Developing Lightweight Multimedia Learning Objects for the Semantic Web

Jinan Fiaidhi, Sabah Mohammed and Marshall Hahn,

Department of Computer Science, Lakehead University,
Thunder Bay, Ontario P7B 5E1, CANADA

Summary

Interactive multimedia elements allow information to be presented in a comprehensible format attuned to the way a students' mind works. A multimedia learning object is defined as an animation that includes a combination of text, graphics, sound, and video packaged together. Unlike the standard lecture mode, learning objects allow flexibility and round-the-clock access by the students. This article develops a method for creating interactive multimedia learning objects based on the SVG standard that is widely used on the semantic web. We refer to our implementation of this method as the learning object presentation (LOP) generator, a tool capable of generating CanCore compliant learning object presentations. This tool has been written using a mixture of Java and JavaScript to make it independent from the platform. The presentations generated by it utilize embedded JavaScript rather than declarative animation for reasons of increased portability, reduced complexity, and file size minimization.

Key words:
Multimedia Learning Objects, SVG, Semantic Web

Introduction

ONE of the more recent developments with the Web is the Semantic Web initiative. The Semantic Web is not a separate Web but an extension of the current one, in which information is given well-defined meaning, better enabling computers and people to work in cooperation [1]. Various scenarios and possibilities which helped the evolution of the Semantic Web are based on the assumption that each object on the web can be described using an XML-like machine-readable form. More precisely, the machine-readable descriptions of Web resources are essential for building advanced Web applications that can process Web information, can reason about this information, and thus can provide better support for interactions with the Web. In fact, the development of the Semantic Web has a great impact on the future of e-Learning. Many achievements have been made in creating standards for Learning Objects. For example, initiatives such as LOM (Learning Objects Meta-data) [2] IMS [3] and Ariadne [4] have been highly successful. These standards shift the focus from the more or less closed e-Learning environments forward to open e-Learning environments, in which Learning Objects from multiple sources (e. g. from different courses, multiple Learning Object providers, etc.) can be integrated into the learning process. Learning objects developed and stored at many different Web locations have a tremendous potential to benefit e-learning. However, numerous technical issues must be dealt with before learning objects can be effectively reused from one situation to the next. Given the context in which learning objects can be reused on the Semantic Web, a hotbed of activity has emerged around how XML content formats and XML-based metadata systems can support e-learning and instructional technology on the Web. The XML-based learning objects should represent complex open object which contains, or refers to, physical learning resources capable of being rendered in multiple display formats, with links to one or more metadata specifications, and perhaps links to other related learning objects [5]. Ultimately, the conceptual learning object will contain links to one or more ontologies that provide sufficient information for reusing the learning object in different contexts.

However, describing an open learning object that involves multimedia can be achieved using several available standards (e.g. MPEG4, MPEG7, MPEG-21, XHTML, SMIL and SVG). The usage of MPEG standards proved to be more complex that the other standards [6]. Realizing the complexity of the MPEG-s scene description, the other standards took the lead in the area of developing light weight scene description [7,8]. Actually, by combining SVG with existing Web technologies like HTML, SMIL, JavaScript, DOM (Document Object Model), and Java, developers can create extremely rich lightweight open learning objects [9].

This paper develops an approach to the problem of generating SVG-based learning objects for composite mixed-media digital objects by appropriately combining

Manuscript revised by November 25, 2006.

and exploiting existing techniques useful for both client and server side manipulations of these SVG objects. Two interesting combinations are particularly useful, the SVG-JavaScript at the learning object client side and the SVG-Java at the learning object server side. On one hand, the new Web browsers supporting the DOM (Document Object Model), the use of some powerful client-side scripting, such as JavaScript will add a lot to the manipulation and interactivity of the SVG DOM structures [10]. On the other hand, the processing of SVGs at the server side requires the use of Java. With Batik API [11], SVG source code can be transferred to a dynamic DOM structure by reading it from a URI, an InputStream, or a Reader - using the SAXSVGDocumentFactory. The following example illustrates how to create an SVG document:

```
import java.io.IOException;
import
org.apache.batik.dom.svg.SAXSVGDocumentFact
ory;
import
org.apache.batik.util.XMLResourceDescriptor
;
import org.w3c.dom.Document;

try {
  String parser =

XMLResourceDescriptor.getXMLParserClassName
();
  SAXSVGDocumentFactory f = new
  SAXSVGDocumentFactory(parser);
  String uri = "http://...";
  Document doc = f.createDocument(uri);
} catch (IOException ex) {
   // ...
}
```

Moreover, the Batik API provides several ways to use an SVG DOM tree. Two modules can be immediately used to render the SVG document.: (1) the `JSVGCanvas` which is a swing component that can display SVG document, and (2) the `ImageTranscoder` which is a transcoder that can take a URI, an InputStream or an SVG DOM tree and produces a raster image (such JPEG, PNG or Tiff). Since most of the learning objects on the Web take the form of visual presentations (e.g., slideshows), including those used in educational contexts such as distance learning courses, we will focus on generating SVG-based learning object presentations.

2. Describing SVG Learning Object Presentations

There are several applications that can be used to create slide presentations, but all of them have several disadvantages that would be interesting to avoid, like needing a special viewer[12]. Therefore, the presentations produced by using these tools have a limited portability. The utilization of standard and open format documents can solve this problem. SVG, an animation-capable vector graphics format, can be used to represent such documents. However, producing a coherent SVG document representing a complex presentation, likely composed of a large sequence of slides, is not an easy task. Indeed, simple SVG editors (e.g. Amaya, EvolGrafiX XStudio 6, Inkscape Open Source Editor, Jasc WebDraw, Sketsa , Sodipodi) cannot be used to generate slide presentations [13]. Problems will be encountered even with professional SVG presentation generators like [14]:

1. Dojo Slide Toolkit (http://dojotoolkit.org/).
2. JackSVG(http://titanium.dstc.edu.au/xml/jacksvg/index.shtml).

Such presentation editors do not adhere to the standards used to describe learning objects metadata. Moreover, the presentations they generate utilize declarative animation elements such as the set, animate and mpath elements that are not well supported by a number of SVG rendering implementations such as the Apache Batik. Moreover, our reliance on JavaScript over declarative animation can reduce the size and complexity of presentation documents.

This section introduces a possible structure for the SVG document, and presents an interactive tool to create and edit the SVG slide presentation learning object or LOP for short. This tool adheres to the CanCore metadata standard [15] and it has been written in Java to make it platform independent. The CanCore application profile consists of eight main categories, 15 "placeholder" elements that designate sub-categories, and 36 "active" elements for which data are actively supplied in the process of creating a metadata record (Figure 1).

1 general	3.1 identifier	5.7 typicalagerange
1.1 identifier	3.2 catlogentry	5.11 language
1.2 title	3.2.1 catalog	**7 relation**
1.3 catalogentry	3.2.2 entry	7.1 kind
1.3.1 catalog	3.3 contribute	7.2 resource
1.3.2 entry	3.3.1 role	7.2.1 identifier
1.4 language	3.3.2 entity	7.2.3 catalogentry
1.5 description	3.3.3 data	7.2.3.1 catalog
1.7 coverage	3.4 metadatascheme	7.2.3.2 entry
2 lifecycle	3.5 language	**9 classification**
2.1 version	**4 technical**	9.1 purpose
2.3 contribute	4.1 format	9.2 taxonpath
2.3.1 role	4.2 size	9.2.1 source
2.3.2 entity	4.3 location	9.2.2 taxon
2.3.3 date	**5 educational**	9.2.2.2 entry
3 metametadata	5.6 context	9.4 keyword

Figure 1: Overview of CanCore Metadata elements.

An LOP is generated based upon an XML description shown in Figure 2. All data describing the presentation is contained within the <ss:presentation> tag. In this direction, the CanCore metadata describing the presentation is placed within the <ss:cancore> tag. We may set some global properties for the design of the learning object presentation. For this purpose, we can place such properties under the <ss:properties> tag. For simplicity our learning object presentation generator includes one property that allows us to change the transition type. Other properties such as font color and font size can be easily added to our learning object presentation generator.

```
<ss:presentation
xmlns:ss='urn:SLIDESHOW:0-395-36341-6'>
<ss:cancore> ... </ss:cancore>
<ss:properties>
  <ss:transition             type="fade"
duration="1000" frames="20"/>
</ss:properties>
<ss:slide delay="">
  <ss:titlebox>
    <ss:title>The Title</ss:title>
    <ss:subtitle>Slide 1</ss:subtitle>
  </ss:titlebox>
  <ss:bodybox>
    <ss:point>
      <ss:text>The Text<ss:text>
      <ss:point>...<ss:point>
      ...
    </ss:point>
      ...
  </ss:bodybox>
  <ss:images>
   <ss:image path="picture.jpg" />
  </ss:images>
</ss:slide>
  <ss:slide delay="">
    ...
  </ss:slide>
  ...
</ss:presentation>
```

Figure 2: The XML LOP Input File.

The <ss:slide> tag can be used to describe a slide. The slide that will be displayed first in the slideshow should appear at the top of the document, the second slide next, and so on. The delay attribute specifies how long the slide should be displayed if the slideshow is in “play mode”. Each slide must have a titlebox and may have one bodybox and\or one image.

The input XML description will be transformed into a LOP with the document structure shown in Figure 3. The output LOP will consist of the following components: a copy of the original input XML document, embedded JavaScript, and graphical elements defining the controls and slides of the presentation. An example LOP slide is shown in Figure 4.

```
<svg ...>
  <ss:presentation
    xmlns:ss="urn:SLIDESHOW:0-395-
36341-6">
    ...
  </ss:presentation>
  <script type="text/ecmascript" ...>
    <![CDATA[
      ...
    ]]>
  </script>
  <g id="controls" …>
    <g id="prev" ...> ... </g>
    <g id="next" ...> ... </g>
    <g id="play-pause" ...> ... </g>
  </g>
  <g      id="slide1"      delay="3000"
opacity="0">
    ...
  </g>
  <g      id="slide2"      delay="3000"
opacity="0">
    ...
  </g>
  ...
</svg>
```

Figure 3: The Overall Structure of SVG Learning Object Presentation.

The opacity attribute of a <g id=”slide#”> element applies to all child elements of the slide. This attribute allows the presentation runtime script to display one slide at a time rather then all of them at once. Each <g id=”slide#”> element may contain the following elements:

1. A <g id=”titlebox”> element used to group together all elements needed to render a slide’s titlebox.
2. A <image> element used to render a base64 encoded jpg image.
3. A <g id=”bodybox”> element used to group together elements needed to render a slide’s bodybox.

The most complex of the above three slide components is the bodybox. The <g id=”bodybox”> element’s <rect> child defines the box that all text will be displayed within. All other children are <g id=”textgroup#”> elements. LOP generator has a point wrap algorithm that is used to split all points in a bodybox amongst a number of <g id=”textgroup#”> elements, which are used to group together a number of points represented by <text> elements. In addition, it uses a word wrap algorithm to split a point amongst a number of lines. Each line of a point is contained within a <tspan> element. The opacity attribute of a <g id=”textgroup#”> element is of particular importance. Using this attribute, the presentation runtime can selectively choose which textgroup to display. Each <g id=”textgroup#”> element has a delay attribute. The sum of all textgroup delays should equal the delay of the slide. LOP generator has an algorithm designed to determine what percentage of a slide’s total delay will be assigned to each textgroup.

```
<g id="slide4" delay="3000" opacity="0">
  <g id="titlebox"
transform="translate(50,10)">
    <rect x="0" y="0" width="924"
height="130" ⋯/>
    <text fill="white" x="178.35" font-
size="65" y="65">
        Introduction to Java </text>
    <text fill="white" x="261.95" font-
size="40" y="120">
      Java is Cross Platform </text>
  </g>
  <image x="50" y="160" width="924"
height="250"

xlink:show="embed"xlink:href="data:;base64,
/9j/4AAQ
  SZJRgABAQEASABI   ...
BRRRQUAFFFFRRRQB//"/>
  <g id="bodybox" opacity="1"
transform="translate(50,425)">
    <rect x="0" y="0" width="924"
height="275" ⋯/>
    <g id="textgroup1" delay="2250"
opacity="1">
      <text fill="white" x="50" font-
size="40" y="15">
        <tspan x="50" dy="40">
            Compiles to machine-independent
bytecode
        </tspan>
        <tspan x="50" dy="40">
            that runs on: </tspan>
      </text>
      <text fill="white" x="100" font-
size="40" y="95">
        <tspan x="100" dy="40"> Windows
</tspan>
      </text>
      <text fill="white" x="100" font-
size="40" y="135">
        <tspan x="100" dy="40"> MacOS
</tspan>
      </text>
      <text fill="white" x="100" font-
size="40" y="175">
        <tspan x="100" dy="40"> Linux
</tspan>
      </text>
      <text fill="white" x="100" font-
size="40" y="215">
        <tspan x="100" dy="40"> and more
</tspan>
      </text>
    </g>
    <g id="textgroup2" delay="750"
opacity="0">
      <text fill="white" x="50" font-
size="40" y="15">
      <tspan x="50" dy="40">
          Java has a portable graphics
library </tspan>
      </text>
      <text fill="white" x="50" font-
size="40" y="55">
        <tspan x="50" dy="40">
            Java avoids hard-to-port
constructs </tspan>
      </text>
     </g>
   </g>
  </g>
```

Figure 4: An Example of an LOP Slide.

3. LOP Runtime Script Internals

Figure 5 describes the embedded JavaScript used to make a LOP interactive. Class control provides the interface through which graphical SVG elements can communicate with the presentation runtime to achieve tasks such as switching slides, etc. Its overButton method will change the color of a button to green when the mouse is on top of that button, as specified via the onmouseover attribute found on that button's group element. The outButton method will change the color of the indicated button back to blue when the mouse is no longer above the button, as indicated via the onmouseout attribute. Function nextSlide is called when the user clicks the next button as indicated via the onclick attribute of the next button's group element. The prevSlide and togglePlay methods are used in similar ways. The implementations of nextSlide and prevSlide are very simple. They simply call setSlideNow with pSlide, which is an index into a data structure we call the displaylist, either incremented or decremented.

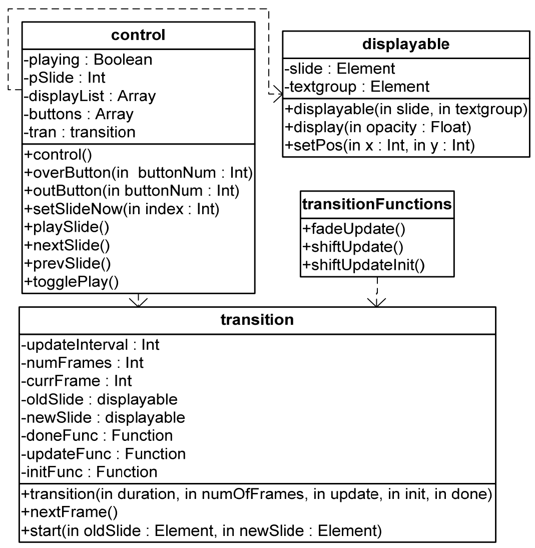


Figure 5: The presentation runtime architecture.

The displayList, an array of displayable objects, is used to define the sequence of slide – textgroup pairs that will be displayed as the presentation progresses, as shown in Figure 6. Using this data structure, the task of displaying one slide-textgroup combination and then another is simplified. For example, to display slide 2, textgroup 1 instead of slide 1, textgroup 1 in Figure 6, the following code could be used:

```
displayList[0].display(0);
displayList[2].display(1);
```

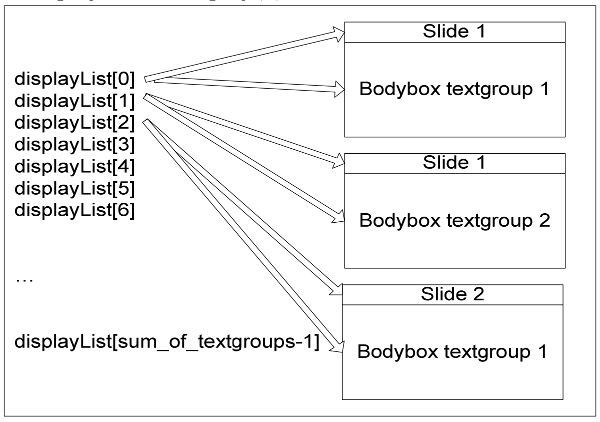


Figure 6: The display list.

Class control utilizes the framework provided by class transition to animate slide transitions. It must be initialized with the following information: how long the transition should take, how many frames the animation should consist of, a function that will be called each frame update, a function that will be called to initialize the transition and finally, a function that should be called when the transition is done. The last argument, doneFunc, is always passed the playSlide function of class control, which will start a timer that upon expiration will cause the next slide to be displayed if the presentation is in play mode. Which update and init functions are passed to the constructor depends on the type of transition desired. Two transition types exist: fade and update. To see how simple the definition of these transitions is, please examine their update function implementations:

```
  function fadeUpdate()
  {
     transition.oldSlide.display(1-

(transition.currFrame/transition.numFra
mes) );
    transition.newSlide.display(

transition.currFrame/transition.numFram
es );
  }
  function shiftUpdate()
  {
    var inc = (docWidth /
transition.numFrames) *
      transition.currFrame;
    transition.oldSlide.setPos( inc ,
0 );
   transition.newSlide.setPos( inc-
docWidth, 0 );
  }
```

To begin animating a transition, the start function of class transition will be called with the old slide's displayable object and the new slide's displayable object as arguments. This function call will originate from the setSlideNow function of class control.

4. The LOP Generator

Much of work performed by the LOP Generator involves the proper placement of text on each slide. The height and length of all text lines must be known for this to be accomplished. Therefore, all text must be rendered so that it dimensions can be retrieved. Necessary adjustments such as those needed to wrap text can then be made. This is the main the concept that lead to the architecture shown in Figure 7. The GUI interface of this utility is provided by class SVGPresGen. Class svgEditor is used to perform the sequence of document transformations needed to produce the output LOM shown in Figure 8. The transformation process begins when SVGPresGen's generateSlideVG method is called. The following steps are involved in its execution:

- An SVG document containing only the Slide Generation Script is loaded into the svgEditor.
- A call to addDocumentAsRootChild will add the input XML document to the SVG document as a child of the root.
- A call to base64encode will create an SVG image element for each image specified in the input XML document. Each <image> element will be placed in the document as a child of its input <ss:image> element.
- Method setDocument of svgCanvas will be called with the SVG document as an argument, causing the document to render. This causes the Slide Generation Script shown in Figure 9 to begin executing.
- When this script is done, it will be replaced with the Runtime Script.
- A call to outputDocument outputs the LOP to a file.

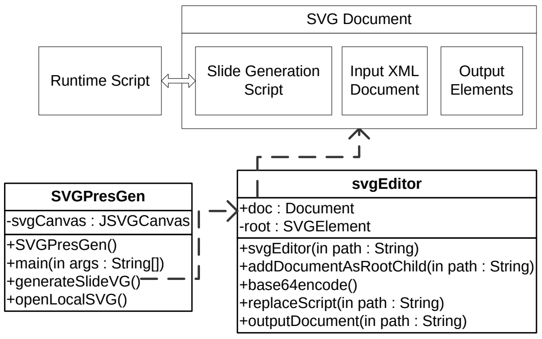


Figure 7: The LOP Generator General Structure.

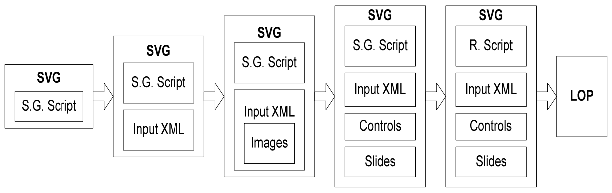


Figure 8: LOP document transformations sequence.

Figure 9 shows the high-level structure of the Slide Generation Script. An object of class presentationFactory instigates the presentation generating process when its constructor is called. Throughout this process, SVG elements will be created using class elementFactory. Class presentationFactory also relies on the processBodyBox function object, which is designed to create a bodybox.

Upon construction, the presentationFactory object will iterate through all <ss:slide> elements found in the input document. For each slide:

1. A slide group element will be created via a call to elementFactory's createSlideGroup method.
2. processImage, processTitleBox, and processBodyBox will each be called in turn. Each of these methods will be passed the slide input data (stored within an <ss:slide> element) as its first argument and the slide group element created in step 1 as its second element.

After all slides have been processed, the scripts job is done. Image processing and titlebox processing are very simple, as an examination of Figure 4 should imply. The most complex processing occurs when a slide's bodybox is created, a task carried out by the processBodyBox function object. This function object is designed to recursively process all input <ss:point> elements.

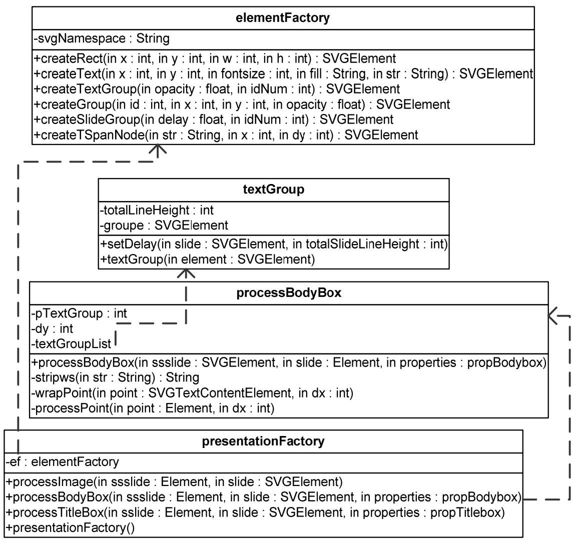

elementFactory
-svgNamespace : String
+createRect(in x : int, in y : int, in w : int, in h : int) : SVGElement
+createText(in x : int, in y : int, in fontsize : int, in fill : String, in str : String) : SVGElement
+createTextGroup(in opacity : float, in idNum : int) : SVGElement
+createGroup(in id : int, in x : int, in y : int, in opacity : float) : SVGElement
+createSlideGroup(in delay : float, in idNum : int) : SVGElement
+createTSpanNode(in str : String, in x : int, in dy : int) : SVGElement
textGroup
-totalLineHeight : int
-groupe : SVGElement
+setDelay(in slide : SVGElement, in totalSlideLineHeight : int)
+textGroup(in element : SVGElement)
processBodyBox
-pTextGroup : int
-dy : int
-textGroupList
+processBodyBox(in ssslide : SVGElement, in slide : Element, in properties : propBodybox)
-stripws(in str : String) : String
-wrapPoint(in point : SVGTextContentElement, in dx : int)
-processPoint(in point : Element, in dx : int)
presentationFactory
-ef : elementFactory
+processImage(in ssslide : Element, in slide : SVGElement)
+processBodyBox(in ssslide : Element, in slide : SVGElement, in properties : propBodybox)
+processTitleBox(in sslide : Element, in slide : SVGElement, in properties : propTitlebox)
+presentationFactory()

Figure 9: The Slide Generation Script.

The processBodyBox constructor gets the process started by passing each top-level point to the recursive processPoints function. Its dx parameter defines the number of pixels the point should be indented. Function processPoint will carry out the following steps to process a point:

- The dx parameter will be incremented by some amount. This is needed to cause indentation of subpoints.
- stripws will be called to normalize the white space found in the text string of the point (more detail on this method later).
- An SVG text element representing the string will be created and appended to the current text group.
- wrapPoint will be called to split the contents of the text element amongst a number of tspan elements. The dx parameter is needed to calculate how much horizontal space is available for each line of the point. The dy instance variable of processBodyBox stores the vertical position of the next line to be drawn. The wrapPoint algorithm will increment it by the font height each time a new line is used.
- If the sum of all line heights exceeds the height of the bodybox, a new textgroup is created. The most recently created point is then moved to this textgroup (bodybox wrap).
- processPoint calls itself to process each subpoint.
- The algorithm will terminate when no subpoints remain.

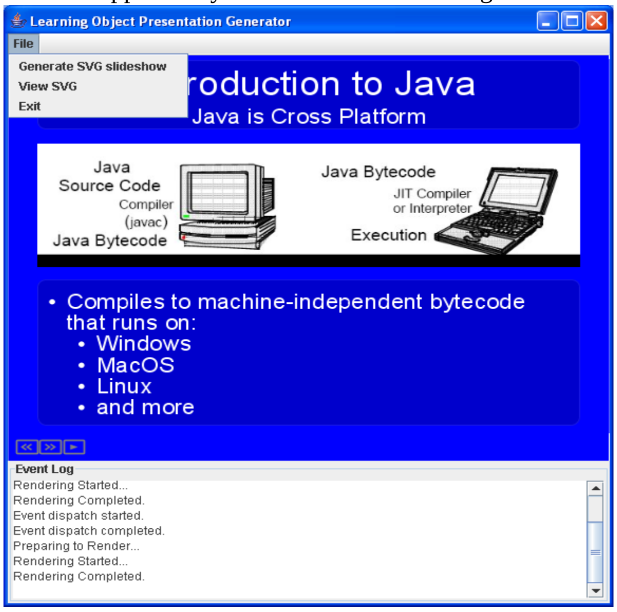
For simplicity, the above description does not include steps needed to calculate the delay each textgroup should receive. Each time a new textgroup is needed, an entry for the new textgroup is created in the textgroupList. We store the height and a reference to the group tag of each textGroup within the textGroup list. After all point processing has completed, this data structure is used to distribute the total delay of a slide across the textgroups belonging to that slide. We define textGroupLineHeight as the number of lines in a textgroup multiplied by the height of the lines (all lines have the same height). We define slideLineHeight as the sum of all text group line heights. Based upon these two variables as well as the slide delay specified by the user, the delay a textGroup should receive is approximately the following:
textGroupDelay = (textGroupLineHeight/slideLineHeight) * slideDelay.

Although the above algorithm is important, it isn't by any means as important as the word wrap algorithm performed by the wrapPoint method. This algorithm assumes that each word in a line of text is separated by no more than one space. This assumption is valid due to the preprocessing performed on all input strings by the stripws method. The stripws method will strip all white space from the front and ends of a text string and all white space between words except for one space. This processing is needed because Apache Batik will render a string of text as if it had been normalized in the way described above. However, when ask for the length of a string, it still returns the length of the non-normalized string, which causes problems for our word wrap algorithm.

The wrapPoint method is passed an SVG text element and the amount a point will be indented as arguments. It relies on the following methods provided by the SVGTextContentElement interface defined in the SVG specification: getComputedTextLength() and getSubStringLength(). The general idea behind the algorithm is as follows. The algorithm begins by calculating the maximum pixel length (mpl) that each line of this point can have based upon the width of the bodybox and the dx parameter. Next, the algorithm checks if the total pixel length of the input string (tpls) exceeds mpl. If no, word wrap is not needed and the algorithm will be terminated. If yes, more processing is required. The algorithm begins iterating over all the characters in the input string. If a single space character is found, the algorithm will check whether or not the length of the string up to the space exceeds mpl. If it does, the string must be broken at the space which occurred before this space. If it does not, search for next space, and so on.

5. Conclusion

For a long time now, it has been known that multimedia learning objects have the potential to support a wider group of learners by providing content in a medium that supports their strengths (e.g. a visual learning style). The difficulty has been that multimedia when used inappropriately can hinder a learner, by possibly providing too complex learning objects. This article presented a tool and a technique for generating multimedia learning objects in the form of a lightweight SVG presentation that enables content experts to easily combine text and images into one synchronized learning object. Users do not need to perform any programming tasks, but instead make use of a graphical user interface to generate the learning object (see Figure 10). The utility adheres to the CanCore metadata and can generate multimedia learning objects that can be easily stored and retrieved by major learning objects repository networks like eduSource (http://www.edusource.ca/). This work is part of an ongoing research at Lakehead to develop an ontology based search engine for multimedia learning objects. This work is supported by the first author NSERC grant.



- Compiles to machine-independent bytecode that runs on:
  - Windows
  - MacOS
  - Linux
  - and more

```
Rendering Started...
Rendering Completed.
Event dispatch started.
Event dispatch completed.
Preparing to Render...
Rendering Started...
Rendering Completed.
```

Figure 10: Displaying a Generated LOP Presentation.

References

[1] Permanand Mohan, Christopher Brooks, "Learning Objects on the Semantic Web," p. 195, Third IEEE International Conference on Advanced Learning Technologies (ICALT'03), 2003

[2] LOM, http://en.wikipedia.org/wiki/Learning_object_metadata

[3] IMS, http://www.imsglobal.org/metadata/

[4] Jehad Najjar, Erik Duval, Stefaan Ternier, Filip Neven, TOWARDS INTEROPERABLE LEARNING OBJECTREPOSITORIES: THE ARIADNE EXPERIENCE, Proc. IADIS Int'l Conf. on WWW/Internet 2003, Vol. I, P. 219-226

[5] Othoniel Rodriguez, Dr. SuShing Chen, Dr. Hongchi Shi, Dr. Yi Shang, Open Learning Objects: the case for inner metadata, The Eleventh International World Wide Web Conference, 7-11 May 2002, Honolulu, Hawaii.

[6] ISO/IEC JTCl/SC29/WG11, "Call for Proposals for Lightweight Scene Representation," MPEG Document N6337, March 2004.

[7] J. FIAIDHI, Guo T. Song, S. MOHAMMED, and Nathan Epp, Developing a Collaborative Multimedia mLearning Environment, 10th western Canadian Conference on Computing Education, WCCCE 2005, May 5-6, 2005, University of Northern British Columbia, UNBC, Prince George, BC, Canada.

[8] MOHAMMED, S and FIAIDHI, J**.,** Developing Secure Transcoding Intermediary for SVG Medical Images within Peer-to-Peer Ubiquitous Environment, IEEE 3rd Annual Conference on Communication Networks and Services Research Conference (CNSR2005), Halifax, Nova Scotia, Canada, May 16 - 18, 2005.

[9] Nicholas Chase, JavaScript and the Document Object Model, IBM Research Journal, 01 Jul 2002 http://www-128.ibm.com/developerworks/web/library/wa-jsdom/

[10] http://www.w3.org/TR/SVG/svgdom.html

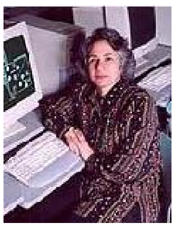
[11] http://xmlgraphics.apache.org/batik/

[12] Anita Petrinjak and Rodger Graham, Creating Learning Objects from Pre-Authored Course Materials: Semantic Structure of Learning Objects — Design and Technology, Canadian Journal of Learning and Technology, Volume 30(3) Fall / automne 2004

[13] **Tobias Hauser,** SVG Editors, SVG Open Conference, Tokyo, Japan · Sept 7-10, 2004

[14] Raul Casado, Juan Carlos Torres, SVG Slide Presentation Editor, SVG Open Conference, Tokyo, Japan · Sept 7-10, 2004.

[15] Norm Friesen**,** Anthony Roberts and Sue Fisher, CanCore: Metadata for Learning Objects, Canadian Journal of Learning and Technology, Volume 28(3) Fall / automne, 2002



Jinan Fiaidhi received her PgD and PhD in Computer Science from England UK (Essex and Brunel Universities) during 1983 and 1986 respectively. She served as faculty member at various universities including Philadelphia, Applied Science, Sultan Qaboos and Lakehead Universities. Since January 2002, she is with Lakehead University, Ontario/Canada. Currently she holds the rank of Professor and the designates of MBCS and I.S.P. of Canada. Her research interests include Learning Objects, Multimedia/Virtual Environments and Peer-to-Peer Learning.

Sabah Mohammed received his MSc and PhD in Computer Science from England UK (Glasgow and Brunel Universities) during 1981 and 1986 respectively. He served as faculty member at various universities including Amman, Philadelphia, Applied Science, Oman HCT and Lakehead Universities. Since January 2002, he is with Lakehead University, Ontario/Canada. Currently he holds the rank of Professor and the designate of MBCS. His research interests include Image Segmentation, Image Protection and Open Source Multimedia.

Marshall Hahn is HBSc fourth year Computer Science student. He is conducting the research in this article as part of his NSERC grant on multimedia programming.