An SVG Browser of XML Languages

Musbah Shahop Sagar
Department of Computing, Oxford Brookes University
Oxford, OXON, UK

Abstract

In this paper, we will show how XML languages such as XHTML, MathML and SVG can be rendered into SVG. The SVG generated can then be displayed on a screen or printer. A variety of architectural approaches is explored and we present both server-side and client-side solutions using engines written in Java and JavaScript. Our approach enables a document containing mixed XHTML, MathML and SVG to be rendered for display. This approach has potential for the creation of scientific web sites that frequently require a combination of text, mathematics and diagrams.

1. Introduction

Writing a web site for e-science is a challenging task given by the nature of the web today. Scientists and engineers require text along with mathematical expressions and diagrams in the same web pages. Using the ordinary technology available today, one has to assume that web users are using web browsers support for different types of XML languages (such as XHTML[19], MathML[3], SVG[2], etc). However, browsers may or may not support these specifications and even if they do support them it is most certainly through a plug-in or collection of plug-ins. This may make it difficult, for example, to render a document containing nested diagrams and mathematics.

The W3C Amaya browser [16] does support the three markup languages (XHTML, MathML and SVG), but documents written in Amaya may not work in other web browsers.

Native support and full integration of such specifications will be of great benefit to all members of Web community. Support for the XHTML, MathML and SVG profile[4] from W3C[1] might solve the problem, but the specification is still under development and native implementations are not available. In this paper we address the problem in a different way, by transforming documents written in this profile to SVG.

The paper gives a brief explanation of the work which was carried out to explore this approach through an attempt to partially implement the XHTML, MathML and SVG profile. Users can use the resulting software to write mixed documents containing the three languages + XFORMS and having the result rendered as output in SVG format.

2. SVG as an output platform

Documents stored in a computer need to be presented visually to users. This could be done on a printer or some form of Graphical Output Device (e.g. SVGA for PC computers). These devices are still the most popular way to produce visual output from any sort of stored information. SVG can be used for a similar purpose. There is an analogy with display Postscript which was used as an intermediate language in the NeWS windowing system. Instead of directly filling pixels on the screen to draw in ordinary Graphical devices, one can use SVG drawing primitives to capture a description of the output at a higher level of abstraction using the primitives path, text and image.

Batik [10] is one example which uses SVG as an Output Device. Batik provides an SVG toolkit for Java applications that uses the SVG format for display. The package is called SVGraphics2D. (See http://xml.apache.org/batik/svggen.html for more information). One of the benefits of using SVG as an output platform is accessibility. Users can zoom in/out of a display easily for a more convenient view.

3. Benefits of using SVG for output

Most of the W3C recommendations have agreed to use CSS[17] for styling. Using SVG to render XHTML benefits from the fact that both use CSS for styling. For instance to implement 'letter-spacing', the only work required is to produce a 'text' element in SVG with a 'letter-spacing' attribute on it. This common approach to styling within W3C recommendations made displaying XHTML and MathML in SVG much simpler. This is also the case for CSS inheritance rules.

4. How to integrate with SVG

There are three possible way to render XML documents through SVG. Two of these approaches are quite similar and the third uses a different techniques. Documents containing languages such as XHTML and MathML need to be laid out before being displayed. The laying-out process should be applied to the markup language using a convenient programming language such as Java. After the document is formatted SVG elements are then generated to represent the output that should appear. The part of the software which does the layout and the display can be located in one of the following places:

1. Server side
2. Client side

As shown in Figure 1 it is possible to have a proxy-kind-of-software (called here, SVG Gateway) to be used as a gateway connected online to the web. SVG clients could request documents (XHTML, MathML, etc) from anywhere on the web through this gateway. The gateway then requests the documents on the client's behalf from the web and converts it to SVG and sends it back to the client (Figure 1- C2).

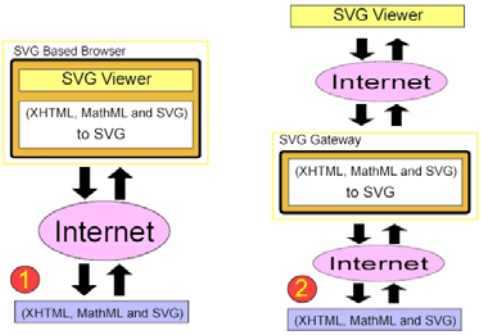

Figure 1. Client side (left hand figure) and gateway architectures (right hand figure)

The Client side approach is similar. If the client has the capabilities such as processing power and memory there will be no reason for having a proxy to handle the conversion process. The client could request web pages and do the conversion locally. See Figure 1- C1.

The last technique which can possible be used is to write the markup code directly inside an SVG document using Namespaces[18]. In this case the foreign elements (XHTML, MathML) could be handled by a scripting language such as JavaScript. JavaScript then has to do the formatting and display them by generating SVG elements dynamically. This approach shows how powerful JavaScript can be and also it shows how SVG is a potential graphical environment and has show benefits beyond even what the W3C SVG working group appear to have considered so far.

5. Integrating MathML modules with XHTML

According to the new recommendation of W3C, MathML expressions should be treated as inline elements when written in XHTML context. To achieve that the overall result of the layout process over the MathML markup should have similar attributes to that of XHTML inline elements such as baseline. Knowing the baseline of the Math expressions makes it possible to align them relative to the surrounding XHTML elements. This will be explained further later in the paper.

6. Handling mixed documents of XHTML, MathML and SVG

The basic approach we have taken is shown in Figure 2. A SAX parser is used to read the mixed document in the first step in the layout process. From there a tree is to be built (the elements tree), where the XHTML, MathML and SVG elements are nodes/leaves of the tree. From this tree, another tree (the areas collections tree) is created where the nodes represent areas. All the nodes in the new tree will have width and height and properly position the information to be displayed. This work is carried out by the Layout Manager.

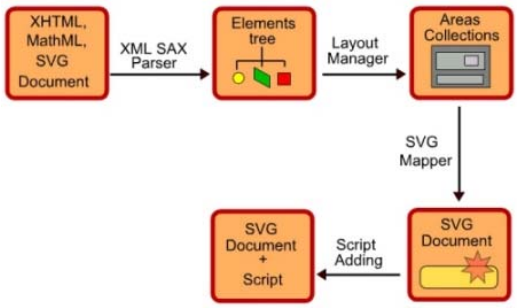

Figure 2. Layout manager architecture

From the area elements and by knowing their content appropriate SVG code is generated. The generated code is what we see on our screens. JavaScript code might also be added at final stage to provide for interaction with the output (such as displaying the contents of the 'alt' attribute of XHTML Images when the mouse cursor moves over the image).

6. Software architecture

The structure of the software is divided into two parts. Part one which called the PROXY is responsible for accepting users requests for documents of type XHTML,

MathML , SVG or even a mixed document of the three. Control should be then passed to part two of the software (CONVERTER), after downloading the required document from the web. The CONVERTER processes the document and produces an SVG document to be sending back to the client by the PROXY.

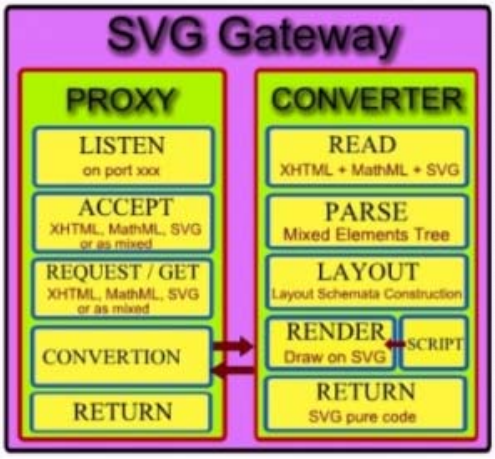

Figure 3. Software architecture

6.1. PROXY

As shown in Figure 3, there are five stages to handle any document request from a client. The PROXY first starts to listen to a certain port waiting for requests. If a request arrives the PROXY accepts it and downloads the document from the web on the client's behalf. The downloaded document then has to be passed to the CONVERTER and the PROXY waits for the conversion to be done on the document and accepts a pure SVG document back from the CONVERTER.

7. SVG, MathML and XHTML Presentation models

XHTML, MathML and SVG use three different presentation models. But in order to display them together a common model should be defined. The MathML Layout Schemata Model has been chosen as the Layout Model for this mixed document (XHTML, MathML and SVG). Our view is that the CSS presentation model is not mature enough to handle such document unless we use CSS only to handle the SVG and XHTML components and leave MathML

Figure 4 shows some common features between CSS Box model and MathML Layout Schemata Model which the design of this work benefits from.

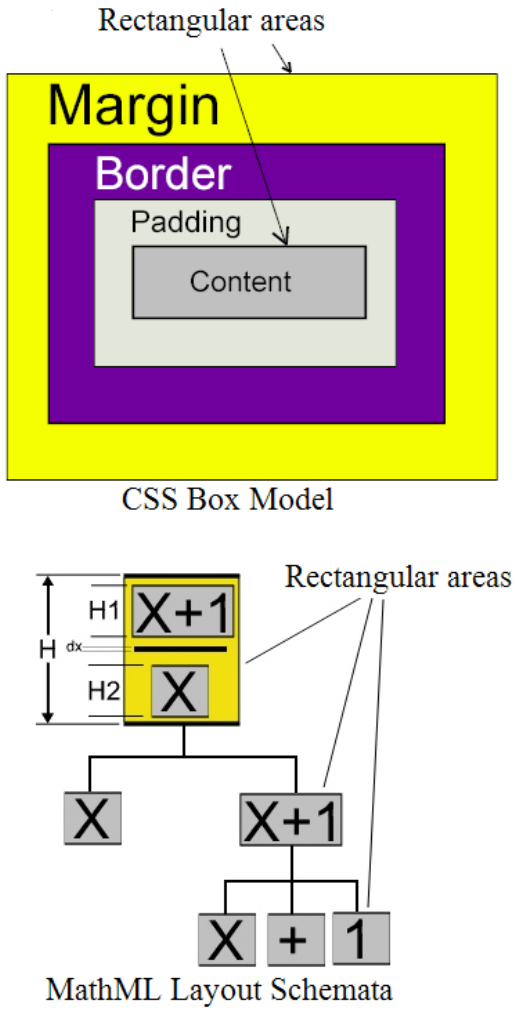

Figure 4. Common features of CSS and MathML models

SVG on the other hand uses a less restricted model. But since SVG elements are to be mixed with XHTML and MathML elements then some restrictions have to be applied on them.

8. Mixed document layout model

In order to solve the differences between the presentation models of XHTML, MathML and SVG our approach uses a so-called Box Model. It is similar to the W3C CSS Box model but a little simpler. As shown in Figure 5 The box model is just a rectangular area with a horizontal or vertical baseline. Every element in the mixed document is to be treated as such (i.e. characters, words, mathematical expressions, rectangles, circles, images, etc). The baseline is used for alignment.

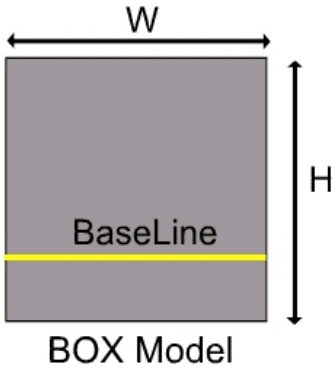

Figure 5. The Box Model presentation model

The box model is easily applied to XHTML and MathML since it is similar to their presentation models but it is not compatible with SVG. Applying this new layout model to SVG elements effectively is a remodeling of SVG. All SVG elements then are to be enclosed in this invisible rectangular area and they would have a baseline. Having the SVG elements with baselines solves the problem of alignment to the possible surrounding XHTML and MathML. According to the W3C XHTML, MathML and SVG profile, SVG elements could have content inside them. This content is to be enclosed in their inner area as shown in Figure 6.

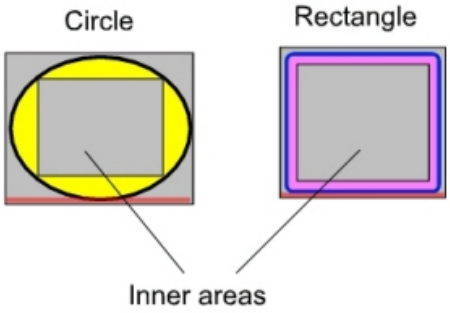

Figure 6. SVG inner areas

One can have a whole XHTML document inside an SVG rect element. However the way the content in this inner area could be treated differently depends on the behavior of the element. Here is a list of possibilities:

1. Normal: No actions are to be taken. The content could exceed its parent's height (SVG element).
2. Resize: in this case the related SVG element should be resized to enclose its content and that could be done in the following way:
    - If the SVG element supports the height attribute then its height should be changed.
    - In other cases, the SVG element should be re-scaled.
3. Stretch: in this case the content should be scaled to fit within the SVG element inner area.
4. Clip: any content exceeding the height of the object is to be clipped.
5. Scroll bar: This works as web browsers do to control content with a height larger than the screen view.

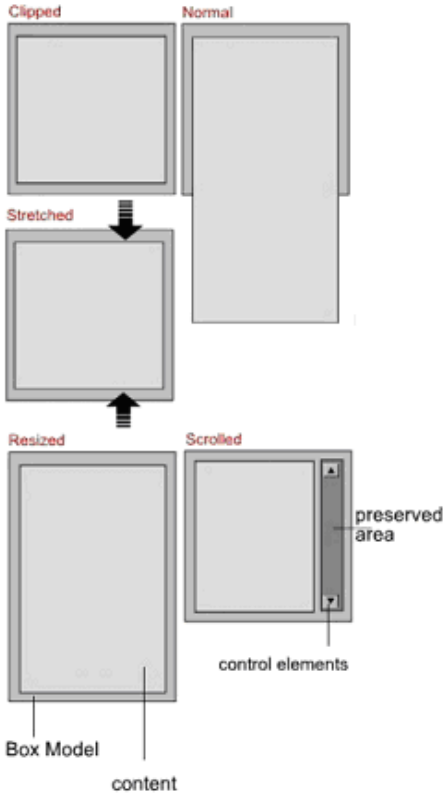

Figure 7. Possible treatment of content within inner area

9. Implementation of the New Layout Model

An Object Oriented Design approach was used to design the software and the new document presentation model. Two separate implementations were developed for the Layout Model of this work. One was written in Java and the other one in JavaScript. Both used the same software architecture and the object structure.

9.1. Box class

This class is at the heart of the implementation (Figure 8). It represents the Box Model of the new layout model. It is also the super class of all classes in the software.

This object was identified based on the requirement of the MathML rendering engine at first and then the requirement of the new layout model overall. The Box class has different attributes (properties) and methods (behaviors) to serve different purposes. However some of the object methods work more as templates where other classes override them to do more sophisticated work. Figures 8-1 and 8-2 shows how the box class is used to implement some of the MathML elements. A similar approach was followed to implement XHTML and SVG.

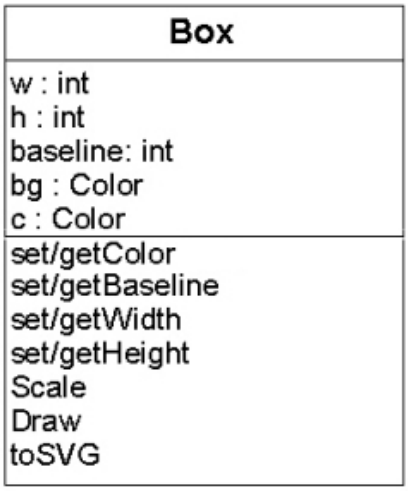

Figure 8-1. Class diagram for the Box model

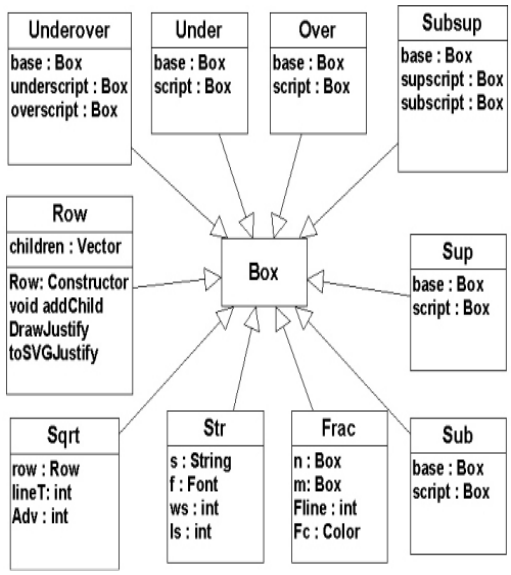

Figure 8-2. Class diagram of MathML elements

10. Implementation in JavaScript

By using the power of SVG and its associated DOM, it was possible to implement a similar layout engine in JavaScript based on the Box Model. That was possible was also due to the fact that JavaScript supports OOP but in a very simple way. Since the SVG DOM can be used to deal with foreign elements using DOM commands such as getElementByTagNameNS, getNamedItemNS, createElementNS, there was no reason not to write XHTML, MathML markup inside the SVG document itself. The benefit of doing this is that the SVG document can be used to display the XHTML, MathML alongside with SVG elements using the appropriate JavaScript Layout Engine for them. They also can be integrated as mentioned before (for instance XHTML code inside SVG elements). A sub tree of the foreign code (XHTML, MathML, XFORM) can be extracted from the SVG DOM and converted to SVG elements after the layout process is done. Figure 9 shows the result of inserting mixed code in an SVG document using the JavaScript Layout Engine developed by the author.

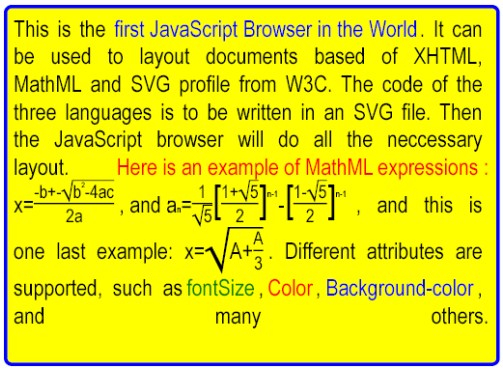

Figure 9. Mixed language code rendered with JavaScript layout engine

The actual mixed code of Figure 9 is showed below. The code is written in an SVG file.

Source document corresponding to Figure 9

```
<?xml version="1.0">
<!DOCTYPE svg PUBLIC - - - - - >
<svg onload="initGraphics(); initSwing();
initBrowser(evt);"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:math="http://http://www.w3.org/Math"
xmlns:xhtml =
"http://http://www.w3.org/XHTML">
<script xlink:href="Graphics.js"/>
<script xlink:href="swing.js"/>
<script xlink:href="Browser.js"/>
<rect class="Resize" x="10" y="10"
width="400" height="205">
```

```
<switch>
<foreignObject>
<xhtml:html>
<head> <title>JavaScript Browser!</title>
</head>
<body>
<p style ="text-align:justify"> This is the
   <em color="blue">first JavaScript   - - - - - -
- - - - - - - - - - - - - - - - - - - -   : </em>
 , and this is one last example:
<math:math>
  <mi>x</mi>
   <mo>=</mo>
    <msqrt>
     <mi>A</mi>
      <mo>+</mo>
      <mfrac>
      <mi>A</mi>
      <mi>3</mi>
    </mfrac>
  </msqrt>
</math:math>. Different attributes are
supported,
such as <em style="fontSize:16;color:green">
fontSize </em>,     - - - - - - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - --
and many others.</p>
 </body>
</xhtml:html>
</rect>
</svg>
```

---

11. Achievement

As explained earlier, This work was an attempt to implement the XHTML, MathML and SVG profile. Support for Xform has also been provided; this does not raise any particular problems in terms of our architecture and is not described further in this paper. The work succeeded to implement a layout engine to handle such mixed document. The document flow is similar to XHTML. It starts from top to bottom, left to write. The integration between SVG elements and XHTML, MathML and XFORMS was also similar to what W3C suggested in the XHTML, MathML and SVG profile. Figure 9 Shows an SVG rect element and inside it an XHTML paragraph with some MathML expression. A limited number of MathML, XHTML, SVG elements is supported, since the goal of this work was not to implement fully functional Layout Engine but rather to show the power of displaying such mixed document in SVG and to demonstrate the viability of the approach.

12. Problems

Most of the problems faced in the development of this work using both languages (Java and javaScript) are Font related. Java font support for fonts was not sufficient to implement MathML in its full potential for instance. Using a powerful, publicly available, font systems such as FreeType [6] would add great value to the overall work.

A lack of Font support in JavaScript/DOM made it difficult to carry-on the work too. There was no way, for example, to acquire information related to a font used in the document in JavaScript from inside an SVG document. Important Font information such as baseline is vital to implement any text layout engine.

13. Further work

The development of the XHTML, MathML and SVG profile within W3C is still at an early stage. However here is a list of possible future work based on the achievements described in this paper:

1. Support more elements from XHTML, MathML and SVG, in order to meet W3C XHTML, MathML and SVG profile.
2. Use of more complete Font system in Java and finding better ideas (if any) to calculate Font baseline in Java Script.
3. Include some SMIL animation for interaction.
4. More support for CSS, such as full inheritance between elements and style sheets support.

A lack of Font support in JavaScript/DOM made it difficult to carry-on the work too. There was no way, for example, to acquire information related to a font used in the document in JavaScript from inside an SVG document. Important Font information such as baseline is vital to implement any text layout engine.

14. Acknowledgements

The work described in this paper forms part of a Dissertation submitted in September 2002 in partial fulfillment of the requirements of the MSc Degree in Web Technologies at Oxford Brookes University. The author thanks his supervisors Bob Hopgood and David Duce for their help.

15. References

[1] The World Web Consortium- Web site http://www.w3c.org

[2] SVG , Scalable Vector Graphics(SVG) 1.0 specification, W3C Recommendation - Web site http://www.w3c.org/TR/SVG/.

[3] MathML- Web site http://www.w3.org/Math/

[4] XHTML, MathML and SVG profile - Web site http://www.w3.org/TR/XHTMLplusMathMLplusSVG/

[5] W3C's Editor/Browser (Amaya), The Languages of Thot - Web site http://www.w3.org/Amaya/User/languages.html

[6] Font Engine Software (FreeType 2) - Web site http://www.freetype.org

[7] The Common Gateway Interface (CGI) - Web site http://www.w3.org/CGI/

[8] SchemaSoft (Custard) MathML to SVG - Web site http://www.schemasoft.com/MathML/

[9] The design in implementation of Lout Document Formatting Language (Jeffrey H. Kingston) Basser Department of Computer Science, the university of Sydney 2006, Australia, 27 January 1993.

[10] Appache Batik SVG Toolkit - Web site http://xml.apache.org/batik/

[11] Jazilla - Web site http://jazilla.sourceforge.net/

[12] Mozilla Layout Engine - Web site http://www.mozilla.org/newlayout/

[13] New HTML Layout - Web site http://www.mozilla.org/newlayout/doc/layout.html

[14] Echoing an XML File with the SAX Parser - Web site http://java.sun.com/webservices/docs/ea2/tutorial/doc/JAXPSAX3.html#64190

[15] Basic XHTML - Web site http://www.w3.org/TR/2000/REC-xhtml-basic-20001219/xhtml-basic10-f.dtd

[16] W3C's Editor/Browser (Amaya) - Web site http://www.w3.org/Amaya/

[17] Cascading Style Sheets, level 2 - Web site http://www.w3.org/TR/REC-CSS2/

[18] Namespaces in XML - Web site http://www.w3.org/TR/REC-xml-names/

[19] XHTML™ 1.0 The Extensible Hypertext Markup Language (Second Edition) - Web site http://www.w3.org/TR/xhtml1/