Understanding the Vex Rendering Engine

Mohamadou Nassourou
Department of Computer Philology & Modern German Literature
University of Würzburg
Am Hubland
D - 97074 Würzburg
mohamadou.nassourou@uni-wuerzburg.de

Abstract:

The Visual Editor for XML (Vex)[1] used by TextGrid [2]and other applications has got rendering and layout engines. The layout engine is well documented but the rendering engine is not.
This lack of documenting the rendering engine has made refactoring and extending the editor hard and tedious. For instance many CSS2.1 and upcoming CSS3 properties have not been implemented. Software developers in different projects such as TextGrid using Vex would like to update its CSS rendering engine in order to provide advanced user interfaces as well as support different document types. In order to minimize the effort of extending Vex functionality, I found it beneficial to write a basic documentation about Vex software architecture in general and its CSS rendering engine in particular. The documentation is mainly based on the idea of architectural layered diagrams. In fact layered diagrams can help developers understand software's source code faster and easier in order to alter it, and fix errors.

This paper is written for the purpose of providing direct support for exploration in the comprehension process of Vex source code. It discusses Vex software architecture. The organization of packages that make up the software, the architecture of its CSS rendering engine, an algorithm explaining the working principle of its rendering engine are described.

Keywords: CSS, Processing model, Software architecture, Software design.

Introduction

Structured data such as HTML, XML are usually formatted and presented using Cascading Style Sheets (CSS).
Formatting and presenting structured data with style sheets is known as rendering. The technique of rendering HTML, XML with CSS is called CSS rendering engine.
Vex is an editor for XML documents based on the Eclipse platform [3]. It hides the raw XML tags from the user, providing instead a wordprocessor-like interface. Vex uses CSS to style its editor's content. Its layout engine has been documented as an implementation of the W3C CSS box model [4]. However there is no documentation about the architecture and design of its CSS processing model. If Vex had UML and sequence diagrams of its software, refactoring and extending its software would have been easier. Unfortunately those diagrams do not exist. In order to complement Vex documentation and facilitate the task of those who would like to factor and extend its functionality, I found it beneficial to document its software architecture in general and its CSS rendering engine in particular. The documentation is mainly based on architectural layered diagrams ideology.
In fact layered diagrams can help developers understand software's code faster and easier in order to alter it, and fix errors.

This paper is written for the purpose of providing direct support for exploration in the comprehension process of Vex source code.

Following is a description of the Vex Engine comprising rendering and layout engines.

The Vex Engine

The Vex engine comprises rendering and layout engine modules. A rendering engine module transforms graphics elements and text that constitute a XML or web document into a raster that can be displayed on screens or papers, whereas a layout engine module computes the position of the textual and graphical elements to be displayed on a given page.
A CSS processing model is part of the rendering engine module. However a CSS processing model could be split between the two modules.

Fig.1. shows the diagram of the Vex engine.

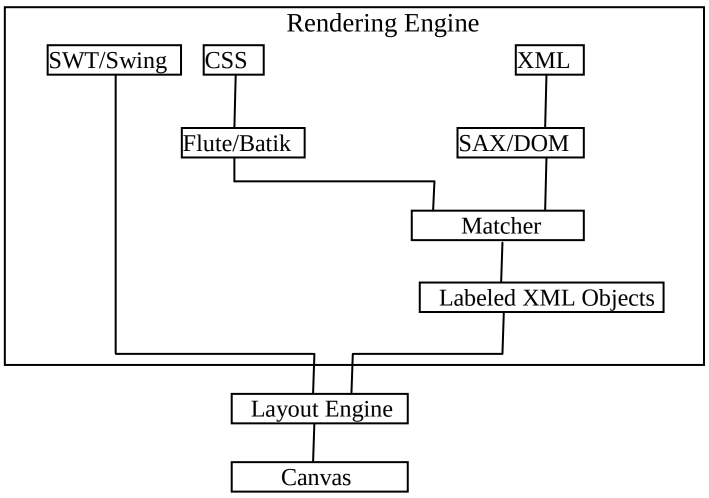


Fig.1. Architecture of Vex Engine

Batik [5] was first used to parse CSS files, and then replaced with Flute [6], because Batik seems to have problems parsing some CSS rules where Flute does it without difficulty.
Flute and Batik parsers do not present Tokenizers or Lexers separately, but rather consider them as modules of the parser. However this approach is undesirable as far as maintenance and reuse are concerned.
SAX is used to generate an object model for XML documents.

The task of the Matcher is to find out which CSS properties have to be applied to which XML element. It does that by comparing CSS selectors with XML tags.

The Layout Engine draws XML elements on the Canvas by mapping the Labeled XML objects to SWT graphics primitives.
It might be important to notice that making the mapping process within the Layout Engine increases the difficulty of refactoring and reuse. Therefore I would suggest a modification of the Vex Layout Engine.

The Vex Layout Engine

The purpose of the Vex Layout Engine is to create a visual representation of a document given a CSS stylesheet. This visual representation is a nested hierarchy of rectangular boxes, implemented as a tree of objects. There are two main types of box. Block boxes containing other boxes and stack their children vertically. Inline boxes whose children are stacked horizontally and are splittable to wrap content into a series of lines.

The Vex Rendering Engine

The rendering engine consists of a processing model. The processing model produces labeled XML objects made of XML elements with their corresponding stylesheets.

The working principle of the Rendering Engine is explained with the following algorithm.

1. First parse the CSS file using Flute/Batik/… which implements the SAC recommendation, Create a HashMap of rules with each one containing triplet of [Selector][property(ies)][value(s)].
2. Then parse the XML document using SAX/DOM/… and create a tree structure of the document. Each element of the tree is made of [Element][attribute(s)][value(s)]
3. Compare each [Element] with each [Selector]
4. If a match is found
5. Check if it is Pseudo-element, if it is not go to step 10
6. Check if ‘content’ property is present, if it is there go to step 9
7. Check if more properties are there, if yes then go to step 10
8. Go to step 3 next element
9. Evaluate value of content property. Eval() is a function that must be implemented to evaluate the value of the content property. For instance counter () function requires counting of some elements
10. Then apply corresponding SWT/Swing/AWT drawing methods for each property (including inherited ones) to the text node of the XML element, and pass the drawing functions with the text node to the Layout Engine for display
11. Repeat steps 3 to 10 till all the elements and selectors are covered

Refactoring and Reuse of Vex Software

After having presented the architecture of the Vex Engine, let's describe and explain how Vex's source code could be modified and reused in other applications.
Generally refactoring and reuse are concerned with modifying software source code in order to improve its readability, maintainability, and facilitate its insertion in other software.

In order to easily factor and extend the source code of Vex, we need first to understand how the packages are organized and linked to each other. There are several tools [7] available for assisting developers in comprehending source code of software. However for someone who is interested in a quick and easy way of understanding Vex with the purpose of partly refactoring or extending its source code, then this paper is the most appropriate tool to consult. Nevertheless a complete graph of the whole software might increase the easiness of maintenance.

Vex's source code is organized in terms of packages. We will discuss about the rendering and layout engine processes. Therefore packages that deal with user interfaces, document source files and types, as well as reference libraries will not be explained.

The packages listed below are just an overview, inline comments of each package and class offer rather more details.

Following are the main directories that will be described:

1. vex-toolkit
2. vex-editor

The vex-toolkit directory

This directory contains classes for all backend functionality, especially the data model and the layout.
It comprises the following packages:

net.sf.vex : this package contains a class that extends and instantiates the Abstract base class (AbstractUIPlugin) for plug-ins that integrate with the Eclipse platform UI.

net.sf.vex.action: this package contains classes that provide command objects that can act on VexWidget.

net.sf.vex.core: this package contains classes

net.sf.vex.css: this package contains classes implementing an object model for CSS stylesheets, the implementation of CSS parsing, as well as implementation of the CSS algorithms for calculating computed style values.

net.sf.vex.dom: this package contains classes implementing an object model for XML documents, and implementation of XML parsing using SAX parser.

net.sf.vex.layout: this package contains classes implementing Vex Layout Engine. It creates a visual representation of a document given a CSS stylesheet. This visual representation is a nested hierarchy of rectangular boxes, implemented as a tree of objects each implementing the W3C CSS box model.

net.sf.vex.swing: this package contains classes that wrap AWT graphics functions, as well as user actions.

net.sf.vex.swt: this package contains classes that implements SWT graphics primitives, as well as the implementation of Vex Widget based on SWT. Therefore it contains the main class (VexWidget) of Vex.

net.sf.vex.undo: this package contains classes that control the do and undo actions.

net.sf.vex.widget: this package contains interfaces and implementation classes based on Swing.

The vex-editor directory

This directory contains the Eclipse interfacing part i.e EditorPart.

net.sf.vex.editor: this package contains classes that create the editor, Vex events, and user interfaces such as content assistants, menus.

net.sf.vex.editor.action: this package contains classes that controls actions on Vex elements by adapting JFace Action to IVexAction instance.

net.sf.vex.editor.config: this package contains classes about configurations items such as document types, styles, registration and identification of configuration sources and listeners.

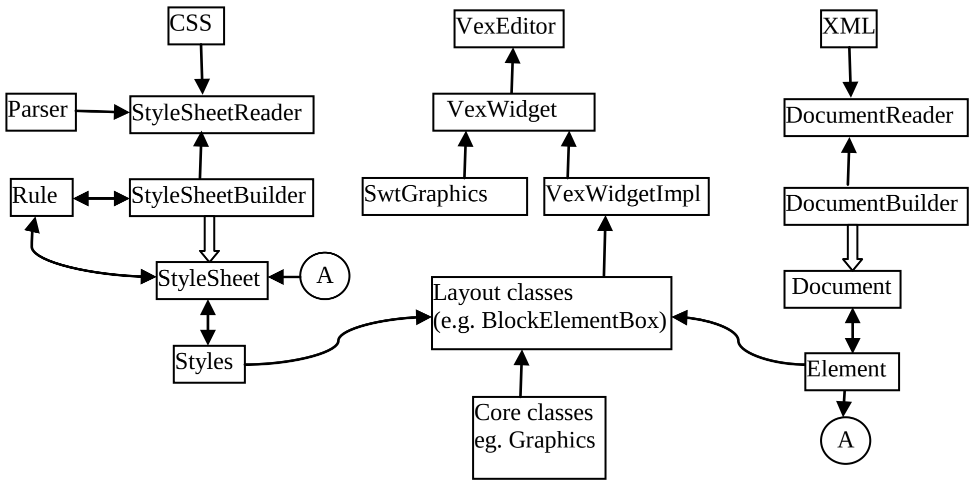
Basic Idea of Vex Rendering and Layout Process

Understanding how vex-toolkit and vex-editor directories communicate with each other is fundamental for acquiring the ability of refactoring, reusing, and extending vex functionality.

There are several tools [7] available for assisting developers in comprehending source code of software projects.

Visual explanation by example is the best way of effectively propagating knowledge. Let’s consider a simple example of drawing a string on the canvas.

The flow view of interaction between most relevant classes is shown below.



To make the explanation of the above diagram easy, let’s make an analogy.

“We want to draw a text or graphic on a canvas according to a given styling format, and show it.”

From this sentence one could try identifying the following elements:

a. who is the presenter or shower
b. who is the drawer
c. what is to be drawn
d. how it should be drawn

The presenter is the vexeditor.java
The drawer is the SwtGraphics.java
The thing to be drawn is the Element.java
The manner to draw is the Styles.java

Now the drawer that is the SwtGraphics.java draws things one after the other according to given rules set by the VexWidgetImpl.java. Therefore there must be someone to transmit piecewise what to be drawn and how it should be drawn. This task is assigned to the Layout classes (e.g. BlockElementBox.java). This means that Layout classes get as inputs the XML element to be drawn as well as the stylesheet to be applied to it.
In fact all the classes in the Layout package will call the stylesheet class which is the generator of the Styles.java class. The Styles.java class holds stylesheets for each XML element. The stylesheets class calls the Rule.java class in order to identify which selector matches XML tags. Based on this match Styles.java class for the identified XML element is instantiated.

The VexWidgetImpl.java is a kind of instructor as well as coordinator to the Layout classes (e.g. BlockElementBox.java).

Now we need to find out the process of generating what to be drawn, likewise the way to draw. Therefore the question is how Styles.java and Element.java are created and instantiated.
The XML source document is loaded by the DocumentReader.java which calls the DocumentBuilder.java to perform the task of importing and instantiating the Document.java. The Document.java is in turn the one that calls and fills the Element.java which is required by the Layout classes.
The DocumentReader.java assists the DocumentBuilder.java by providing it the SAX parser that actually does the parsing and the generation of the XML DOM.
The StyleSheetReader.java loads the CSS file, calls the Flute Parser to tokenize and parse it. It then calls the StyleSheetBuilder.java to generate the StyleSheet.java and instantiate the Rule.java.

The StyleSheet.java will then call and instantiate the Styles.java according to Element.java and Rule.java. In other words the StyleSheet.java generates Styles.java by comparing Element.java with Rule.java.

How to implement new CSS properties in Vex

For instance:

- Implementing generated content such as attr(), counter(), first-line, first-letter functions, and so on.
- Adding image support through the use of <graphic/> element recommended by the TEI [9], and so on.

One might even be in need of implementing new user interfaces such as descriptive content assistant or favorite menu, …etc.

Mastering the previously described Architecture of Vex Engine in Fig. 1 will make the task affordably easy. In fact it facilitates the identification of classes that need to modified or extended with new ones. In other words one should know whether the changes will affect the CSS path or XML path or both simultaneously.

See **Appendix section** for working examples that provide a starting point for refactoring and extending Vex source code with new functionalities. Displaying graphics and adding support for CSS pseudo-elements are discussed in that section.

Discussion and Conclusion

The Vex engine consisting of layout and rendering engines has been explained. The organization of packages that constitute the software, the architecture of its CSS rendering engine, an algorithm explaining the working principle of its rendering engine have been described. Some working examples of extending Vex functionality have been provided.
However this paper is not in itself a sufficient document for completely understand Vex software. Based on this paper any interested person could further the exploration of the software. In fact I have not talked about other packages, and how to add support for different types of documents. This will be part of the future topics on understanding and extending Vex software.

References

[1] http://wiki.eclipse.org/Vex
[2] http://www.textgrid.de/
[3] http://www.eclipse.org/
[4] http://www.w3.org/TR/CSS2/box.html
[5] http://xmlgraphics.apache.org/batik/javadoc/org/apache/batik/css/parser/package-summary.html
[6] http://www.w3.org/Style/CSS/SAC/
[7] Vineet Sinha (2008), PhD thesis: "Using Diagrammatic Explorations to Understand Code", MIT, USA
[8] http://www.w3.org/Style/CSS/
[9] http://www.tei-c.org/Guidelines/P5/
[10] http://en.wikipedia.org/wiki/Simple_API_for_XML
[11] http://en.wikipedia.org/wiki/Document_Object_Model
[12] http://en.wikipedia.org/wiki/Web_browser_engine
[13] http://en.wikipedia.org/wiki/Comparison_of_layout_engines_(CSS)

Appendix

1. How to display image in Vex editor

Changes and additions that need to be made:

Package net.sf.vex.css;

StyleSheet.java

```
/*
add following code in the IProperty[] CSS_PROPERTIES = new IProperty[] {......};*/

new ImageHeightProperty(),
new ImageWidthProperty(),
new BackgroundImageProperty(),
```

Styles.java

```
/*
add following code
*/
   public float getElementWidth() {
                float Image_Width_NORMAL = 50.0f;
                 if(this.values.get(CSS.WIDTH)!=null){
                 return ((Float)this.values.get(CSS.WIDTH)).floatValue();
      }
      return Image_Width_NORMAL;
   }
   public float getElementHeight() {
                float Image_Height_NORMAL = 50.0f;
                 if(this.values.get(CSS.HEIGHT)!=null){
                    return ((Float)this.values.get(CSS.HEIGHT)).floatValue();
         }
      return Image_Height_NORMAL;
   }
   public String getBackgroundImage() {
                String image_loc= this.values.get(CSS.BACKGROUND_IMAGE).toString();
                return image_loc;
   }


/*
create ImageHeightProperty.java class as shown below
*/
public class ImageHeightProperty extends AbstractProperty {
   private static final float Image_Height_NORMAL = 50.0f;
   public ImageHeightProperty() {
      super(CSS.HEIGHT);
   }
   public Object calculate(LexicalUnit lu, Styles parentStyles, Styles styles) {
                if(lu!=null){
                  Float fObj = new Float(Math.max(Image_Height_NORMAL, lu.getFloatValue()));
                  return fObj;
                }
                return Image_Height_NORMAL;
   }
}
/*
create ImageWidthProperty.java class as shown below
*/
public class ImageWidthProperty extends AbstractProperty {

   private static final float Image_Width_NORMAL = 50.0f;
   public ImageWidthProperty() {
      super(CSS.WIDTH);
   }
   public Object calculate(LexicalUnit lu, Styles parentStyles, Styles styles) {
                if(lu!=null){
                            Float fObj = new Float(Math.max(Image_Width_NORMAL, lu.getFloatValue()));
                        return fObj;
                }
                return Image_Width_NORMAL;
  }
}
```

```
/*
create BackgroundImageProperty.java class as shown below
*/
public class BackgroundImageProperty extends AbstractProperty {
   private static final String bImage = "";
   /**
    * Class constructor.
    */
   public BackgroundImageProperty() {
      super(CSS.BACKGROUND_IMAGE);
   }
   /**
    * Calculates the value of the property given a LexicalUnit. Returns the value.
    */
   public Object calculate(LexicalUnit lu, Styles parentStyles, Styles styles) {

           if(lu!=null){
                   if (lu.getLexicalUnitType() == LexicalUnit.SAC_ATTR) {
                           String sObj = new String(lu.getStringValue());
                           return sObj;
                   }
           }
           return bImage;
      }}

/*********************************/
```

Package net.sf.vex.layout;

BlockElementBox.java

```
/*
add following code in the protected List createChildren(LayoutContext context) method, just before "return childList;"

*/
           Styles st = context.getStyleSheet().getStyles(this.getElement());
           if(this.getElement().getName().equals("graphic") && !st.getDisplay().equalsIgnoreCase(CSS.NONE)){
                                   if(this.getElement() !=null) {
                                           this.callDrawImage(context, this.getElement(), st);
                                   }
           }

/*
add the two following methods
*/
           private void callDrawImage(LayoutContext context, IVexElement element, Styles styles){
                   String filename = element.getAttribute(styles.getBackgroundImage());
                   if(filename != null){
                           InlineBox markerInline;
                           markerInline = DrawImage(element, styles, filename);
                           this.beforeMarker = ParagraphBox.create(context, this.getElement(),
                                   new InlineBox[] { markerInline }, Integer.MAX_VALUE);
                   }
           }

           private static InlineBox DrawImage(final IVexElement element, Styles styles, final String filename){
               float w = styles.getElementWidth();
                   float h = styles.getElementHeight();
                   final int height = (int) h;
                   final int width = (int) w;
                   final int offset=5;
                   Drawable drawable = new Drawable() {
                           public Rectangle getBounds() {
                                   return new Rectangle(0, -height, width, height);
                           }
                           public void draw(Graphics g, int x, int y) {
                                   final Image image = new Image(Idisplay,filename);
                                   g.drawImage(image, g.getClipBounds().getX()+offset, g.getClipBounds().getY()-height, width, height);
                           }
                   };
                   return new DrawableBox(drawable, element);
           }

/*************************************/
```

Package net.sf.vex.core;

Graphics.java

```
/*
add following line
*/
    public void drawImage(Image image, int x, int y, int width, int height);

/*************************************/
```

Package net.sf.vex.swt;

SWTGraphics.java

```
/*
add following method
*/
                public void drawImage(Image image, int x, int y, int width, int height) {
                                gc.drawImage(image, x+ originX, y+ originY);
                }
```

(For testing: use <graphic url=”image file location”> in your XML file, and add graphic {width: XYZpx;   height:XYZpx;   display: block; background-image: attr(url);} in your CSS file). The string “url” in <graphic> could be replace with any other string.

2. Adding Support for (TEI P5) Pseudo-elements functions: attr(), counter(), open-quote, close-quote

(The next paper will cover remaining pseudo-elements and classes)

Changes and additions that need to be made:

In vex-sac directory

Package org.w3c.flute.parser

/*
Open **Parser.java, Parser.jj**, and make following modification. Normally making the changes in Parser.jj only should be sufficient.
*/
Add a "(" after counter as shown below. In fact this is an error from Flute itself.

```
} else if ("counter(".equals(f)) {
```

Package net.sf.vex.css;

StyleSheet.java
/*
add following code in bold in the calculateStyles(IVexElement element) method, and extend the class with "GeneratedContentPropertiesImpl" as shown below.
*/

```
public class StyleSheet extends GeneratedContentPropertiesImpl implements Serializable {……
……
        if (element instanceof PseudoElement) {
            if(element.getName().equalsIgnoreCase("before")){
                generateCounterBefore(element);
            }
            lu = decls.get(CSS.CONTENT);
            if (lu == null) {
                return null;
            }
            Sring myStringValue = "";
            List<String> content = new ArrayList<String>();
            while (lu != null) {
                if (lu.getLexicalUnitType() == LexicalUnit.SAC_COUNTER_FUNCTION) {

                    String val = lu.getParameters().getStringValue();
                    if(val.equalsIgnoreCase("div1")){
                      content.add(Integer.toString(mc1));
                    }
                    else if(val.equalsIgnoreCase("div2")){
                        content.add(Integer.toString(mc2));
                    }
                    else if(val.equalsIgnoreCase("div3")){
                        content.add(Integer.toString(mc3));
                    }

                } else if (lu.getLexicalUnitType() == LexicalUnit.SAC_ATTR) {
                    String attributeName = lu.getStringValue();
                    String value = element.getParent().getAttribute(attributeName);
                    if (value != null)
                    content.add(value);
                }
                if (lu.getLexicalUnitType() == LexicalUnit.SAC_STRING_VALUE) {
                    myStringValue = lu.getStringValue();
                    content.add(myStringValue);
                }
                String re = lu.getStringValue();
                if (re != null) {
                    if (re.equalsIgnoreCase("open-quote") || re.equalsIgnoreCase("close-quote")) {
                        content.add("\"");
                    }
                }
                lu = lu.getNextLexicalUnit();
            }
            styles.setContent(content);
        }
```

/* Create the following class */

```
import net.sf.vex.dom.IVexElement;
public abstract class GeneratedContentPropertiesImpl {
                public int mc1=0;
                public int mc2=0;
                public int mc3=0;

        public GeneratedContentPropertiesImpl() {}
        public void generateCounterBefore(IVexElement element){
                if(element.getParent().getName().equalsIgnoreCase("body")){
                        mc1=0;
                        mc2=0;
                        mc3=0;
                }
                else if(element.getParent().getName().equalsIgnoreCase("div1")){
                        mc1++;
                        mc2=0;
                        mc3=0;
                }
                else if(element.getParent().getName().equalsIgnoreCase("div2")){
                        mc2++;
                        mc3=0;
                }
                else if(element.getParent().getName().equalsIgnoreCase("div3")){
                        mc3++;
                }
    }
}
```