Position statement

Common rendering framework for compound web documents

Authors: Cameron McCormack, Kim Marriott, Bernd Meyer

Background

Our background is in layout and in particular supporting adaptive layout in web documents so that they can adapt to their viewing context, including user needs and preferences and viewing device characteristics. As part of this we are investigating adaptive layout in SVG.

Combining different web document types

Currently there is little mainstream support for web documents which combine different XML applications, such as XHTML, SVG and MathML. While browsers such as Amaya and XSmiles can render such documents, there is no clear recommendation on how different XML applications should be combined extensibly. The current practice seems to be to develop a profile for a particular combination of XML applications without dictating how the different subdocuments should interact. XHTML+MathML+SVG is an example of such a profile. It defines a DTD for these compound documents but little more. Understandably, it is difficult to provide modular combinations of document types using DTDs, since they are not namespace aware. Other schema languages such as XML Schema or RelaxNG are viable alternatives. This, however, is not the main problem. Without defining what it means for one document type to be used inside another, or a mechanism for specifying this extensibly, true web document modularity cannot be realised.

Popular browsers such as Netscape and Internet Explorer have had a mechanism for including non-HTML content into documents for a long time through the use of plugins. While this is extensible, there are a couple of problems with this method. First, it requires the user to download some binary library to be used by the browser for handling the foreign content. For an accessible, cross platform web, this is unacceptable. A different plugin library must be provided for different platforms. Web site authors use plugins which are not available on some platforms which results in pages which are tied to a particular browser or operating system. A second problem with plugins is that they provide only a fixed dimension opaque canvas onto which the library can draw. This can result in objects which seem to be not part of the outer document, and are merely pasted on top at a given position. Another troublesome aspect of plugins is that, with the exception of some Internet Explorer specific trickery, the subdocument must be given at a separate URL. Ideally, document authors should have the freedom to include the foreign namespace subdocuments directly in the containing document or reference them from cexternal URLs as they wish.

A common rendering framework

For arbitrary web documents to be combined without treating subdocuments as second class, we need a specification for rendering each part into the same document canvas. One method for achieving this is to find a common rendering language for any document type to be converted to. XHTML and MathML deliberately do not specify exactly how they should be rendered. SVG, on the other hand, is designed to allow precise specifications of graphical documents and could thus be seen as a reasonable choice for a common rendering language. If a browser can somehow convert any XML document type to SVG then this resulting SVG can be easily included into the rendering tree. If a subdocument can be transformed to SVG then it can participate in the whole document more consistently, allowing effects such as translucency to work as they should.

The proposed RCC mechanism for specifying the rendering and behaviour of foreign namespace elements in SVG documents can be used to handle different document types. RCC works by allowing the document author to specify how a shadow tree can be generated for particular foreign namespace

elements. The shadow tree can contain a mixture of base SVG elements and other foreign namespace elements, which will in turn have their own shadow trees. There should be at two least ways of generating the SVG rendering tree for other XML documents: XSLT and script. While the idea of using XSLT on the client side to transform XML data into a presentation form such as XHTML is a popular one, it has not been enthusiastically adopted by implementors. Script has been the primary method for shadow tree generation in recent SVG drafts and has been used successfully in a preview release of Adobe's SVG viewer. Major XML applicaations like XHTML and MathML are likely to be used often and so the browser could include built-in support for these rather than relying on RCC implementations. Using RCC for handling other document types will provide the modularity that is needed to handle compound web documents. Since RCC will be written as XSLT or script, a single implementation of a document type will work in any browser that supports this framework.

If all parts of a compound web document now generate an SVG rendering tree we have a unified DOM for the whole document. This leads to the beneficial property of having styles be inherited into subdocuments. It also will allow easier scripting and event handling as everything is now in the one document.

As well as rendering subdocuments into the same canvas as the rest of the document, such a framework would also have to consider the layout of each document within its parent document. In XHTML the author has the option of either forcing the subdocument to have particular dimensions (by specifying a width and height on the object element) or allowing the subdocument to choose its own dimensions (by omitting the width and height). If the dimensions specified by the XHTML document do not match those of the subdocument, then, in the case of SVG, the subdocument will simply be scaled to fit those dimensions. Such simple layout logic may not always be sufficient.

One possible layout negotiation technique would be for each subdocument to be able to specify a minimum, desired and maximum width and height (some of these values might be omitted). The subdocument would compute these taking into account the styles inherited from parent documents, as these may affect the eventual size of the subdocument. After the subdocument has provided the browser with these values the parent document will perform its layout to determine the best dimensions to give the subdocument. The subdocument can then lay itself out (by generating its shadow tree) and inform the parent document of its final dimensions. This last step would be necessary if the subdocument provided only a desired width and not a height, as may be the case in, for example, inline text layout in XHTML.

Some pertinent questions about a common rendering framework:

- What should happen if a foreign namespace element is modified when it already has a shadow tree? The latest SVG draft that discussed RCC tended to the view that script which generates the shadow tree in the first place should manually attach event handlers to the foreign namespace element and update the shadow tree accordingly when the event is fired. This method, while flexible, would result in a lot of work for authors writing rendering support for an XML document type. Every author will have to attach these event handlers and update the shadow trees so that they remain synchronised with the foreign namespace element. A declarative method for specifying how the shadow tree should be updated automatically when mutations occur on the element would be better.

- Should there be a declarative method for specifying the events which can be fired from the foreign namespace element? Sometimes an event in the shadow tree, such as clicking on an SVG rect element which forms a button widget, will have a simple correspondance to the button click event which the widget element should fire. More complex events, such as selecting a range of text in a text entry widget with the mouse, will not.

- Is SVG the right base rendering representation? SVG is a reasonably complex language, with features often unnecessary for document rendering, such as raster filters. On the other hand, what if there is a language which requires some features not support by SVG, for example 3D visualisation? Should this framework deal only with paper-like documents (just including text, tables, diagrams, mathematical formulae, etc.)?

- Should the layout negotiation support more than just simple minimum/desired/maximum rectangles? Some dimensions may be preferred over others. Such choices make the layout optimisation problem a lot harder, however. This also applies to non-rectangular areas.
- What should happen if a subdocument cannot be provided with dimensions that lie within its minimum and maximum? Perhaps then the subdocument can dictate whether this will cause the window to have scrollbars, the subdocument to be padded out/cropped or scaling to occur.