FOA: an XSL-FO Authoring Tool

Fabio Giannetti
Publishing Systems and Solutions Laboratory
HP Laboratories Bristol
HPL-2001-130
May 25th , 2001*

E-mail: Fabio_Giannetti@hp.com

XML, XSL, XSL-FO, FO, authoring, FOA, XSL-T, presentation, rendering, document

FOA (Formatting Object Authoring) is an authoring tool that applies rich styling to XML content. It allows the styling to be re-used across multiple documents. It also allows the author to build or import a library of style components. It is based on XSL-FO, the W3C-defined markup language, whose aim is to add rich styling to XML content, especially for paginated documents.

The architecture of FOA permits the author to create and modify style sets, and also allows these styles to be re-applied to different XML content. The tool generates an XSL stylesheet, based on the style information, that takes the XML content and produces an XSL-FO document, which can then be rendered into different output formats. An extension to an XSL-FO renderer allows the author to preview the output within the authoring tool.

The architecture was developed as my part of the co-supervision of a Master Thesis, in collaboration with the University of Genoa, during which significant pieces of FOA have also been implemented.

* Internal Accession Date Only
Approved for External Publication
© Copyright Hewlett-Packard Company 2001

1. Introduction

The separation of content, expressed in application-specific XML-based dialects, from presentation is now well established. There are plenty of tools that allow authors to create XML documents, defining their own tags, DTDs and Schemas. The addition of appropriate semantic tags to the content gives a much greater chance that the content can be reused, including for different presentation purposes.

Of course, an XML document requires a presentation to be defined before it can be rendered and made easily readable to users. So far, most emphasis has been placed on transforming documents for presentation using web delivery markup languages such as HTML or WML. These are relatively simple formats compared to the richness of presentation possible on the printed page.

Paginated documents for web delivery have so far mainly been authored using conventional word processing tools. Adobe's PDF is often used as a delivery format because it preserves the appearance of the document and provides good quality when it is printed. However, this approach assumes the content is re-authored specifically for printing.

Our authoring tool anticipates the creation of XML-based documents with a high semantic content and the need to style them in a rich paginated form for delivery using PDF and similar page description languages. Web authoring tools can hide the complexities of styling markup from authors. For example, today it is possible to write HTML documents without knowing anything about Cascading Style Sheets (CSS), simply using WYSIWYG tools. The styling complexity required to express rich paginated layout is considerably more difficult than that used in HTML.

The FOA authoring tool sets out to provide an improved authoring environment for styling XML documents using XSL-FO styling markup. It gives authors a better way to style their documents, maintaining the separation between content and style and making the transformation stylesheets reusable as much as possible.

2. Creating documents using XSL-FO

XSL (Extensible Stylesheet Language) [1] is a W3C standard that consists of two parts, a transformation language called XSLT [2] and a presentation markup language usually called XSL-FO [3]. XSL-FO, currently a W3C Candidate Recommendation, defines an object-based presentation markup language, independent from any platform or user agent implementation, that can be used to add rich styling to XML content.

Given an arbitrarily structured XML document, an author can use an XSL stylesheet to express how the content should be presented [4]. The presentational part of the stylesheet expresses the way in which the document will be styled, laid out and paginated onto some presentation medium or a set of physical pages.

Processing a stylesheet involves two steps: first to transform and add styling information to the document, and then to render it into a suitable delivery format. The first step is normally performed using an XSLT processor that transforms the XML content according to the XSL stylesheet into a target presentation markup language. A render engine then interprets the presentation markup to make it viewable. For instance a web browser such as Microsoft's Internet Explorer [7] can invoke an XSLT processor to transform XML into HTML according to an XSL stylesheet. The browser itself includes an embedded render engine to render the HTML as a web page [5].

In our case, for rich paginated documents, the presentation markup language is XSL-FO. The XSL-FO markup is added by the XSLT processor [8] [9] during the XSL stylesheet transformation. **Figure 1** shows the two step process to obtain a final rendered document in a deliverable page description language.

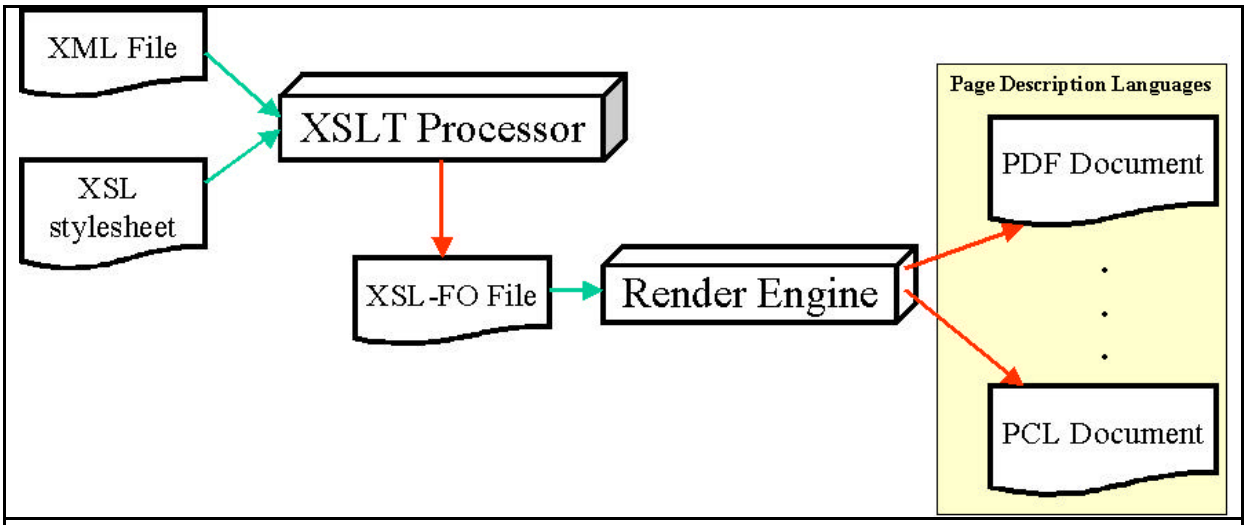

Figure 1 – Two steps process: transform and render

XSL-FO Formatting Objects provide an output-independent way to express document layout. When the XSL-FO file is parsed by the render engine it will be translated into an internal area-based model that positions each presentation element in the correct relative position. The internal representation of the area-based document can then be transformed into a page description language, converting each area, with the associated traits, into the corresponding output notation. The area representation gives the render engine the possibility of creating multiple formats, starting from the same document description. This is an advantage for the delivery of documents across multiple platforms and devices.

To achieve this goal the document author must write an XSL stylesheet that transforms an XML document into an XSL-FO document, and defines how the appropriate styling is applied for each XML tag or group of tags. The transformation part of a stylesheet has all the power of a programming language. In fact, the author must know concepts, like tree manipulation, that are more familiar to a programmer than an author/designer.

Today XSL is a technology that is not usable by many authors because it involves a lot of knowledge and skill before someone can start to style non-trivial documents. Like for HTML, XSL need some user-friendly tools that help authors to style their documents, maintaining the separation between semantic content, transformation processes and styling of the document. FOA is intended to provide such support.

3. The FOA authoring tool

FOA is an authoring tool that helps authors/designers to give a layout and style to their XML content. It is not a WYSIWYG editor but, using a consistent preview and a property manager, it can give the necessary feedback to understand the obtained results. FOA is intended to style documents that are expressed in XML and may have even been generated automatically.

FOA creates an XSLT file, which when applied to the XML using an XSLT processor, generates the appropriate XSL-FO. In addition, FOA stores the style attributes in one or more separated XSL files, so the author can re-use them for processing other XML documents.

Figure 2 shows the FOA functional context. In particular, it shows how multiple XML source documents many be used to compose the final XSL-FO document, and how multiple style attribute sets can be combined to generate the correct transformations.

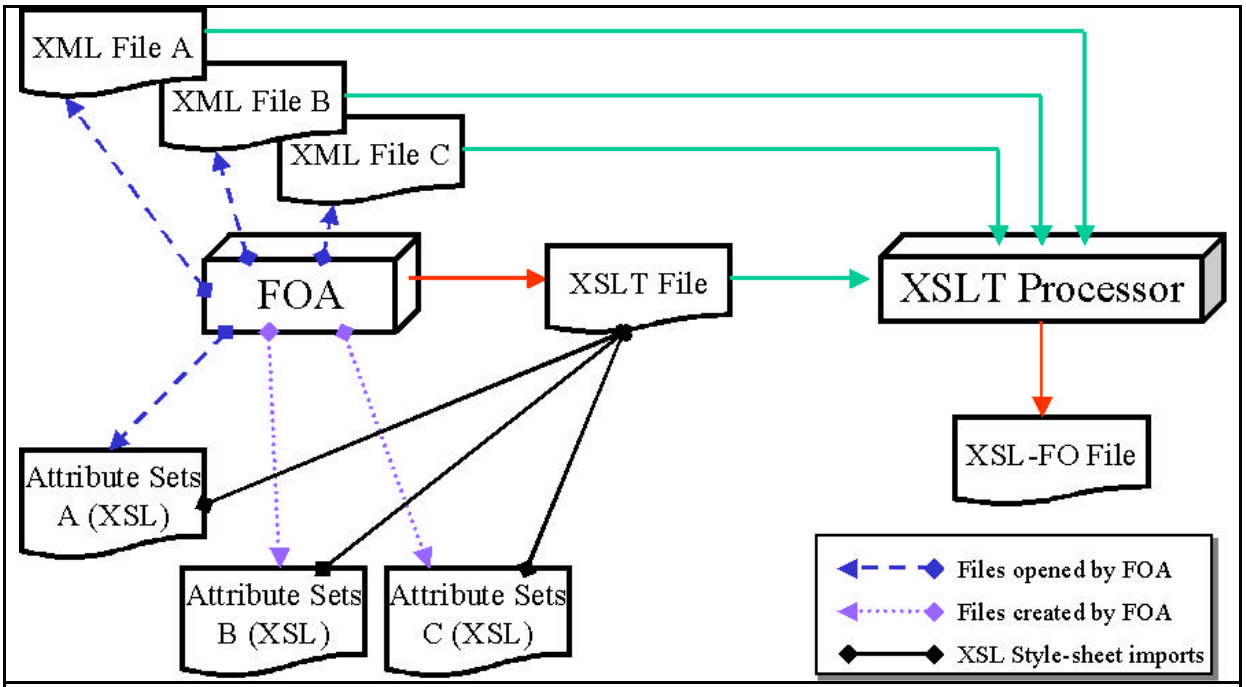

Figure 2 – FOA functional context

An author can create a new or open an existing XSLT file. FOA will automatically open the related Attribute Set(s) and the related XML content file(s). At this point the author can modify/add/delete the attributes, stored inside the Attribute Sets, or generate a new family of Attribute Sets. The author can also include new content by importing a new XML file. When the authoring is completed, FOA will save a new XSLT file that includes all of the Attribute Sets actually used, and references to all the XML content involved, into the transformation process.

There several different situations in which FOA can help authors format XML content with XSL-FO:

- to create new transformations and layout styles for some XML content; in this case, the author will create a new XSLT which takes the XML content and transforms it into a new XSL-FO document with the newly defined styles
- to create new transformations in order to apply styles developed for one piece of XML content to a different piece of XML content; in this case, the author can re-use the Attribute Sets previously created and associate them to different XML content which may use different tags from the original
- to re-use an existing transformation by modifying or adding new styles; in this case, some existing XML content will be re-styled by changing the style information only.

To achieve these results FOA uses two paradigms, a fundamental transformation element called a “Brick” and a set of properties to apply to the transformed content called an “Attribute Set”. In the next sections we will explain the approach taken to give the necessary independence to the transformation and the styling parts.

4. A re-usable transformation component: “Brick”

As already mentioned, many authors will prefer a re-usable and reliable way to create a transformation stylesheet without needing to learn the details of the transformation language.

To achieve this, FOA introduces the concept of a re-usable transformation component called a “Brick”. Each Brick is used to apply styling to a well-defined component of the XML document. There will be a Brick that represents a text block, or an image or a table and so on. As part of each Brick, there is some XSLT code that transforms the XML content into the appropriate set of Formatting Objects.

Some Bricks can be very simple, others can be very complicated (e.g. for tables), but the author doesn’t have to worry about the details because FOA will create the necessary XSLT transformations to achieve the results. The only action the author must perform is to relate the content to the style. In complex cases, like tables, FOA will

ask the author to relate more than a single tag to the style, and sometimes it will automatically import a content sub-tree.

The Bricks are subdivided into classes. When the author chooses to create a block of text, first of all he will ask the application to create the corresponding object in the FO space: a `<fo:block>`. Secondly, the author will choose a defined style from those that apply to that class of block. This subdivision helps the author in choosing a suitable set of style attributes that will be applied to the content and also can provide a series of hints explaining the effect given by the chosen attributes.

```
<xsl:template match=path  foa:class=class >
      <fo:element class=label  xsl:use-attribute-sets=label >
           <xsl:apply-templates/>
      </fo:element>
</xsl:template>
```

Figure 3 – A simple Brick

Figure 3 shows the basic architecture of a Brick. It is formed of a standard XSLT template, including an XPath match on XML content. A special attribute `foa:class` specifies the kind of Formatting Object that will be created by the transformation process. This class information is used when the style-sheet is re-opened and/or the author creates a new Brick, because it helps keep track of the created Bricks and their purpose.

Figure 4 describes how a Brick will be filled in by FOA:

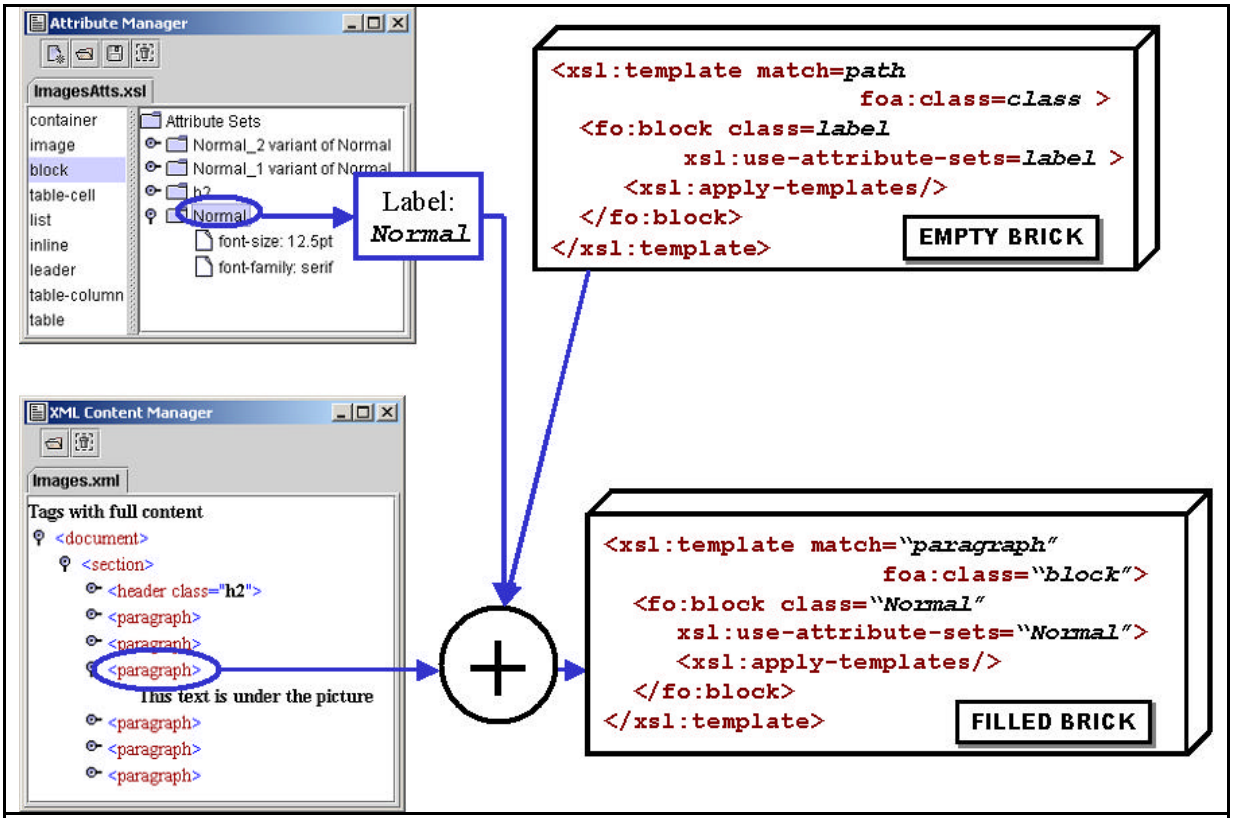

Figure 4 – How FOA fills a Brick

The author can select which part of the content is to be styled by simply selecting a node on the XML content tree. FOA allows the author to select one of the Attribute Sets, using the Attribute Manager, only from those in the same class as the Brick.

When the XSLT is processed the result will be that in **Figure 5**:

```
<fo:block font-size="12.5pt" font-family="serif" text-align="center"
      class="Normal">This text is under the picture</fo:block>
```

Figure 5 – Result of a processed Brick

A complex Brick, such as the one for tables, is composed of a set of simple Bricks that transform the different parts of a complex element. Inside a table, there are some rows and cells; so the table Brick is composed of three different Bricks: one is responsible for the layout of the entire table, one for the rows and one for the cells. The author can express different styles for each of these levels, and must also provide the appropriate content information. The **Figure 6** shows a filled table Brick.

```
- <xsl:template match="table" foa:class="table">
  - <fo:table class="Table" table-layout="fixed" width="426.1pt" xsl:use-attribute-sets="Table">
      <fo:table-column column-number="1" column-width="142pt" />
      <fo:table-column column-number="2" column-width="142.05pt" />
      <fo:table-column column-number="3" column-width="142.05pt" />
    - <fo:table-body>
        <xsl:apply-templates />
      </fo:table-body>
    </fo:table>
  </xsl:template>
- <xsl:template match="row" foa:class="table">
  - <fo:table-row class="table">
      <xsl:apply-templates />
    </fo:table-row>
  </xsl:template>
- <xsl:template match="cell" foa:class="table-cell">
  - <fo:table-cell class="Cell" column-number="{position()}" xsl:use-attribute-sets="Cell">
      <xsl:apply-templates />
    </fo:table-cell>
  </xsl:template>
```

Figure 6 – A filled table Brick (formed of 3 sub-Bricks)

5. A re-usable style component: Attribute Set

Like the transformation process, the author will expect style components to be re-usable. In fact, defining complex styles for some content can be a long process for the author and it can often be achieved only after some trial and error. Moreover documents created in some well-defined environments must share the same layout, such as for company reports. The author will want to create sets of attributes and a transformation process once, and re-use them both for many documents.

The Formatting Object doesn't provide the same structure that CSS does in giving a general definition of attributes to be applied to all HTML paragraphs. However this kind of "global" style can be achieved using the XSLT paradigm `<attribute-set>` and `xsl:use-attribute-sets` to specify which set must be used for that FO element.

The XSLT specifications also allow the author to define a set that can use another Attribute Set. FOA supports this by using the concept of variant. Starting from an attribute set, the author can define a variant of it in order to add or modify some of its defined values. FOA allows the creation of variants only inside a class to avoid the possibility of adding incorrect attributes to a Formatting Object.

Figure 7 depicts an example of an Attribute Set:

```
- <xsl:attribute-set name="Normal" foa:class="block">
    <xsl:attribute name="font-size">12.5pt</xsl:attribute>
    <xsl:attribute name="font-family">serif</xsl:attribute>
  </xsl:attribute-set>
```

Figure 7 – An Attribute Set

FOA stores in each Attribute Set all the style information that the author has specified. To make it easier to select the correct property settings, FOA subdivides the properties into common and specific subsets, in the same way

as in the XSL-FO specification [3]. Like the Brick, the Attribute Set also has a `foa:class` definition that tells FOA which kind of class can use the set.

Figure 8 shows how the author can modify the properties starting from an Attribute Set, and how the properties are subdivided into classes with both common and specific subsets.

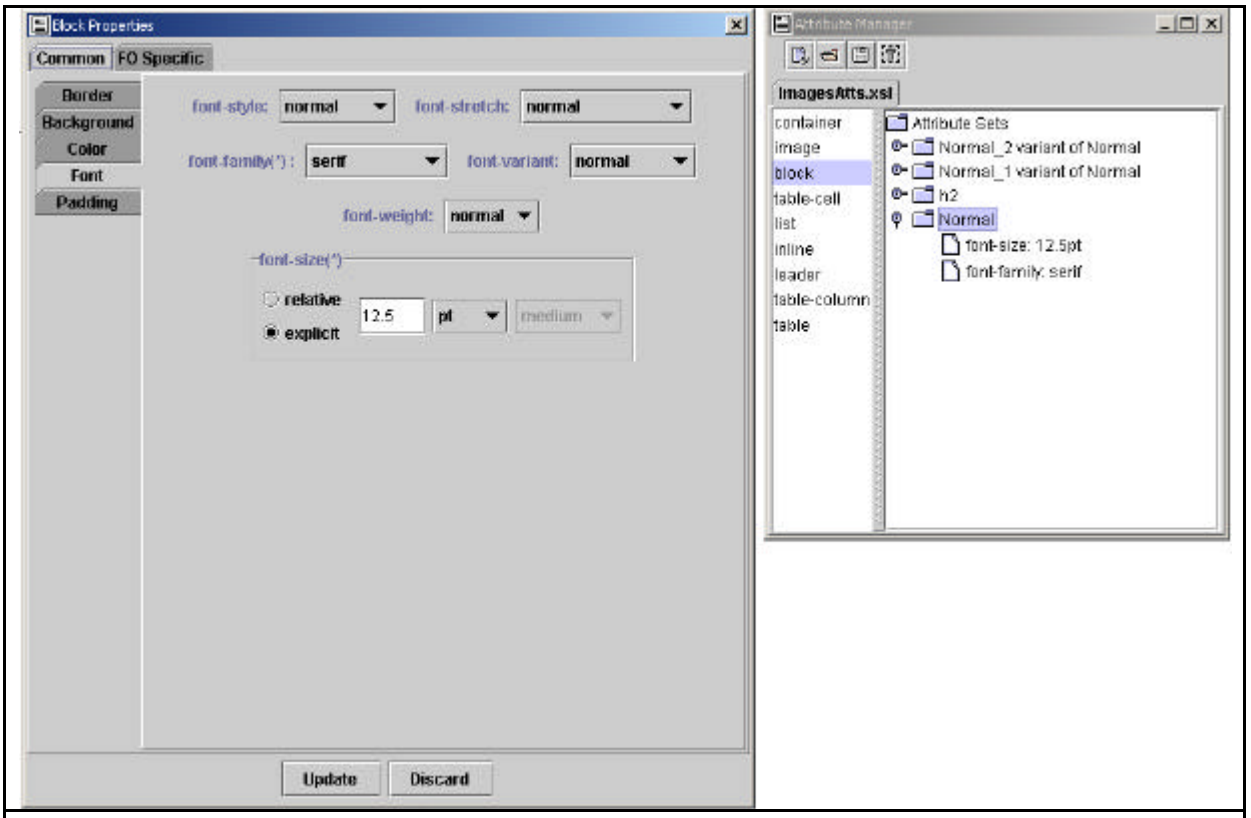

Figure 8 – Management of Attribute Sets (using subdivision in classes)

6. A consistent preview: Rendered Document XML Description

FOA is not a WYSIWYG application. It therefore becomes important to have a consistent preview to give the author feedback about the effect of the chosen styling on the document. To achieve this, the application must render the entire document using the generated XSLT applied to the XML content. In order that FOA could be an authoring tool that is not dependent on any specific implementation of a render engine, an XML description of a rendered document has been defined as part of the architecture. Using this description, FOA can open a preview window in which the author can see the rendered result.

Figure 9 shows the way in which FOA uses a render engine to generate the Rendered Document XML Description while maintaining independence from both it and the XSLT processor.

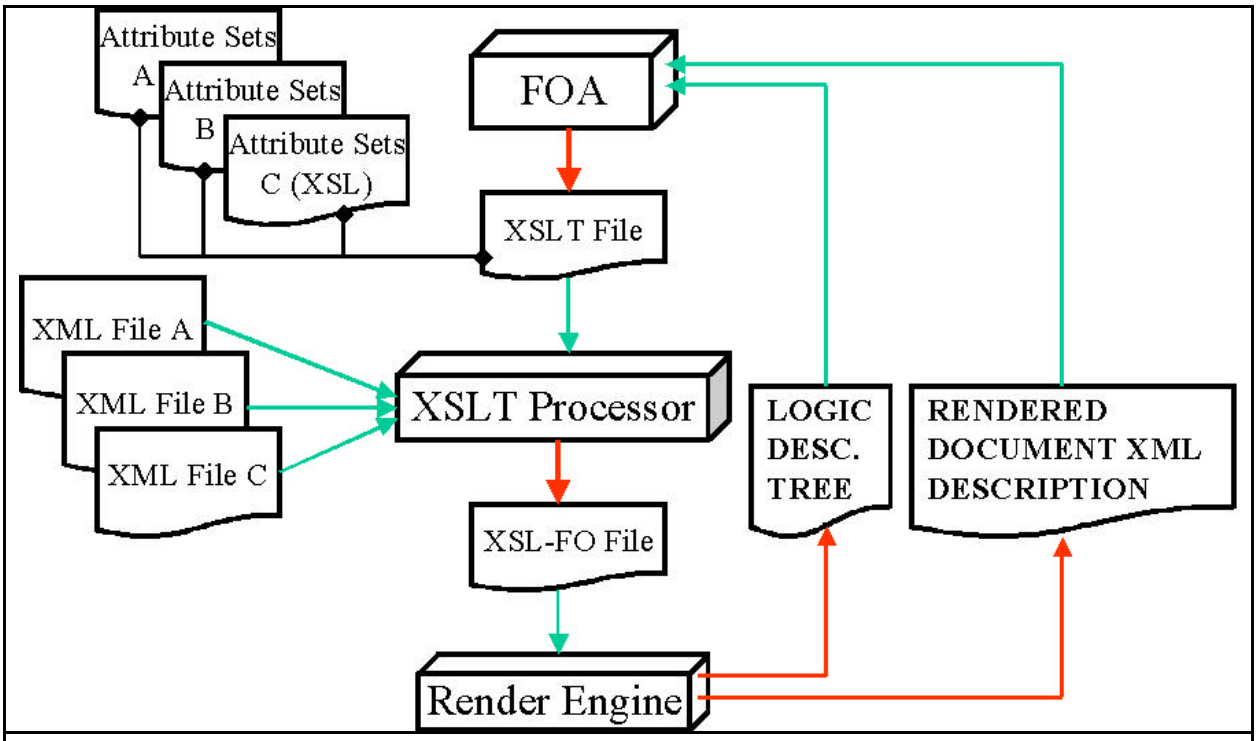

Figure 9 – Separation of FOA from the XSLT Processor and the Render Engine

As well as the rendered description, another XML file describing the logical structure of the rendered document is also returned by the render engine. This second description preserves the logical structure of the document in order to better relate the preview to the original style information.

In general, a single Brick could be used to style many elements inside a document. For example, a paragraph style can be applied to many paragraphs inside the document. Using the information in the Logic Description Tree, the tool can generate a visual link between the rendered areas of the document and the associated style so that the author can check or modify the effect on a particular paragraph.

As another example, in the case of a table the rendered document may have several areas that are created starting from the same Brick. So if the author wants to see the impact of changing the table style, it must be possible to show the amount of content inside the document that will be affected.

In **Figure 10** you can see an example of a Rendered Document XML Description and in **Figure 11** the corresponding XML Logic Description Tree. These examples were made with an experimental version of a render engine developed locally. However, other render engines, such as Apache's FOP [6], could be extended to generate similar output.

Figure 10 – Rendered Document XML Description (extract)

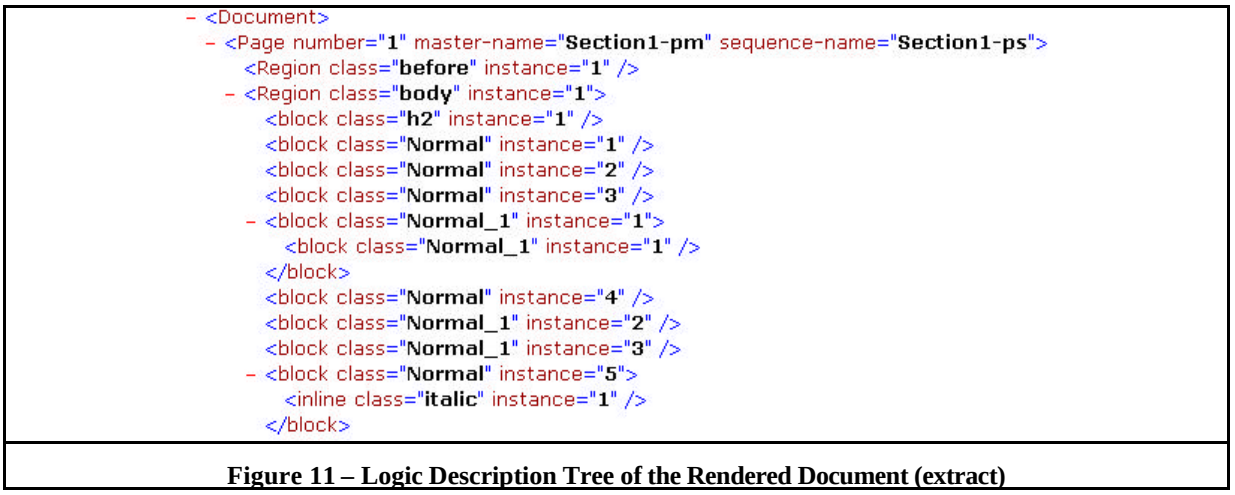

Figure 11 – Logic Description Tree of the Rendered Document (extract)

The Rendered Document XML Description is primarily a notation that expresses the areas created during the rendering process including the area position information inside the page (always absolute) and the area dimensions.

The FOA preview can take this information, together with the Logic Description Tree, and present the rendered document, page by page. The author can highlight an entry in the Logic Description Tree and see the associated area in the rendered document. FOA also has all the information to retrieve the Brick that has generated the area and the corresponding style that was applied to it. The author will be able now to modify the attributes and asking for a new preview to check the effect of the modification.

The class and instance information from the tree are used to highlight the correct amount of information inside the rendered document and to retrieve the applied style. FOA will understand in which class the template was created from the name of the tree tag (in **Figure 11** the `class` `Normal_1` is related to the tag `block`).

Figure 12 shows the preview and the highlighted bracketing provided on selecting a node on the Logic Description Tree.

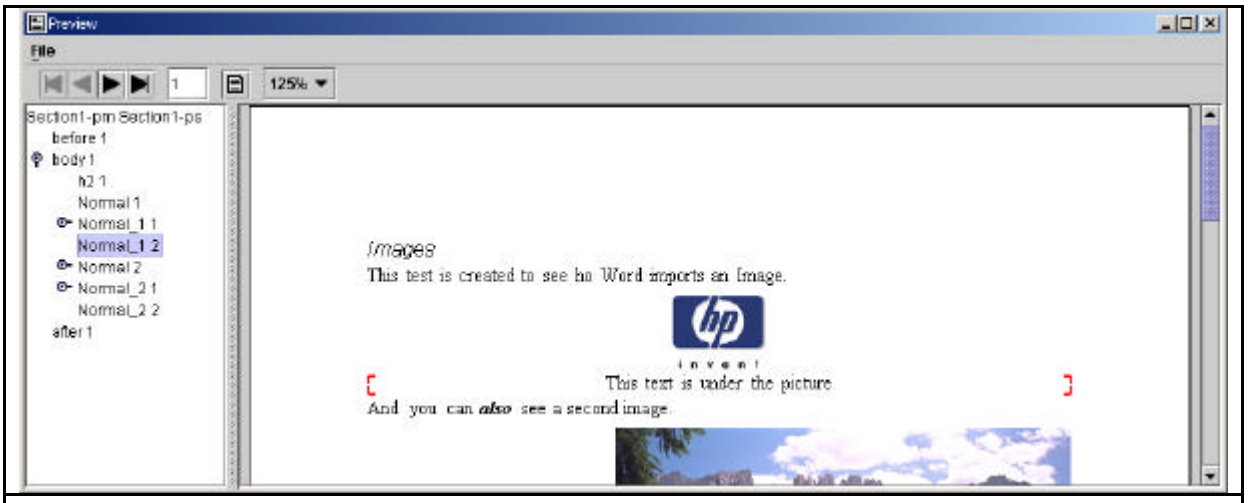

Figure 12 – FOA Preview

7. Conclusion

The adoption of a complex but powerful technology such as XSL-FO depends on providing tools that allow an author to create documents without needing the skills of a programmer. FOA is an experimental authoring tool that provides visual support for creating XSLT style-sheets which transform XML content into XSL-FO documents.

The re-usability provided by the Bricks and Attribute Sets permit authors to create a pool of styling components suitable for more than a single document. The Rendered Document Description and the corresponding Logic Description Tree allow the author to gain maximum value from a rendered document preview.

In this way, FOA provides the kind of assistance in the XML authoring environment that style templates offer in conventional word processing.

8. Acknowledgments

This work has been carried out in collaboration with the Biophysical and Electronic Engineering Department (D.I.B.E.) of the University of Genoa.

In particular, I would like to thank Dr. Anna Marina Scapolla for her support in proposing this as a Masters project and her co-supervision, and to Emiliano Grigis for his implementation of substantial pieces of the tool as part of his thesis.

I would also like to thank my colleague at HP, Roger Gimson, for his comments and help in preparing this document.

9. References

[1] *XSL web page*, World Wide Web Consortium, http://www.w3.org/Style/XSL/ (2001)

[2] *XSL Transformations (XSLT) Specifications Version 1.1 (Working Draft)*, World Wide Web Consortium, http://www.w3.org/TR/xslt11 (12 December 2000).

[3] *XSL Specifications (XSL-FO) Version 1.0 (Candidate Recommendation)*, World Wide Web Consortium, http://www.w3.org/TR/xsl/ (21 November 2000).

[4] Walsh, N., The Extensible Style Language: XSL, *WEB Techniques*, Vol.4 no. 1 (49-50,52,54-5) (Jan 1999).

[5] McGrath, S., Rendering XML documents using XSL, *Dr. Dobbs Journal*, Vol. 8 Art. 8 (Jul 1998).

[6] *Apache's FOP*, XML Apache Project, http://xml.apache.org/fop/

[7] *Microsoft Internet Explorer 5.x*, Microsoft Corp., http://www.microsoft.com/windows/ie/

[8] *XT*, James Clark, http://www.jclark.com/xml/xt.html

[9] *Saxon*, Michael Kay, http://users.iclway.co.uk/mhkay/saxon/