOTATION MARK）

  QU は、前後で分割されません。日本語組版では、これらを OP（Opening Punctuation）または CL（Closing Punctuation）として処理します。
- 次の句読点の行分割クラスは CL です。
  - U+3001（IDEOGRAPHIC COMMA）
  - U+3002（IDEOGRAPHIC FULL STOP）
  - U+FF0C（FULLWIDTH COMMA）
  - U+FF0E（FULLWIDTH FULL STOP）

  CL は、直後に AL が連続しているときにそこで分割されません。日本語の（プロポーショナルでない）全角にはもともとアキが含まれているので、そこで分割されるべきです。日本語組版では、これらを NS として処理します。
- 日本語組版での行分割クラス AI（Ambiguous Alphabetic or Ideographic）は、ID（Ideographic）として処理します。ただし、U+2015（HORIZONTAL BAR）は、JIS X 4051:2004 での分離禁止文字なので、IN（Inseparable）として処理します。

- 半角カナ（U+FF61〜 U+FF9F）の行分割クラスは AL（Ordinary Alphabetic and Symbol Characters）です。これは、アルファベットと同じで分かち書きをしないと行分割されません。半角カナは、全角カナ（U+3001〜 U+30FF）として行分割処理します。

行分割クラスの調整ではどうにもならないものがあります。

- 全角句読点の直後に半角の引用符 U+0022 が来ると、その間で行分割しません。これは、UAX#14 で、QU 前後の分割が禁止されているからです。U+0022 のような引用符は、開きか閉じかがあいまいです。

```
"どうしたこうした。"
ああしたそうした。"abc..."
```

  半角引用符を使わないようにするとか、分割を許すような場所には、空白や、U+200B（ZERO WIDTH SPACE）、U+200C（ZERO WIDTH NON-JOINER）などを挿入する必要があります。

```
“どうしたこうした。”
ああしたそうした。&#x200B;"abc..."
```

Unicode には、分離禁止という考えがありません。これは、均等割付のときなどアキを挿入できない場所として扱われます。日本語組版では、分離禁止の場所はすべて分割禁止となります（原則）。XSL Formatter では、分離禁止の処理は行なっていません。

> U+200D（ZERO WIDTH JOINER）は、分割禁止を強制する文字です。分離禁止は意味しません。

和欧文間空白

日本語組版では、和欧文間に四分（原則）アキを入れることが行なわれます。XSL Formatter では、それを自動的に行なうことができます。

> JIS X 4051:2004 には欧文の定義がありません。XSL Formatter では、Latin、Greek、Cyrillic の各スクリプトを欧文とみなしています。つまり、Arabic や Thai などのスクリプトとの間には適用されません。

日本語組版をよく知った執筆者は、原稿にあらかじめ半角空白を挿入して和欧文間を分離していることがあります。たいていの欧文フォントでの半角空白は、ちょうど四分幅程度なので、ほとんど同じに組版されます。

ルビ等

日本語組版で用いられる次のような構造は、XSL-FO では表現できません。XSL Formatter でも表現できません。

- ルビ
  ルビは、<fo:inline-container> を使って、ある程度模倣できます。しかし、これでは日本語組版が要求する仕様を満足しません。行分割に対応できませんし、均等割付もできません。CSS3 では考慮されています。
- 圏点
  ルビと同じように <fo:inline-container> を使って、無理やり圏点を付けることができるかも知れません。CSS3 では考慮されています。
- 割注
  割注も、ルビと同じように <fo:inline-container> を使って、ある程度模倣できます。しかし、行分割に対応できません。CSS3 では考慮されています。
- 縦中横
  縦中横にしたい文字列に、回転などを指示しなければなりません。CSS3 では少しだけ触れられていますが、縦中横という構造を直接表現できるわけではありません。

- 振分け
  CSS3 でも考慮されていません。

タブ処理

XSL-FO でも CSS でも、JIS X 4051:2004 や MS Word にあるようなタブストップの考えはありません。

図版の配置調整

図版がページ境界などにかかったとき、近隣テキストとの入れ替えを行なってうまくページ頭やページ末に収めようとか、ページ途中の図版を、強制的に天側や地側に持って行こう、などという機能です。これは、JIS X 4051:2004 で規定されています。この機能自体は、何も日本語組版に限ったことではないのですが、XSL-FO にも CSS にもありません。

XSLT サンプル

この文書の原稿は、XHTML で記述されています。CSS は、XHTML 内に記述されています。これを XSL-FO に変換する XSLT サンプルも作成してあります。これは、弊社 Web サイト からダウンロードできます。このアーカイブには、以下が含まれています。

- OOXMLBK.html
  原稿 XHTML ファイルです。Web ブラウザで閲覧可能です。
- *.jpg、*.svg、*.emf
  XHTML 中で使用している画像です。SVG や EMF は、PDF 化のときに使われます。
- xhtml2fo.xsl
  XHTML を XFL-FO に変換するスタイルシートです。このスタイルシートには、XHTML 自体の汎用的な変換指示のみが含まれています。CSS は処理しません。各書籍ごとに異なる組体裁に関する情報は、params.xsl などに分離しています。このスタイルシートは、弊社 Web サイトでサンプルとして公開しているものを改造したものです。
- params.xsl
  版面の指定、フォントの指定など、組体裁にかかわる基本的な情報がパラメタ（xsl:param）として定義されています。xhtml2fo.xsl からインポートされます。
- attributes.xsl
  XHTML の各タグに対して、どのような属性の組を適用させるか（xsl:attribute-set）が定義されています。xhtml2fo.xsl からインポートされます。
- classes.xsl
  XHTML 中に記述されている class 属性の値を処理します。目次や索引の処理は、この中に含まれています。xhtml2fo.xsl の最後でインクルードされます。
- OOXMLBK.pdf
  XSL Formatter を用いて、OOXMLBK.html を組版した PDF です。

XSL Formatter で、OOXMLBK.html と xhtml2fo.xsl を指定して組版することができます。このスタイルシートは、B5 版にデザインされています。どのような XSL-FO が生成されているのかは、GUI の [ファイル]-[FO の保存] メニューを選んで XSL-FO を保存して確認することができます。

> インタネットに接続されていない環境では、MSXML が XSLT 処理に失敗することがあります。このときは、OOXMLBK.html の先頭にある DOCTYPE 宣言を削除してください。DOCTYPE 宣言のある XHTML を XSLT 処理するとき、MSXML など

の XSLT プロセッサは、DTD を取得しに行き、適合性を調べ、実体参照を解決しようとします。これは、環境によってはコストがかかります。わたしは、前処理で XHTML から DOCTYPE 宣言を削除する処理を行なうことがよくあります。

まとめ

XSL-FO を用いた書籍の自動組版は、欧米では広く行なわれるようになっていますが、日本では多くありません。それにはいくつかの理由が挙げられます。

- 欧文組版の規則のほとんどが XSL 仕様に含まれている。
- 欧文組版のできる XSL プロセッサがいくつも存在する。
- XSLT 開発のできる技術者が多い。
- XML のいろいろなスキーマに関するコミュニティが発達している。

日本では、これらは次のように対応します。

- 日本語組版の規則のいくつかが XSL 仕様に不足している。
- 日本語組版のできる XSL プロセッサは、XSL Formatter だけである。
- XSLT 開発のできる技術者が少ない。
- XML そのものを積極的に取り入れようとしているところがあまり多くないのではないか。

XSL Formatter は、日本語だけでなく、アラビア語、タイ語、ヒンディ語など、多言語組版ができる XSL プロセッサです。XSL Formatter は、多くの仕様の不備を補い、日本語組版ができるようになっていますが、日本語組版の専門家から見て、及第点をもらえるというレベルには達していないかも知れません。これについては、もっとよい製品に育てる努力をすれば解決可能だと思います。しかし、XSL-FO を使う側の人たちにとって、すぐに問題となるのは次のようなことでしょう。

- MS Word で原稿を書くように、XML 原稿を楽に書きたい。
  規模の大きい XML データベースシステムの末端の出力として、XSL-FO が使われる場合は問題とはなりません。そのときは XML も自動的に生成されるからです。
  DTP の代替と考えるとき、安価でよいオーサリングツールが必要となるでしょう。
  MS Word そのものを入力ツールとして XML を出力する方法も考えられるのですが、文書構造の表現方法が XSL-FO とは大きく異なるので、XSLT で苦労することになります。WordML を XSL-FO に変換するスタイルシートは、XSL Formatter にも組み込まれていますが、いろいろ制約のあるのが現状です。今回の書籍のテーマである Open Office XML を扱うスタイルシートは、まだ準備できていません。
- XSL-FO や XSLT を勉強したいが、よい参考書やサンプルが欲しい。
  XSL-FO は、CSS に比べてとっつきにくく、仕様としても難しいといえます。仕様書は 500ページ（A4換算）に及び、しかも文章は平易ですが内容の理解は容易ではありません。初学者の理解を助けるための、セミナなどの機会と、多くの「有意義な」サンプルコードが必要でしょう。

  最近では、バッチ組版のための [XSL-FO 指南書]（藤島雅宏編著）という参考書も出ています。

これらは、もっと多くの方が自動組版に興味を持ってくださるようになれば、その分解決も早まるでしょう。皆さんも、ものは試し、やってみてください。
一方、XSL でなく、CSS で組版をしようという動きもさかんです。

Prince は、CSS による組版を行ないます。

（まだまだ勧告には程遠いですが）CSS3 のページモデルは強力で、CSS で組み体裁が指定できるようになれば、利用者に選択肢が増え、自動組版もより普及していくでしょう。

現在、W3C では、日本語組版の要求仕様をまとめる作業を（日本で）行なっています。これは、日本語の知識のない欧米人などに、日本語組版を理解してもらい、XSL や CSS などへの反映を促す目的で行なわれています。この要求仕様は、日本人にとっても役に立ちます。（難解で読みにくい JIS とは違って）記述が平易であり、JIS X 4051:2004 に含まれていない事柄も含まれており、網羅的です。XSL Formatter では、これらの要求は積極的に取り入れ、その実装仕様は W3C に提案していく予定です。

索引

written by ISHINO Keiichiro.